



TRABAJO FIN DE MÁSTER

Máster Universitario en Arquitectura del Software

**Desarrollo de componentes web basados en Angular JS,
Polymer, Node.js, Django y MongoDB para el pre-procesado,
consumo y homogeneización de fuentes Open Data
gubernamentales.**

Autor/a: Juan Francisco Hernández Sánchez

Director/a: David Lizcano Casas

Curso 2015/2016

ÍNDICE

INDICE DE ILUSTRACIONES Y TABLAS	5
RESUMEN	10
ABSTRACT	11
1 INTRODUCCIÓN	12
1.1 Objetivo	12
1.2 Motivación	13
2 ESTADO DE LA CUESTIÓN.....	14
2.1 Open Data	14
2.1.1 Iniciativa Aporta.....	15
2.2 XML.....	15
2.2.1 Aspectos generales de XML.....	15
2.2.2 Ejemplo documento XML	16
2.3 JSON.....	18
2.4 Node.js	20
2.4.1 Arquitectura Node.js.....	21
2.4.2 Módulos en Node.JS.....	22
2.4.3 Ejemplo Hola Mundo en Node.js y express	24
2.5 AngularJS.....	28
2.5.1 Arquitectura.....	28
2.5.2 Características.....	30
2.5.3 Hola Mundo en AngularJS	31
2.5.4 Angular Material.....	34

2.6	MongoDB	34
2.6.1	NoSQL y Documental	34
2.6.2	Comparación MongoDB con MySQL.....	35
2.7	MEAN Stack.....	38
2.8	Django.....	40
2.8.1	Arquitectura Django	41
2.8.2	Principales características de Django	42
2.8.3	Hola Mundo en Django	43
2.9	Polymer	47
2.9.1	Arquitectura.....	47
2.10	Bower	48
3	DESARROLLO	50
3.1	Funcionalidad.....	50
3.2	Formato Datos.....	50
3.2.1	Formato datos obtenidos en XML	50
3.2.2	Formato datos procesados en JSON	53
3.3	Desarrollo NodeJS y AngularJS	54
3.3.1	BackEnd NodeJS	54
3.3.2	Gestión de las URL	67
3.3.3	Front con Jade y AngularJS.....	70
3.3.4	Jade	70
3.3.5	Ventana Login	71
3.3.6	WebApp.....	72
3.4	Desarrollo Django y Polymer	77

3.4.1	BackEnd con Django	77
3.4.2	Settings.py	77
3.4.3	urls.py	80
3.4.4	views.py	82
3.4.5	Front con Polymer	86
3.4.6	Ventana Login	87
3.4.7	WebApp.....	88
4	CONCLUSIONES Y LÍNEAS FUTURAS	95
4.1	Comparativa BackEnds.....	95
4.2	Comparativa FrontEnds	97
4.2.1	Comparativa técnica.	97
4.2.2	Comparativa visual.	97
4.3	Conclusiones.....	100
4.4	Líneas de trabajo futuras.....	101
5	REFERENCIAS BIBLIOGRÁFICAS Y RECURSOS ELECTRÓNICOS .	102
6	AUTORIZACIÓN DE DIFUSIÓN	104

INDICE DE ILUSTRACIONES Y TABLAS

ILUSTRACIÓN 1: LOGO INICIATIVA APORTA	15
ILUSTRACIÓN 2: LOGO NODEJS	20
ILUSTRACIÓN 3: ARQUITECTURA NODE.JS	21
ILUSTRACIÓN 4: LOGO NPM	22
ILUSTRACIÓN 5: LOGO EXPRESS	23
ILUSTRACIÓN 6: LOGO PASSPORT	23
ILUSTRACIÓN 7: LOGO JADE	24
ILUSTRACIÓN 8: EJEMPLO JADE	24
ILUSTRACIÓN 9: EJEMPLO HOLA MUNDO NODEJS	25
ILUSTRACIÓN 10: HOLA MUNDO NODEJS 1	25
ILUSTRACIÓN 11: HOLA MUNDO NODEJS 2	25
ILUSTRACIÓN 12 HOLA MUNDO NODEJS 3	26
ILUSTRACIÓN 13: HOLA MUNDO NODEJS 4	26
ILUSTRACIÓN 14: EJEMPLO HOLA MUNDO NODEJS 5	27
ILUSTRACIÓN 15: EJEMPLO HOLA MUNDO NODE.JS 6	28
ILUSTRACIÓN 16: LOGO ANGULARJS	28
ILUSTRACIÓN 17: ARQUITECTURA ANGULARJS (1)	29
ILUSTRACIÓN 18: ARQUITECTURA ANGULARJS (2)	29
ILUSTRACIÓN 19: HOLA MUNDO ANGULARJS(1)	31
ILUSTRACIÓN 20: HOLA MUNDO ANGULARJS(2)	32
ILUSTRACIÓN 21: HOLA MUNDO ANGULAR JS – CÓDIGO	32
ILUSTRACIÓN 22:HOLA MUNDO ANGULAR JS – CÓDIGO(1)	33
ILUSTRACIÓN 23: HOLA MUNDO ANGULARJS - CÓDIGO (2)	33
ILUSTRACIÓN 24: HOLA MUNDO ANGULARJS - CÓDIGO (3)	33
ILUSTRACIÓN 25: HOLA MUNDO ANGULARJS - CÓDIGO (4)	33

ILUSTRACIÓN 26: LOGO ANGULAR MATERIAL	34
ILUSTRACIÓN 27: LOGO MONGODB	34
ILUSTRACIÓN 28: MEAN LOGO	39
ILUSTRACIÓN 29: MEAN FLUJO	40
ILUSTRACIÓN 30: LOGO DJANGO	40
ILUSTRACIÓN 31: ARQUITECTURA DJANGO	42
ILUSTRACIÓN 32: DJANGO HOLA MUNDO (1)	43
ILUSTRACIÓN 33: DJANGO HOLA MUNDO (2)	43
ILUSTRACIÓN 34: DJANGO HOLA MUNDO (3)	43
ILUSTRACIÓN 35: DJANGO HOLA MUNDO (4)	44
ILUSTRACIÓN 36: DJANGO HOLA MUNDO (5)	45
ILUSTRACIÓN 37: DJANGO HOLA MUNDO (6)	45
ILUSTRACIÓN 38: DJANGO HOLA MUNDO (7)	46
ILUSTRACIÓN 39: DJANGO HOLA MUNDO (8)	46
ILUSTRACIÓN 40: DJANGO HOLA MUNDO (9)	46
ILUSTRACIÓN 41: LOGO POLYMER	47
ILUSTRACIÓN 42: ARQUITECTURA POLYMER	48
ILUSTRACIÓN 43: FORMATO DATOS APARCAMIENTO XML	52
ILUSTRACIÓN 44: FORMATO DATOS APARCAMIENTOS XML	53
ILUSTRACIÓN 45: FORMATO DATOS JSON	53
ILUSTRACIÓN 46: FICHERO PACKAGE.JSON	55
ILUSTRACIÓN 47: SCHEMA BASE DE DATOS	55
ILUSTRACIÓN 48: APP TWITTER (1)	57
ILUSTRACIÓN 49: APP TWITTER(2)	58
ILUSTRACIÓN 50: APP FACEBOOK	59
ILUSTRACIÓN 51: ESTRATEGIA PASSPORT TWITTER	61

ILUSTRACIÓN 52: ESTRATEGIA PASSPORT FACEBOOK	62
ILUSTRACIÓN 53: OBTENCIÓN DATOS (1)	63
ILUSTRACIÓN 54: OBTENCIÓN DATOS (2)	64
ILUSTRACIÓN 55: OBTENCIÓN DATOS(3)	65
ILUSTRACIÓN 56: OBTENCIÓN DATOS(4)	66
ILUSTRACIÓN 57: OBTENCIÓN DATOS (6)	67
ILUSTRACIÓN 58: CONFIGURACIÓN EXPRESS (1)	68
ILUSTRACIÓN 59: CONFIGURACIÓN EXPRESS (2)	70
ILUSTRACIÓN 60: CAPTURA LOGIN	71
ILUSTRACIÓN 61: JADE LOGIN	72
ILUSTRACIÓN 62: VARIABLES INDEX	72
ILUSTRACIÓN 63: MENÚ ANGULARMATERIAL (1)	73
ILUSTRACIÓN 64: MENÚ ANGULARMATERIAL (2)	73
ILUSTRACIÓN 65: MENÚ ANGULARMATERIAL (3)	74
ILUSTRACIÓN 66: JADE MENÚ ANGULARMATERIAL	74
ILUSTRACIÓN 67: CAPTURA WEBAPP	75
ILUSTRACIÓN 68: JADE NG-MAP	76
ILUSTRACIÓN 69: CONTROLADOR ANGULARJS	76
ILUSTRACIÓN 70: SETTINGS.PY INSTALLED_APPS	78
ILUSTRACIÓN 71: SETTINGS.PY TEMPLATES	79
ILUSTRACIÓN 72: SETTINGS.PY STATICS	79
ILUSTRACIÓN 73: SETTINGS.PY AUTHENTICATION	80
ILUSTRACIÓN 74: URLS.PY URLPATTERNS	81
ILUSTRACIÓN 75: URLS.PY STATICFILES	81
ILUSTRACIÓN 76: VIEWS.PY WEBAPP(REQUEST)	82
ILUSTRACIÓN 77: VIEWS.PY GENERATEPATH	83

ILUSTRACIÓN 78: VIEWS.PY GETXML	83
ILUSTRACIÓN 79: VIEWS.PY PARSEARXML(PATH)	84
ILUSTRACIÓN 80: VIEWS.PY GETJSONFROMPARKING	86
ILUSTRACIÓN 81: VIEWS.PY DELETXML	86
ILUSTRACIÓN 82: CARGA FICHEROS ESTATICOS	87
ILUSTRACIÓN 83: VENTANA LOGIN	87
ILUSTRACIÓN 84: VENTANA LOGIN CÓDIGO	88
ILUSTRACIÓN 85: MENÚ	88
ILUSTRACIÓN 86: WEBAPP CÓDIGO (1)	89
ILUSTRACIÓN 87: WEBAPP CODIGO (2)	89
ILUSTRACIÓN 88: CAPTURA WEBAPP	90
ILUSTRACIÓN 89: CONTROLADOR JAVASCRIPT (1)	91
ILUSTRACIÓN 90: PETICION()	91
ILUSTRACIÓN 91: PINTARMARCA(1)	91
ILUSTRACIÓN 92: PINTARMARCA (2)	92
ILUSTRACIÓN 93: PINTARMARCA(3)	93
ILUSTRACIÓN 94: PINTARMARCA(4)	94
ILUSTRACIÓN 95: CAPTURA WEBAPP (2)	94
ILUSTRACIÓN 96: MENÚ ANGULAR	98
ILUSTRACIÓN 97: MENÚ POLYMER	98
ILUSTRACIÓN 98: COMPARATIVA (1)	99
ILUSTRACIÓN 99: COMPARATIVA (2)	100
 TABLA 1: COMPARATIVA XML JSON	 19
TABLA 2: TABLA COMPARATIVA MONGODB Y MYSQL	35
TABLA 3: TABLA MAPEOS CAMPOS	53

TABLA 4: COMPARATIVA BACKENDS.....	95
TABLA 5: COMPARATIVA FRONTENDS	97

RESUMEN

En este trabajo se tratará los diversos métodos de transformar la información obtenida a través de fuentes gubernamentales catalogadas como “Open Data”, y su tratamiento para poder mostrarle esta información de forma amigable al usuario.

Para esta finalidad, se desarrollará la misma aplicación con las mismas funcionalidades e intentando que tengan un aspecto similar en dos tecnologías diferentes. Cuando se comenzó este trabajo no se disponían conocimientos en las tecnologías utilizadas, por ese motivo, durante el desarrollo de este trabajo, se han aprendido las características, ventajas y desventajas de cada una de ellas.

La funcionalidad de esta aplicación es el mostrar en tiempo real el estado de los parkings de una ciudad, añadiendo la gestión de usuarios para poder acceder a ver el mapa.

En la primera versión del desarrollo está hecho con NodeJS y AngularJS en su totalidad. Ambas tecnologías provienen de JavaScript, NodeJS se utiliza en BackEnd para gestionar el servidor, y AngularJS, se utiliza para gestionar el FrontEnd. También se utilizará un login a través de redes sociales, almacenando la información en una base de datos no relacional, como es MongoDB.

La segunda versión del desarrollo está desarrollada, con Django, que es un framework de Python para la gestión de servidores en la parte del BackEnd, y el uso de Polymer para mostrar la información en el FrontEnd. Finalmente, se utilizará el gestor de identidades de Django junto con el login social.

Finalmente, se realizará una comparativa entre las diversas tecnologías para comparar las diversas fortalezas y debilidades de todas las tecnologías utilizadas.

ABSTRACT

This project is about the different ways to transform information obtained from a government source cataloged as “Open Data” and his treatment to display this information in a friendly way.

For this purpose, it will be developed the same application with the same functionality, trying to obtain the same interface in two different tecnologies. When this project begin, I hadn’t any knowledge in the technologies used, for this reason, during this project, we learn the features, advantages and disadvantages for each one.

The functionality of this application is to show real time status of parkings from a city, and also User administration for access to view the map.

The first versión of this development is made in NodeJS and AngularJS. Both tecnologies come from JavaScript, NodeJS is use in BackEnd for managing the server and AngularJS in FrontEnd. Also, this app use a login social, Storing information in a non-relational data base, MongoDB.

The second versión is developed with Django, a framework of Phytion for manage the server in BackEnd and Polymer to show the information in FrontEnd. Finally, we use the identity manager in Django coordinated with social login.

In the last part of this project, a comparison between the various technologies is done by comparing the strengths and weaknesses of all the technologies used.

1 INTRODUCCIÓN

Los datos abiertos, más conocidos como Open Data^[2], proviene de una filosofía y una práctica que prosigue que cierta información esté disponible para su uso, sin restricciones de derechos de autor, patentes o de otros mecanismos de control, su mentalidad es muy similar a la mentalidad que tienen las demás comunidades abiertas, como puede ser el software libre.

Estos datos históricamente han estado limitados sus accesos a través de diversas limitaciones, como licencias, patentes o copyright, y se pretende, que a través de este movimiento de datos abiertos, puedan ser accedidos por los ciudadanos para que puedan entender los datos mejor, aumentando así el nivel de conocimiento de la población, y otra motivación que ha influido en el avance de la práctica de los “Open Data” es el hecho de que a través de ellos, las instituciones públicas aumentan la transparencia de sus administraciones ofreciendo los datos al sector público, para que puedan ser accedidos libremente.

1.1 Objetivo

Los principales objetivos del desarrollo de este trabajo es el obtener datos de fuentes públicas y mostrarlos visualmente a los usuarios de tal modo que la información sea comprensible y accesible a los ciudadanos.

Como para ello existen múltiples modos de mostrar dicha información, en modo texto, gráfica, se va a realizar también un estudio de las principales tecnologías web que ofrecen la posibilidad de mostrar estos datos, realizando comparativas entre ellas, tanto cualitativamente (cual es más fácil de utilizar, ...) como cuantitativamente (cual es más eficiente).

Para ello, se va a realizar la misma funcionalidad en con distintos lenguajes de programación, para poder realizar la comparativa. La funcional escogida es localizar en un mapa los parkings de Pamplona, mostrando gráficamente el estado de ocupación del parking que se actualizará en tiempo real.

De los lenguajes elegidos para realizar el desarrollo no se tiene conocimiento ni teórico ni práctico, por lo que otro objetivo de este desarrollo es profundizar en ellos.

Por lo tanto, se podrían resumir brevemente los principales objetivos:

- Ofrecer una funcionalidad con datos obtenidos a través de “Open Data”
- Aprender nuevos lenguajes y paradigmas de programación.
- Comparar los diversos lenguajes aprendidos.

1.2 Motivación

La elección de las diversas tecnologías que se deben utilizar en el desarrollo de un proyecto informático es un punto de inflexión en la realización de dicho proyecto, por lo tanto una de las principales características de un buen arquitecto software es conocer las principales características de las diversas alternativas que se ofrecen en el mercado, para ello debe tener una mente abierta para seleccionar una o varias de ellas, sin centrarse en las que está más familiarizado y se encuentran en su zona de confort.

Debido a esta característica deseable de un arquitecto, la principal motivación de este desarrollo, desde el punto de vista tecnológico, es el conocer y profundizar los conocimientos en las tecnologías que se utilizan, poniendo foco en las principales características que nos ofrecen, teniendo como finalidad tener los conocimientos para poder decidir, cuál sería la mejor elección en desarrollos futuros.

Desde el punto de vista práctico, se ofrece una herramienta al usuario final, que hace que pueda interpretar fácilmente los diversos datos que ofrecen las fuentes gubernamentales, de los que no tienen acceso ni sabe interpretarlos en la mayoría de los casos.

2 ESTADO DE LA CUESTIÓN

2.1 Open Data

A lo largo de los últimos años se han puesto en marcha por parte de algunos gobiernos e instituciones públicas programas destinados a fomentar y coordinar la publicación de datos públicos abiertos, que reciben el nombre Open Data, destacando en España la “iniciativa aporta”, para que a partir de estos datos se puedan desarrollar aplicaciones. ^[2]

Otra definición que podemos obtener de Open Data es la siguiente:

Los datos abiertos son datos que pueden ser utilizados, reutilizados y redistribuidos libremente por cualquier persona, y que se encuentran sujetos, cuando más, al requerimiento de atribución y de compartirse de la misma manera en que aparecen. ^[3]

Por lo tanto, las principales características que debe ofrecer las siguientes características:

- **Disponibilidad y acceso:** la información debe estar disponible como un todo y a un costo razonable de reproducción, preferiblemente descargándola de internet. Además, la información debe estar disponible en una forma conveniente y modificable.
- **Reutilización y redistribución:** los datos deben ser provistos bajo términos que permitan reutilizarlos y redistribuirlos, e incluso integrarlos con otros conjuntos de datos.
- **Participación universal:** todos deben poder utilizar, reutilizar y redistribuir la información. No debe haber discriminación alguna en términos de esfuerzo, personas o grupos. Restricciones “no comerciales” que prevendrían el uso comercial de los datos; o restricciones de uso para ciertos propósitos (por ejemplo, sólo para educación) no son permitidos.

2.1.1 Iniciativa Aporta



Ilustración 1: Logo Iniciativa Aporta

Aporta es una iniciativa promovida por el Ministerio de Industria, Energía y Turismo, a través de la Entidad Pública Empresarial Red.es, y en colaboración con el Ministerio de Hacienda y Administraciones Públicas, que se lanza en 2009 con el fin de promocionar la cultura de la apertura de información en España.

Aporta nace con el objetivo de crear las condiciones para el desarrollo del mercado de la reutilización de la información del sector público, así como, para dar apoyo a las unidades administrativas, en las actividades técnicas y organizativas necesarias para que publiquen de acuerdo con la legislación vigente y de la forma más amigable para su reutilización, la información de acceso no restringido que recogen.^[1]

Los datos ofrecidos por la “Iniciativa Aporta” se encuentran en varios formatos, siendo los principales: XML y JSON.

2.2 XML

XML (eXtensible Markup Language) es un lenguaje de marcado diseñado para el transporte y almacenamiento de información desarrollado por el consorcio W3C.

2.2.1 Aspectos generales de XML

Los orígenes de XML se remontan a cuando IBM creó el lenguaje llamado GML (*Generalized Markup Language*), con el objetivo de almacenar grandes cantidades de

información. Cuando se intentó normalizar ese lenguaje a través de la ISO se creó el SGML (*Standard Generalized Markup Language*), diseñado sobre GML.

Con el nacimiento de la web, apareció HTML (*HyperText Markup Language*), y surgió XML para permitir mezclar elementos de los diversos lenguajes que se utilizaban, facilitando la interpretación y tratamiento de la información almacenada. ^[2]

Las principales ventajas de XML son:

- Extensible: Permite añadir nuevas etiquetas, sin necesidad de modificar la esencia del lenguaje.
- Los analizadores sintácticos son sencillos y estándares.
- Sencillez: Los documentos XML son sencillos de entender y procesar, ya que su estructura está claramente definida.
- Permite convertir en información los datos almacenados, al enriquecerlos con etiquetas que añaden un significado concreto dentro de un contexto determinado.

También XML tiene algunas limitaciones:

- No es fuertemente “tipado” como la mayoría de lenguajes de programación, complicando el “mapeo” entre los tipos de datos de XML y programas escritos en otros lenguajes.
- Es un lenguaje que tiene un coste computacional elevado para interpretar los datos.

2.2.2 Ejemplo documento XML

Un ejemplo de un documento XML sería el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Ejemplo de XML -->

<ficha>
    <nombre>Juan Francisco</nombre>
```



```
<apellido>Hernández</apellido>
<dni>123456789A</dni>

<otros cantidad="0" />
</ficha>
```

Un documento está formado por las siguientes partes:

- Prólogo: indica la versión de XML utilizada, codificación, etc. No es obligatorio su uso, pero sí es recomendable. El prólogo en el ejemplo es:

```
<?xml version="1.0" encoding="UTF-8"?>
```

- Cuerpo: El cuerpo del XML está formado por los demás elementos que no forman parte del prólogo, como pueden ser comentarios, atributos, etc. El cuerpo es una sección obligatoria del fichero XML.
 - Elementos: El cuerpo de un XML está formado por uno o varios elementos. Los elementos están delimitados por marcas, llamadas etiquetas de la forma `<nombre>` y `</nombre>`, donde nombre es el elemento a delimitar. En el caso de no tener ningún elemento, se utiliza el autocierre, `<nombre />`. Los elementos pueden tener elementos anidados.

```
<ficha>
  <nombre>Juan Francisco</nombre>
  <apellido>Hernández</apellido>
  <dni>123456789A</dni>

  <otros cantidad="0" />
</ficha>
```

- Atributos: Permiten añadir al elemento información adicional. En el ejemplo se ve que la cantidad de otros es 0:

```
<otros cantidad="0" />
```

- Comentarios: Los comentarios se pueden poner en cualquier parte del cuerpo y siguen el formato `<!-- comentario -->`:

`<!-- Ejemplo de XML -->`

2.3 JSON

JSON (JavaScript Object Notation) ^[4] es otro de los formatos típicos a la hora de mostrar y transmitir información a lo largo de internet.

Permite varios tipos de datos:

- Números: Se permiten números negativos y opcionalmente pueden contener parte fraccional separada por puntos. Ejemplo: 123.456
- Cadenas: Representan secuencias de cero o más caracteres. Se ponen entre doble comilla y se permiten cadenas de escape. Ejemplo: "Hola"
- Booleanos: Representan valores booleanos y pueden tener dos valores: true y false
- null: Representan el valor nulo.
- Vector: Representa una lista ordenada de cero o más valores los cuales pueden ser de cualquier tipo. Los valores se separan por comas y el vector se mete entre corchetes. Ejemplo ["hola", "adios", "que tal"]
- Objetos: Son colecciones no ordenadas de pares de la forma <nombre>:<valor> separados por comas y puestas entre llaves. El nombre tiene que ser una cadena y entre ellas. El valor puede ser de cualquier tipo de los anteriores.

JSON es más difícil de ver la información para las personas visualmente al contrario de XML, sin embargo, el tratamiento por los lenguajes de programación, es mucho más eficiente con JSON que con un XML, sobretodo en JavaScript y sus frameworks, ya que utiliza objetos con notación propia, mientras que XML tiene que “decodificar” la información.

Tabla 1: Comparativa XML JSON

XML	JSON
<pre><ficha> <nombre>Juan Francisco</nombre> <apellido>Hernández</apellido> <dni>123456789A</dni> <otros cantidad="0" /> </ficha></pre>	<pre>{ "ficha": { "nombre": "Juan Francisco", "apellido": "Hernández", "dni": "123456789A", "otros": { "cantidad": "0" } } }</pre>

Durante el desarrollo de este trabajo, veremos cómo se trabaja tanto con XML y JSON en diversos lenguajes, se utilizará XML, porque es el formato con el que se nos ofrece la información, y en el servidor, la transformaremos a JSON.

XML y JSON, se utilizarán en los dos desarrollos, ya que es el modo de obtener la información (XML) y de comunicar el cliente con el servidor (JSON).

En el primer desarrollo los lenguajes escogidos son los siguientes:

- Node.js: Se ha escogido Node.js como lenguaje del parte del servidor, debido a que es un lenguaje demandado actualmente en el lado del servidor y así poder comprobar como es desarrollar un servidor en “JavaScript”
- AngularJS: Se ha escogido AngularJS como lenguaje de lado del cliente, ya que al igual que Node.js es un lenguaje muy demandado, y muy potente, que ofrece mucha versatilidad a la hora de desarrollar con él.
- MongoDB: MongoDB forma parte de un nuevo modelo de bases de datos frente a las tradicionales SQL, ofreciendo una gran velocidad en el procesamiento de los datos.

Y en el segundo, se utilizarán las siguientes tecnologías:

- Django: Django es un framework basado en Python del lado del servidor, muy versátil, ya que tiene todas las características de Python, que es un lenguaje de programación muy extendido.
- Polymer: Polymer se presenta como una alternativa a la hora de desarrollar la parte front de un web a AngularJS. Está basado en plantillas, y se obtienen resultados muy buenos con facilidad.

2.4 Node.js



Web: <https://nodejs.org/es/>

Última versión estable: 5.6.0

Licencia: Licencia MIT

Desarrollador Ryan Lienhart Dahl

Node.js es un entorno de ejecución desarrollado principalmente para su funcionamiento en la capa del servidor. Está basado en ECMAScript, siendo una programación asíncrona, con una arquitectura orientada a eventos, basado en el motor V8 de Google y compuesto por módulos a la hora de su desarrollo.

Una de las principales características de Node.js que llaman la atención, es su arquitectura, que está desarrollada tanto en C++ como en JavaScript, y la gestión y uso de módulos externos.^[16]

2.4.1 Arquitectura Node.js

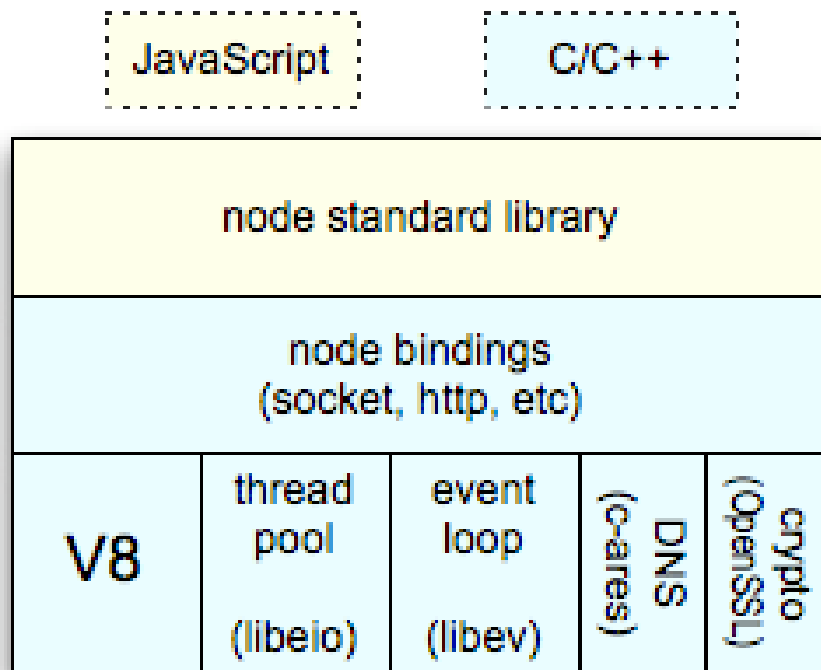


Ilustración 3: Arquitectura Node.JS

Los elementos más destacables que componen la arquitectura de Node.JS son los siguientes ^[5]

- **Motor V8:** V8 es el ambiente de ejecución para JavaScript creado para Google Chrome. Se convirtió en software libre en el año 2008, está desarrollado en C++ y compila el código fuente JavaScript en código máquina en lugar de interpretarlo en tiempo real.
- **Event Loop:** Es una capa de abstracción de funcionalidades de redes y sistemas de archivos, lo que hace que se encargue de la gestión de los eventos. Está basado en la biblioteca de libev desarrollado en C++.
- **Thread pool:** Node.JS utiliza un único thread, en lugar de otros servidores que utiliza un thread por cada petición que recibe, lo que aumenta el consumo de

recursos del sistema. Está basado en la biblioteca libeio desarrollada en C++. Libeio se encarga de mejorar el rendimiento I/O, haciendo que se deba de utilizar un callback siempre que se necesite el uso de alguna función I/O.

- **Node Library:** Son que implementan funcionalidades extras al core de node.js. Están desarrolladas en JavaScript, en contraste con el core de node.js, desarrollado en C++. También existen cantidad de módulos desarrollados por la comunidad de desarrolladores de Node.js, que implementan funcionalidades extras y se instalan a través de npm.

2.4.2 Módulos en Node.JS

2.4.2.1 NPM



Web: <https://www.npmjs.com/>

Última versión estable: 3.8.3

Licencia: MIT License

Ilustración 4: Logo NPM

NPM es un software que se instala junto con Node.JS. Es el encargado de gestionar los paquetes que se utilizan en Node.JS, similar a Apt, en las principales distribuciones de Unix. ^[17]

Los principales comandos de npm son: ^[18]

- **npm init:** Encargado de crear, modificar y generar el fichero package.json (fichero de configuración de la app).
- **npm install <module>:** Busca en el repositorio de módulos de npm, e instala el modulo solicitado.
- **npm update:** Se encarga de actualizar el fichero package.json y los módulos que se utilizan.

NPM ofrece funcionalidades muy interesantes, ya que al almacenarse la información en el package.json se pueden compartir los proyectos en la red de modo muy sencillo y liviano, ya que no hace falta incluir en las bibliotecas utilizadas.

Simplemente con compartir con github o plataformas similares el código fuente desarrollado, con la información correctamente actualizada en el package.json, con el `npm install` se instalan todos los módulos necesarios para el funcionamiento de la aplicación.

2.4.2.2 Express



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

Última versión estable: 4.0.0

Licencia: MIT License

Ilustración 5: Logo express

Express es una infraestructura de aplicaciones web con Node.js, con gran variedad de métodos para crear una aplicación ‘express’, generando el esqueleto de la aplicación.

[18]

2.4.2.3 Passport



Web: <http://passportjs.org/>

Última versión estable: 0.3.2

Licencia: MIT License

Ilustración 6: Logo Passport

Passport es un middleware de autenticación para Node.js, flexible y modular, que se complementa con la arquitectura express. Permite realizar la autenticación a través de Facebook, Twitter,..^[19]

2.4.2.4 Jade



Ilustración 7: Logo Jade

Web: <http://jade-lang.com/>

Última versión estable: 1.11.0

Licencia: MIT License

Jade es un generador de código HTML, que genera código HTML sin la necesidad de utilizar la sintaxis propia de HTML.^[21]

Un ejemplo de Jade es el siguiente, según nos muestra su web:

```
doctype html
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript').
      if (foo) {
        bar(1 + 5)
      }
  body
    h1 Jade - node template engine
    #container.col
      if youAreUsingJade
        p You are amazing
      else
        p Get on it!
      p.
        Jade is a terse and simple
        templating language with a
        strong focus on performance
        and powerful features.
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Jade</title>
    <script type="text/javascript">
      if (foo) {
        bar(1 + 5)
      }
    </script>
  </head>
  <body>
    <h1>Jade - node template engine</h1>
    <div id="container" class="col">
      <p>You are amazing</p>
      <p>
        Jade is a terse and simple
        templating language with a
        strong focus on performance
        and powerful features.
      </p>
    </div>
  </body>
</html>
```

Ilustración 8: Ejemplo Jade

2.4.3 Ejemplo Hola Mundo en Node.js y express

Un ejemplo de Hola Mundo, que se mostrará en el navegador al hacer una petición al puerto 3000 de nuestra máquina local, fichero hello.js:

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



```
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World!');
});

app.listen(3000, function () {
  console.log('Example app listening on port 3000!');
});
```

Ilustración 9: Ejemplo Hola Mundo NodeJS

En este ejemplo, se ha utilizado el modulo ‘express’, que se indica que se requiere para su ejecución e instanciación con las siguientes líneas:

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

var app = express();
```

Ilustración 10: Hola Mundo NodeJS 1

Una vez que se ha definido la estructura de la aplicación con express, se indica que cuando reciba una petición en el end-point (/), ejecute el siguiente código, que es mostrar ‘Hello World!’ en la página del navegador.

```
app.get('/', function (req, res) {
  res.send('Hello World!');
});
```

Ilustración 11: Hola Mundo NodeJS 2

Finalmente, se indica que escuche las peticiones en el puerto 3000, y muestre por consola un log indicándolo:

```
app.listen(3000, function () {  
  console.log('Example app listening on port 3000!');  
});
```

Ilustración 12 Hola Mundo NodeJS 3


Una vez escrito el código, procederemos a ejecutarlo. Para ello hay que tener instalado Node.JS en la máquina. Se puede obtener desde la web oficial para múltiples plataformas. Para realizar el ejemplo se ha instalado en Windows10.

Una vez instalado, desde la consola de Windows (CMD), y desde la ruta donde este el fichero se ejecutarán los siguientes comandos:

```
Npm install express  
Node hello.js
```

Ilustración 13: Hola Mundo NodeJS 4

La primera línea indicia al gestor de paquetes de node, que instale express y la segunda que se ejecute el fichero hello.js



```

Node.js command prompt - node hello.js
Your environment has been set up for using Node.js 4.4.2 (x64) and npm.

D:\Users\jfh1>npm install express
npm WARN package.json jfh1@1.0.0 No description
npm WARN package.json jfh1@1.0.0 No repository field.
npm WARN package.json jfh1@1.0.0 No README data
express@4.13.4 node_modules\express
├── escape-html@1.0.3
├── vary@1.0.1
├── cookie-signature@1.0.6
├── methods@1.1.2
├── parseurl@1.3.1
├── fresh@0.3.0
├── etag@1.7.0
├── cookie@0.1.5
├── content-type@1.0.1
├── array-flatten@1.1.1
├── content-disposition@0.5.1
├── serve-static@1.10.2
├── merge-descriptors@1.0.1
├── path-to-regexp@0.1.7
├── range-parser@1.0.3
├── utils-merge@1.0.0
├── depd@1.1.0
├── qs@4.0.0
├── on-finished@2.3.0 (ee-first@1.1.1)
├── finalhandler@0.4.1 (unpipe@1.0.0)
├── debug@2.2.0 (ms@0.7.1)
├── proxy-addr@1.0.10 (forwarded@0.1.0, ipaddr.js@1.0.5)
├── type-is@1.6.12 (media-typer@0.3.0, mime-types@2.1.10)
├── accepts@1.2.13 (negotiator@0.5.3, mime-types@2.1.10)
└── send@0.13.1 (statuses@1.2.1, destroy@1.0.4, ms@0.7.1, mime@1.3.4, http-errors@1.3.1)

D:\Users\jfh1>node hello.js
Example app listening on port 3000!
  
```

Ilustración 14: Ejemplo Hola Mundo NodeJS 5

Una vez que vemos el mensaje del log por la consola, quiere decir que se está ejecutando nuestra aplicación, por lo tanto, desde el navegador, si hacemos una petición a localhost en el puerto 3000 nos sale la siguiente página:

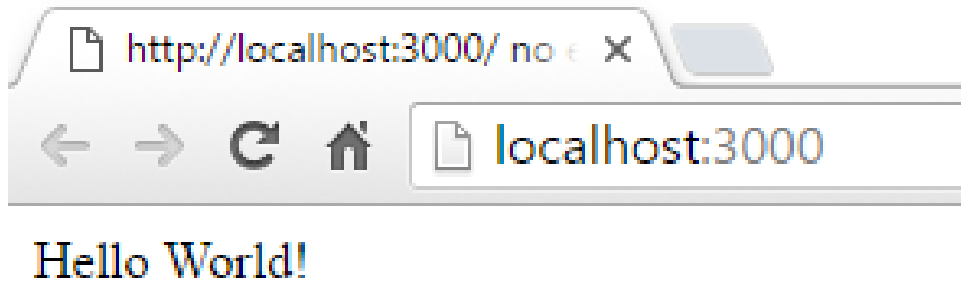


Ilustración 15: Ejemplo Hola Mundo Node.JS 6

2.5 AngularJS



Ilustración 16: Logo AngularJS

Web: <https://angularjs.org/>

Última versión estable: 1.5.5

Licencia: MIT License

AngularJS es un framework MVC de JavaScript desarrollado en código abierto y mantenido por Google utilizándose para desarrollar y mantener aplicaciones web de una sola página. El framework lee el HTML con las etiquetas personalizadas de AngularJS, obedeciendo a las directivas asociadas en AngularJS a esas etiquetas, asignándole atributos dinámicamente mediante JavaScript. ^[22]

Una de las características que llama mucho la atención de AngularJS es el doble-binding, que actualiza la información de las variables en tiempo real en todos los lugares donde esté instanciada y eso es debido a su peculiar arquitectura.

2.5.1 Arquitectura

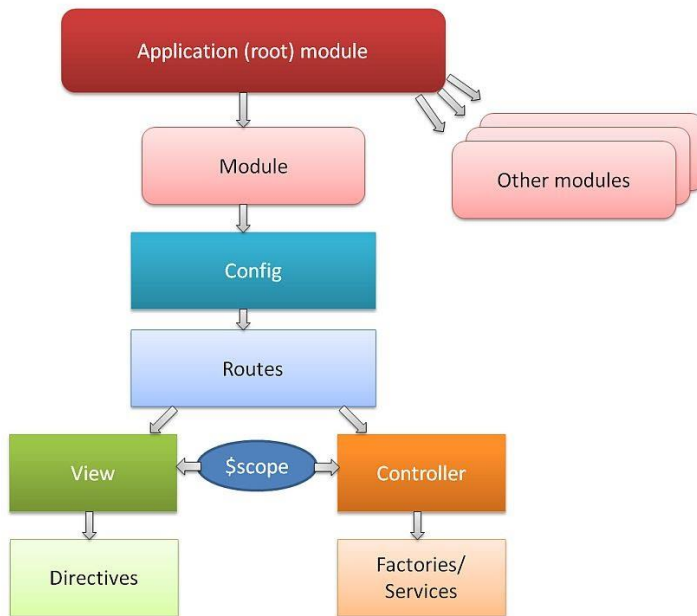


Ilustración 17: Arquitectura AngularJS (1)

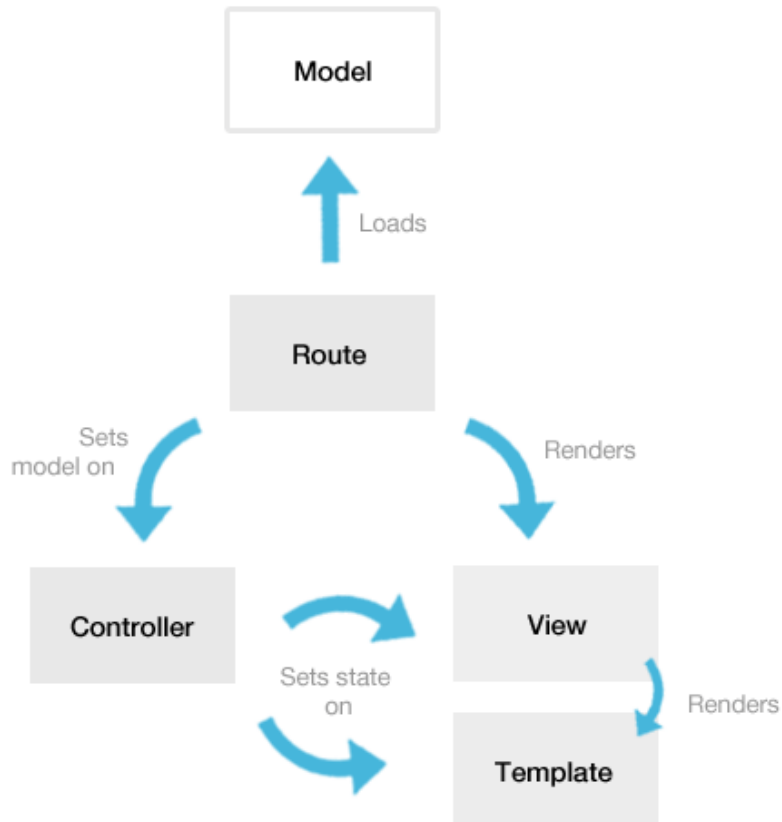


Ilustración 18: Arquitectura AngularJS (2)

Los componentes de AngularJS son los siguientes: ^[6]

- Vistas: Están escritas en HTML, utilizando directivas de AngularJS que preprocesan y generan el HTML final. Cada controlador tiene uno o más bindings con el controller que define un alcance o \$scope con varias variables que hacen de modelo.
- Controller: adapta la vista con el modelo final, habla con los services/factories para manejar la comunicación con la fuente de los datos.
- Services: Reutilizan funcionalidades comunes entre controllers, como puede ser actualizar la caché, o los orígenes de los datos.
- Module: funciona como contenedor de los elementos visuales, la lógica de la vista y el comportamiento del lado cliente.
- Route: relacione un servicio REST con una funcionalidad en el cliente.

2.5.2 Características

AngularJS es, como hemos dicho, un framework MVC, eso quiere decir que implementa el patrón Modelo-Vista-Controlador:

- Modelo: Es el conjunto de datos que se utilizan en la aplicación.
- Vista: Es la interfaz de usuario, y es lo que utilizan el cliente.
- Controlador: Es la parte del patrón que contiene toda la lógica, y se encarga de relacionar la vista con el modelo.

Las principales características de AngularJS son las siguientes:

- Orientado a datos: Se comparten los mismos modelos de datos, tanto en el cliente con el servidor.
- Declarativo: Se puede interpretar el código web que se ejecuta.
- Separación de conceptos: Al implementar el patrón MVC, quedan claro los conceptos que componen AngularJS.

- Inyección de dependencias: La inyección de las dependencias está presente en los apartados de AngularJS, lo que hace se puedan implementar de manera independiente cada módulo que se implementen.
- Extensible: Se pueden integrar las directivas de AngularJS con otros frameworks como bootstrap o JQuery.ui.

Por lo tanto, los principales beneficios que aporta el uso de AngularJS son los siguientes:

- Permite crear webapps de una sola página, gracias a los servicios, se realizan peticiones AJAX.
- AngularJS requiere pocas líneas de código para realizar las tareas si se compara con otros frameworks, como JQuery.
- Su naturaleza declarativa simplifica la tarea de mantenimiento, ya que se entiende perfectamente el código.

2.5.3 Hola Mundo en AngularJS

Hola

Escribe y veras el saludo

Ilustración 19: Hola Mundo AngularJS(1)

Hola MUNDO

Escribe y veras el saludo

Ilustración 20: Hola Mundo AngularJS(2)

El código necesario para el ejemplo es el siguiente:

```
<!DOCTYPE html>

<html lang="es" ng-app>

<head>

  <meta charset="UTF-8">

  <title>Hola Mundo</title>

</head>

<body>

  <h1>Hola {{modelo}}</h1>

  Escribe y veras el saludo <input type="text" ng-model="modelo">

  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.23/angular.min.js"
></script>

</body>

</html>
```

Ilustración 21: Hola Mundo Angular JS – Código

Ahora vamos a analizar el código:


```
<html lang="en" ng-app>
```

Ilustración 22: Hola Mundo Angular JS – Código(1)

En la etiqueta html, se añade la directiva ng-app que indica que es una aplicación de AngularJS. Se le puede dar nombre o no.

```
<h1>Hola {{modelo}}</h1>
```

Ilustración 23: Hola Mundo AngularJS - Código (2)

Cualquier variable entre llaves {{ y }} indica que es una variable de AngularJS. Tiene en doble-binding de AngularJS y si se modifica el ng-model que está asociado, se modifica su valor.

```
Escribe y veras el saludo <input type="text" ng-model="modelo">
```

Ilustración 24: Hola Mundo AngularJS - Código (3)

El tag ng-model indica que es un modelo de datos de AngularJS y que cualquier modificación que sufra ese campo, se actualizará en toda la aplicación (ng-app).

```
<script  
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.23/angular.min.js"  
></script>
```

Ilustración 25: Hola Mundo AngularJS - Código (4)

Con este fragmento de código cargamos el código fuente de AngularJS desde su servidor, para poder utilizarlo dentro de la aplicación AngularJS.

2.5.4 Angular Material



Web: <https://material.angularjs.org>

Última versión estable: 1.0.8

Licencia: MIT License

Ilustración 26: Logo Angular Material

Framework para AngularJS que permite el desarrollo de interfaces utilizando como referencia de implementación las especificaciones de Google Material Design.^[23]

Permite que la interfaz sea similar tanto en la vista web, como en una vista de teléfono móvil. Se utilizará durante el desarrollo para realizar una interfaz que sea similar a la estándar y que el usuario conozca su funcionamiento sin problemas.

2.6 MongoDB



Web: <http://www.mongodb.org/>

Última versión estable: 3.2.4

Licencia: GNU AGPL v3.0 (drivers: licencia Apache)

2.6.1 NoSQL y Documental

Las bases de datos de tipo NoSQL, indica que no se utiliza el modelo clásico de almacenamiento de la información en tablas y sentencias SQL de consultas, como pueden ser MySQL, SQLite.

Las bases de datos NoSQL son más flexibles, en lugar de utilizar tablas y relaciones entre los datos de las tablas, utilizan documentos con estructuras arbitrarias. Su principal característica, son, la flexibilidad que brinda el esquema de los datos, su

simplicidad de uso, obteniendo un alto rendimiento, escalabilidad y facilidad de mantenimiento.

Cabe destacar, y para evitar malentendidos, diferencias una base de datos NoSQL y una base de datos documental. Las bases de datos documentales son una categoría concreta de bases de datos NoSQL, ya que existen bases de datos como MotionDb que son almacenes de pares clave-valor. [7]

La escalabilidad que ofrece estas bases de datos son:

- Escalado vertical: Se pueden añadir más recursos al sistema para que atienda más peticiones.
- Escalado horizontal: Se añaden más nodos al sistema, haciendo que tenga más nodos en paralelo. Es más sencillo que el vertical.

2.6.2 Comparación MongoDB con MySQL

Como se ha explicado en el apartado anterior, MongoDB es realmente distinto a las bases de datos tradicionales como puede ver en la siguiente comparativa con MySQL [8]
[9]

Tabla 2: Tabla comparativa MongoDB y MySQL

	MongoDB	MySQL
Representación de datos	Documentos JSON	Tablas con filas y columnas
Consulta	Documento explicando que se consulta	Sentencias SQL
Consumo de recursos		
RAM	Consumo razonable	Asigna toda la RAM disponible

CPU	Marginal, en relación con la tarea	Cantidad razonable, en relación con la tarea
Índices	B-Tree y Geospacial 2D	B-Tree y Hash
Escala	Replicación y Sharding	Replicación y Clustering
Consultas		
Inserts	db.users.insert({ user_id: 'bcd001', age: 45, status: 'A' })	INSERT INTO users (user_id, age, status) VALUES ('bcd001', 45, 'A')
Seleccionar todos los elementos	db.users.find()	SELECT * FROM users
Seleccionar algún valor algún elemento	db.users.find({ status: "A" }, { user_id: 1, status: 1, _id: 0 })	SELECT user_id, status FROM users WHERE status = "A"
Actualización de datos	db.users.update({ age: { \$gt: 25 } }, { \$set: { status: "C" } }, { multi: true })	UPDATE users SET status = "C" WHERE age > 25

Eliminar registros	<code>db.users.remove({ status: "D" })</code>	<code>DELETE FROM users WHERE status = "D"</code>
Crear tablas o colección	<p>Implicito al realizar un Insert. Si no existe el registro, lo crea. Se puede crear de todos modos:</p> <p><code>db.createCollection("users")</code></p>	<pre>CREATE TABLE users (id MEDIUMINT NOT NULL AUTO_INCREMENT, user_id Varchar(30), age Number, status char(1), PRIMARY KEY (id))</pre>
Editar tabla o colección para añadir columnas	<p>No es necesario. Las colecciones no tienen estructura obligatoria. De todos modos, se puede realizar:</p> <p><code>db.users.update(</code></p> <pre> { }, { \$set: { join_date: new Date() } }, { multi: true })</pre>	<pre>ALTER TABLE users ADD join_date DATETIME</pre>

Editar tabla o colección para eliminar columnas	<p>No es necesario. Las colecciones no tienen estructura obligatoria. De todos modos, se puede realizar:</p> <pre>db.users.update({ }, { \$unset: { join_date: "" } }, { multi: true })</pre>	<p>ALTER TABLE users DROP COLUMN join_date</p>
Eliminar tabla o colección	<pre>db.users.drop()</pre>	DROP TABLE users

2.7 MEAN Stack

Mean stack hace referencia a los desarrollos realizados con MongoDB, Express, Angular y NodeJs. ^[23]

Por lo tanto se podría decir, que el desarrollo de la primera aplicación es un desarrollo MEAN Stack, ya que se ha desarrollado completamente con las tecnologías que le dan nombre. ^[24]



Ilustración 28: Mean logo

Como se puede entender, las funcionalidades los elementos de MEAN son las siguientes:

- MongoDB: Encargado de almacenar la información en la base de datos.
- Express: Módulo de NodeJs encargado de gestionar y armar la estructura del servidor.
- AngularJS: Encargado del desarrollo del front de la aplicación desarrollada con MEAN.
- NodeJs Lenguaje utilizado para la gestión del servidor.

Todos los elementos que forman parte de un desarrollo MEAN, están desarrollados a partir de Javascript, por lo que se puede decir, que se puede crear una aplicación completa y funcional utilizando solamente Javascript.

El esquema del funcionamiento de una petición en una aplicación MEAN es el que muestra el siguiente gráfico ^[25]:

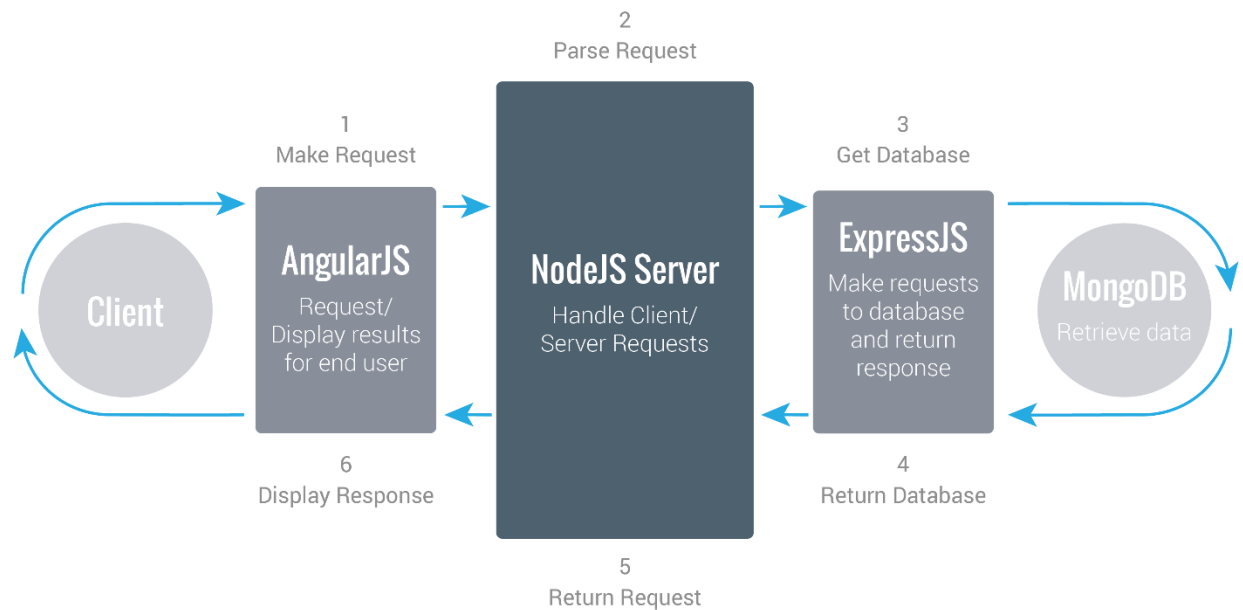


Ilustración 29: MEAN flujo

- El cliente, hace una petición a través de AngularJS
- NodeJS recibe la petición y la trata
- ExpressJs hace la petición a la base de datos.
- MongoDB, recibe la petición y hace la consulta en la base de datos, y devuelve a express el resultado.
- NodeJS responde a la petición de AngularJS con la información solicitada.
- AngularJS muestra la información devuelta por el servidor.

2.8 Django

django

Ilustración 30: Logo Django

Web: www.djangoproject.com

Última versión estable: 1.9.4

Licencia: BSD

Django es el framework de desarrollo web del lado del cliente que está desarrollado en Python, y está desarrollado al patrón MVC (Modelo-Vista-Controlador). Está desarrollado para facilitar la creación de sitios web complejos, poniendo el foco en el reuso, conectividad y extensibilidad de los componentes y el DRY (Dont Repeat Yourself, no te repitas). ^[26]

2.8.1 Arquitectura Django

La arquitectura de Django, hace referencia al patrón MVC, pero en realidad hace el controlador es llamado vista, y la vista template, por lo tanto sería el patrón MVT(Modelo-Vista-Template).

Esto es debido a que la vista en Django hace referencia a los datos que se van a mostrar (modelo en el MVC), y la vista, que es como se representa los datos en verdad, se realiza a través de las diversas templates que tiene Django. ^[10]

Esto lo hace posible a través de capas intermedias, como son mediator y foundation, que permite la comunicación entre Entity y la presentación.

Las capas de Django son las siguientes:

- Presentation: Es la encargada de la interacción con el usuario.
- Control: Es la encargada de la lógica de la aplicación.
- Mediator: Capa encargada entre la interacción de Entity y Foundation
- Entity: Maneja las entidades y el mapeo objeto-relacio de Django.
- Foundation: Se encarga de trabajar a bajo nivel con la base de datos.

Las capas de presentación y control hacen referencia a Template y View. Entity es el Model y Mediator y Foundation es la capa que se encarga de la comunicación entre Model y Database.

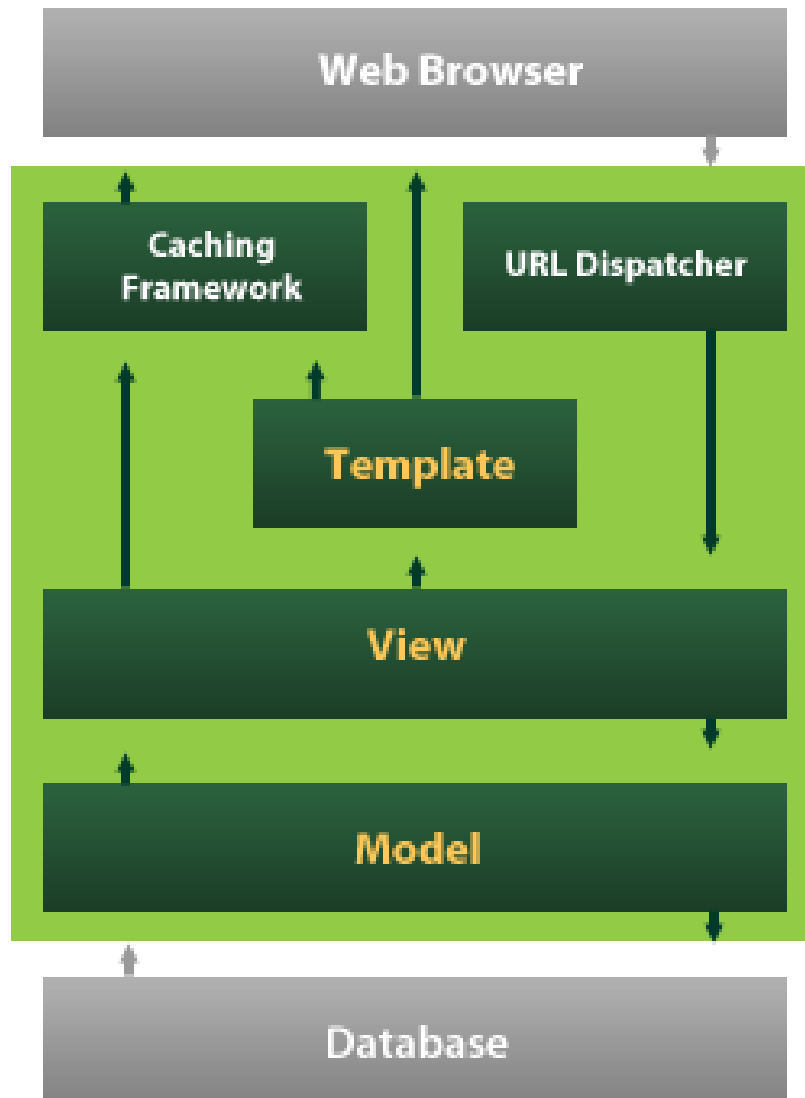


Ilustración 31: Arquitectura Django

2.8.2 Principales características de Django

Las principales características de Django, son las siguientes: ^[25]

- Sencillez a la hora de crear un proyecto: Solamente genera 5 ficheros para tener una configuración inicial y básica del proyecto.
- Administrador por defecto: Es el único framework que incluye el apartado de administración por defecto.

- Rapidez: Otra característica principal en Django es la rapidez que ofrece a la hora de cargar las webs.
- ORM: Permite realizar sentencias para bases de datos SQL, sin necesidad de usar SQL.
- Gestor de módulos: Al igual que Node.JS tiene gestor de módulos, pero es menos potente que npm.

2.8.3 Hola Mundo en Django

Para realizar un hola mundo, primero tenemos que generar el proyecto de Django, que se hace con el siguiente comando:

```
django-admin.py startproject hola
```

Ilustración 32: Django Hola Mundo (1)

Este comando nos genera la siguiente estructura de carpetas:

```
hola
  manage.py
  hola
    __init__.py
    settings.py
    urls.py
    wsgi.py
```

Ilustración 33: Django Hola Mundo (2)

Ahora que tenemos el proyecto, creamos la “app” de Django, para el hola mundo desde la carpeta raíz:

```
python manage.py startapp hola_mundo
```

Ilustración 34: Django Hola Mundo (3)

Este commando, nos genera los siguientes ficheros y carpetas:

```
hola
  manage.py
  hola
    __init__.py
    settings.py
    urls.py
    wsgi.py
  hola_mundo
    __init__.py
    models.py
    tests.py
    views.py
```

Ilustración 35: Django Hola Mundo (4)

Ahora, hay que modificar el fichero settings.py, para indicarle que hay una nueva aplicación Django, que es añadir el nombre de la app creada (hola_mundo), en el apartado `INSTALLED_APPS`.

También registraremos las templates correspondientes que ofrece Django.

Siendo el apartado del fichero manage.py finalmente así:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'hola_mundo'
```

```
]

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [BASE_DIR+"/templates", ],//directorio templates

        'APP_DIRS': True,

        'OPTIONS': {

            'context_processors': [

                'django.template.context_processors.debug',

                'django.template.context_processors.request',

                'django.contrib.auth.context_processors.auth',

                'django.contrib.messages.context_processors.messages',

            ],

        },

    },

]
```

Ilustración 36: Django Hola Mundo (5)

El siguiente que modificar el fichero views.py, definiendo la lógica de la aplicación, en este caso es sencilla. Solamente tiene que renderizar el mensaje “Hola Mundo”:

```
from django.http import HttpResponse

def hola(request):

    context = {

        'saludo': 'Hola mundo con django',

    }

    return render(request, 'saludo.html', context)
```

Ilustración 37: Django Hola Mundo (6)

Creamos la template de Django, para que se pueda rellenar con los datos del método. Para ello se crea el fichero saludo.html en la carpeta template.

Definimos la url que escuchará nuestra app, en este ejemplo, es la raíz, por lo tanto, hay que modificar el fichero urls.py con lo siguiente :url('r', 'hola_mundo.views.hola')

Esto quiere decir, que cuando llegue una petición por esa url, se debe ejecutar ese método, que hemos definido en el paso anterior:

```
from django.conf.urls import url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^hola/', 'hola_mundo.views.hola'),
]
```

Ilustración 38: Django Hola Mundo (7)

Y ahora ejecutamos el servidor Django con el comando:

```
python manage.py runserver
```

Ilustración 39: Django Hola Mundo (8)

Y el resultado es:

Hola mundo con django

Ilustración 40: Django Hola Mundo (9)

2.9 Polymer



Ilustración 41: Logo Polymer

Web: www.polymer-project.org

Última versión estable: 1.0

Licencia: BSD

Polymer es un proyecto open-source liderado por un equipo de desarrolladores front-end que trabajan en Google. ^[27]

Polymer se basa en el uso de polyfill y web componentes:

- Polyfill^[13]: plugin que permite tener funcionalidades que no incluyen por defecto los navegadores.
- Web Components^[12]: Conjunto de normas que permite la creación de widgets o componentes reutilizables impulsado por la W3C.

2.9.1 Arquitectura

La arquitectura de un componente Polymer propio tiene las siguientes capas, en las que nos apoyamos cuando lo desarrollamos ^[14]:

- Elementos customizables: Polymer os ofrece componentes previamente desarrollados para poder utilizarlo sin necesidad de cambiarlo, como la integración de elementos Polymer para Youtube.
- Bibliotecas Polymer: Nos proporciona la sintaxis necesaria para definir componentes, como plantillas, etc..
- Web Components primitivos: proporcionan la base sobre la que construir los componentes personalizados. Polymer utiliza la biblioteca webcomponents.js que funciona en el 90% de los navegadores del mercado.

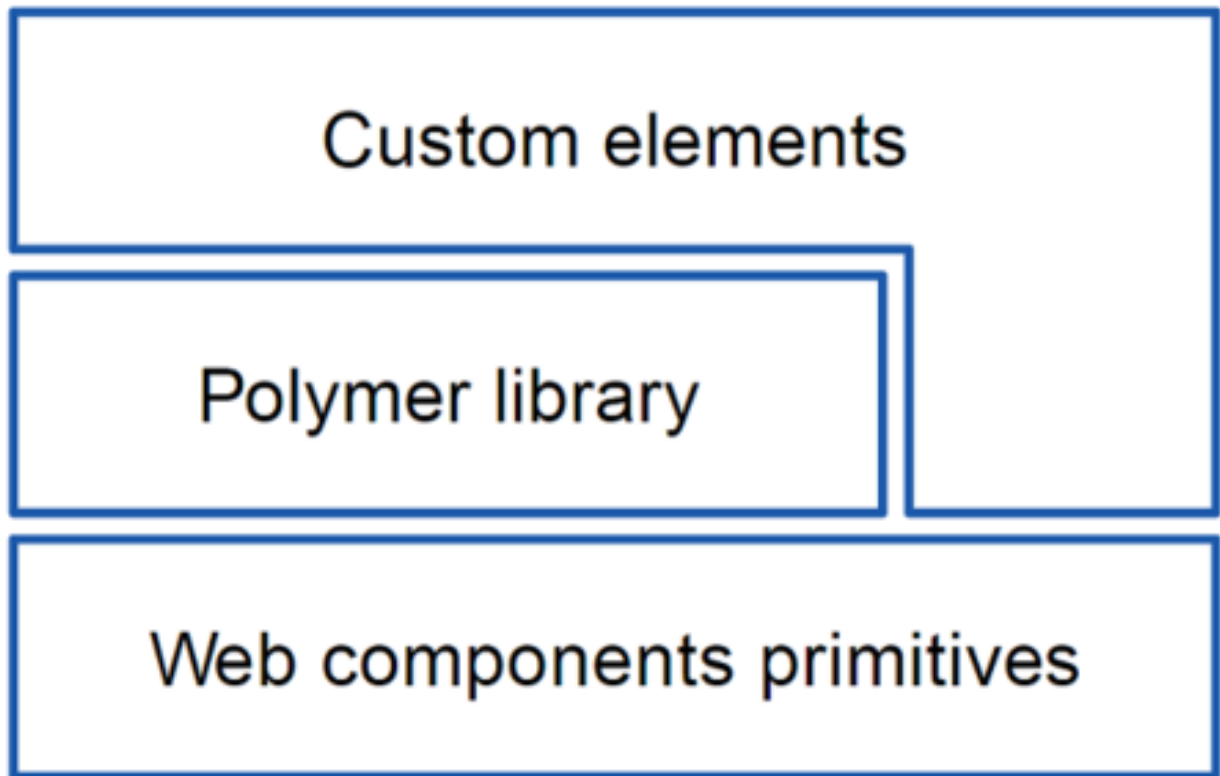


Ilustración 42: Arquitectura Polymer

2.10 Bower



Web: <http://bower.io/>

Última versión estable: 1.7.7¹

Licencia: BSD

Bower es un gestor de paquetes de frameworks, bibliotecas,... Originalmente está destinado para su uso con mpn, pero existe versiones que se pueden instalar y utilizar con Django ^[15]

Desarrollo de componentes web basados en Angular JS, Polymer, Node.js, Django y MongoDB para el pre-procesado, consumo y homogeneización de fuentes Open Data gubernamentales – Juan Francisco Hernández Sánchez

Se utilizará en el desarrollo con Django para importar todo lo necesario para el uso de Polymer.

3 DESARROLLO

3.1 Funcionalidad

La principal funcional que ofrece este desarrollo es el tratamiento de los datos públicos, obtenidos en formato XML con una estructura determinada y procesarlo a formato JSON, para que la webApp lo pueda procesar.

Estos datos obtenidos en el JSON, indicaran el estado actual de los parkings existentes en Pamplona.

Se mostrará en el mapa la localización, el nombre y el porcentaje de ocupación del parking, modificando el fondo del mensaje, para que el usuario pueda distinguirlo fácilmente con el lenguaje de los colores de los semáforos (verde, amarillo, rojo).

3.2 Formato Datos

3.2.1 Formato datos obtenidos en XML

Los datos los obtenemos en un documento XML que tiene la siguiente estructura para cada aparcamiento:

```
<APARCAMIENTO>

  <CODIGO>0</CODIGO>

  <NOMBRE>Baluarte</NOMBRE>

  <NOMBRE_CORTO>Baluarte</NOMBRE_CORTO>

  <DIRECCION>Plaza Baluarte s/n, Sotano 3 4, Pamplona</DIRECCION>

  <TELEFONO>948213293</TELEFONO>

  <LATITUD>42.813316</LATITUD>

  <LONGITUD>-1.647981</LONGITUD>

  <GESTORA>Empark Aparcamientos y Servicios</GESTORA>

  <HORARIO>
```

```
<FORMATO>MMHH</FORMATO>

<DESDE>0000</DESDE>

<HASTA>2400</HASTA>

</HORARIO>

<PLAZAS>

<TOTAL>900</TOTAL>

<LIBRES>370</LIBRES>

</PLAZAS>

<TARIFAS>

<ANYO>2016</ANYO>

<MONEDA>euro</MONEDA>

<RANGO>

<CODIGO>1</CODIGO>

<DESCRIPCION>Menos de 15 minutos</DESCRIPCION>

<IMPORTE>0,0471</IMPORTE>

</RANGO>

<RANGO>

<CODIGO>2</CODIGO>

<DESCRIPCION>De 16 a 30 minutos</DESCRIPCION>

<IMPORTE>0,0368</IMPORTE>

</RANGO>

<CODIGO>7</CODIGO>

<DESCRIPCION>De 601 a 1.440 minutos</DESCRIPCION>

<IMPORTE>0,0022</IMPORTE>

</RANGO>

<PRECIO_MAXIMO_DIARIO>10</PRECIO_MAXIMO_DIARIO>

<PRECIO_MINIMO_DIARIO>0</PRECIO_MINIMO_DIARIO>
```

```

    <PRECIOS_SF>

        <FORMATO_FECHA>mdd</FORMATO_FECHA>

        <FECHA_DESDE>-</FECHA_DESDE>

        <FECHA_HASTA>-</FECHA_HASTA>

        <PRECIO_MAXIMO>-</PRECIO_MAXIMO>

        <PRECIO_MINIMO>-</PRECIO_MINIMO>

    </PRECIOS_SF>

</TARIFAS>

</APARCAMIENTO>

```

Ilustración 43: Formato datos aparcamiento XML

Esa es la información relativa a cada aparcamiento, y el formato genérico del XML es el siguiente:

```

<APARCAMIENTOS>

    <ULTIMA_ACTUALIZACION>

        <FORMATO>aaaammddHHMM</FORMATO>

        <FECHA>201605171835</FECHA>

    </ULTIMA_ACTUALIZACION>

    <APARCAMIENTO>...</APARCAMIENTO>

    <APARCAMIENTO>...</APARCAMIENTO>

    <APARCAMIENTO>...</APARCAMIENTO>

    <APARCAMIENTO>...</APARCAMIENTO>

    <APARCAMIENTO>...</APARCAMIENTO>

    <APARCAMIENTO>...</APARCAMIENTO>

    <APARCAMIENTO>...</APARCAMIENTO>

    <APARCAMIENTO>...</APARCAMIENTO>

    <APARCAMIENTO>...</APARCAMIENTO>

```

```
<APARCAMIENTO>...</APARCAMIENTO>

</APARCAMIENTOS>
```

Ilustración 44: Formato datos aparcamientos XML

Informando también de la ultima hora de la actualización.

3.2.2 Formato datos procesados en JSON

El formato de los datos una vez procesados y accesibles por la webApp tienen la siguiente estructura:

```
{
  codigo: '9', //Código del parking
  latitud: '42.80594517263736', //Latitud del parking
  longitud: '-1.6659602522850037', //Longitud del parking
  nombre: 'Hospitales', //Nombre del parking
  estado: 'bajo', //Estado actual del parking para el color de fondo
  porcentaje: '19%' //Porcentaje de ocupación del parking
}
```

Ilustración 45: Formato datos JSON

La siguiente tabla muestra la relación entre los campos del XML y el JSON:

Tabla 3: Tabla mapeos campos

XML	JSON
Aparcamiento/Codigo	Codigo
Aparcamiento/Latitud	Latitud

Aparcamiento/Longitud	Longitud
Aparcamiento/Nombre_Corto	Nombre
-	Estado
-	Porcentaje

3.3 Desarrollo NodeJS y AngularJS

3.3.1 BackEnd NodeJS

3.3.1.1 Fichero Package.json

Como hemos visto en apartados anteriores, el fichero Package.json es el fichero de configuración. Indica el nombre, versión, del proyecto y las dependencias con módulos externos que se utilizan, y la versión de ellos.

El Package.json del proyecto es el siguiente:

```
{
  "name": "nodejsserver",
  "version": "0.0.1",
  "description": "Server NodeJS for TFM-ARQUITECTURA SOFTWARE developed by: Juan Francisco Hernandez Sanchez",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Juan Francisco Hernandez Sanchez",
  "license": "ISC",
  "dependencies": {
```

```
"express": "3.4.4",  
  
"http": "0.0.0",  
  
"jade": "*",  
  
"mongoose": "~3.8.8",  
  
"passport": "~0.2.0",  
  
"passport-facebook": "~1.0.2",  
  
"passport-twitter": "~1.0.2",  
  
"path": "^0.12.7",  
  
"xml2js": "^0.4.16"  
  
}  
  
}
```

Ilustración 46: Fichero Package.json

3.3.1.2 Esquema de la Base de Datos

En la base de datos solamente almacenamos la información del usuario, que se puede obtener desde el login de Twitter y Facebook. El esquema se encuentra en el fichero user.js de la ruta Models:

```
var UserSchema = new Schema({  
  
  name : String,  
  
  provider_id : {type: String, unique: true},  
  
  photo : String,  
  
  createdAt : {type: Date, default: Date.now}  
  
});
```

Ilustración 47: Schema Base de Datos

Este esquema lo utiliza Passport para realizar el login a través de las redes sociales.

Este esquema representaría, en el diagrama entidad-relación a la estructura de las tablas SQL.

Al ser necesario almacenar únicamente los datos del usuario habría una única tabla, con los siguientes valores:

- Name: Nombre del usuario proporcionado por la API de la red social.
- Provider_id: Nos indica que red social nos da la información. Está desarrollado para que sea Twitter o Facebook, como se explica en el siguiente apartado.
- Photo: Nos indica la url de la foto que vamos a mostrar. Esta url será de la red social que se utilice y no se almacena en el servidor ni en la base de datos como una imagen.
- Created_at: Nos indica la fecha de creación del registro en la base de datos. Se rellena con el Date exacto en el que se realiza la creación del registro.

3.3.1.3 Login con Passport

Se definen las estrategias a seguir a la hora de realizar el login, que vienen predefinidas en Passport.

Se exporta el módulo para que se pueda utilizar desde la app, haciendo que sea un código reutilizable.

Tiene métodos de serialización y deserialización para poder almacenar el objeto del usuario en la variable sesión y que sea accesible desde la aplicación.

No se ha implementado un login típico, de user y pass, para hacerlo más acorde a los tiempos actuales.

Para realizar el login con las redes sociales, se necesitan diversos parámetros que son necesarios para poder utilizarlo. Para ello hay que registrar previamente una aplicación en sus servidores, para poder así utilizar las API's que nos ofrecen para poder obtener los datos necesarios para realizar el login de los usuarios de esas redes sociales.

3.3.1.3.1 Registro de aplicaciones

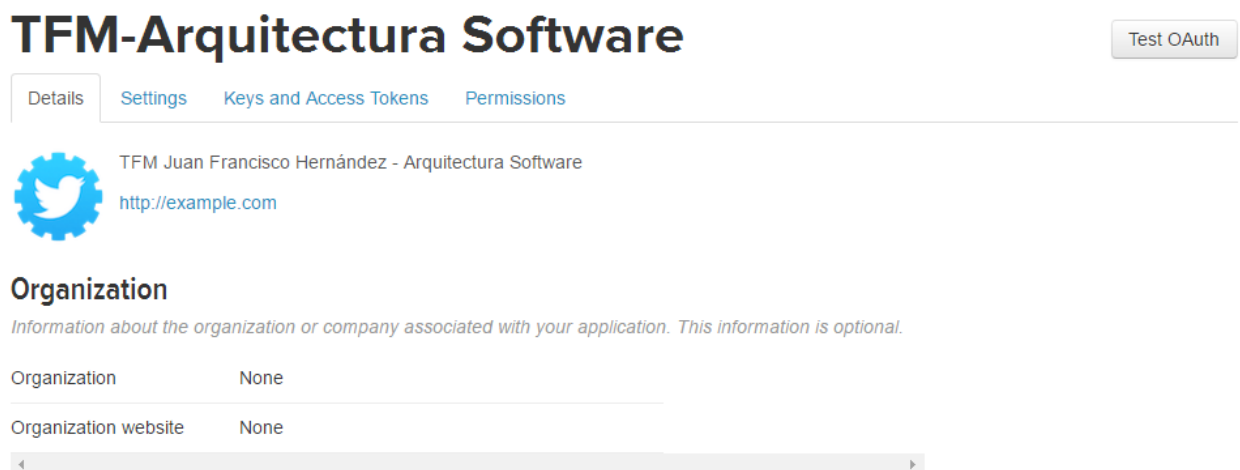
3.3.1.3.1.1 Registro aplicación con Twitter

Para registrar una aplicación en Twitter es necesario tener un usuario en esta red social, para vincular la nueva aplicación al usuario. Para ello, tenemos que ir a la siguiente url: <https://apps.twitter.com/app/new> y crear la aplicación nueva. Hay que introducir los siguientes datos:

- Name: Nombre de la aplicación. En nuestro caso es: TFM-Arquitectura Software.
- Description: Descripción de la aplicación. En nuestro caso es: TFM Juan Francisco Hernández - Arquitectura Software.
- Website: Es necesario definir un website donde se alojará la aplicación. Al ser una aplicación no productiva, se nos ofrece el dominio <http://example.com> que es el que utilizaremos

Una vez registrada la aplicación, obtendremos las keys que son necesarias para poder realizar el login.

La configuración de la aplicación que vamos a usar es la siguiente:



The screenshot shows the Twitter application settings page. At the top, the title 'TFM-Arquitectura Software' is displayed in large, bold letters. To the right of the title is a 'Test OAuth' button. Below the title is a navigation bar with four tabs: 'Details' (selected), 'Settings', 'Keys and Access Tokens', and 'Permissions'. Under the 'Details' tab, there is a Twitter bird icon, the application name 'TFM Juan Francisco Hernández - Arquitectura Software', and the website 'http://example.com'. Below this is a section titled 'Organization' with a subtitle 'Information about the organization or company associated with your application. This information is optional.' There are two input fields: 'Organization' with the value 'None' and 'Organization website' with the value 'None'. At the bottom of the form is a horizontal scrollbar.

Ilustración 48: App Twitter (1)

Y las claves necesarias son se encuentran en el apartado Keys and Access Tokens:

TFM-Arquitectura Software

[Details](#)[Settings](#)[Keys and Access Tokens](#)[Permissions](#)

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)

Consumer Secret (API Secret)

Access Level Read and write ([modify app permissions](#))

Owner

Owner ID

Ilustración 49: App Twitter(2)

Los datos que necesitaremos para realizar el login con la red social Twitter son el Costumer Key y Costumer Secret.

3.3.1.3.1.2 Registro aplicación con Facebook

Al igual que con Twitter, hay que crear una aplicación en Facebook con un usuario registrado. Para que el login con Facebook funcione hay que disponer de un dominio propio, por lo que no se ha podido realizar el login, pero se deja configurado para su futuro uso.

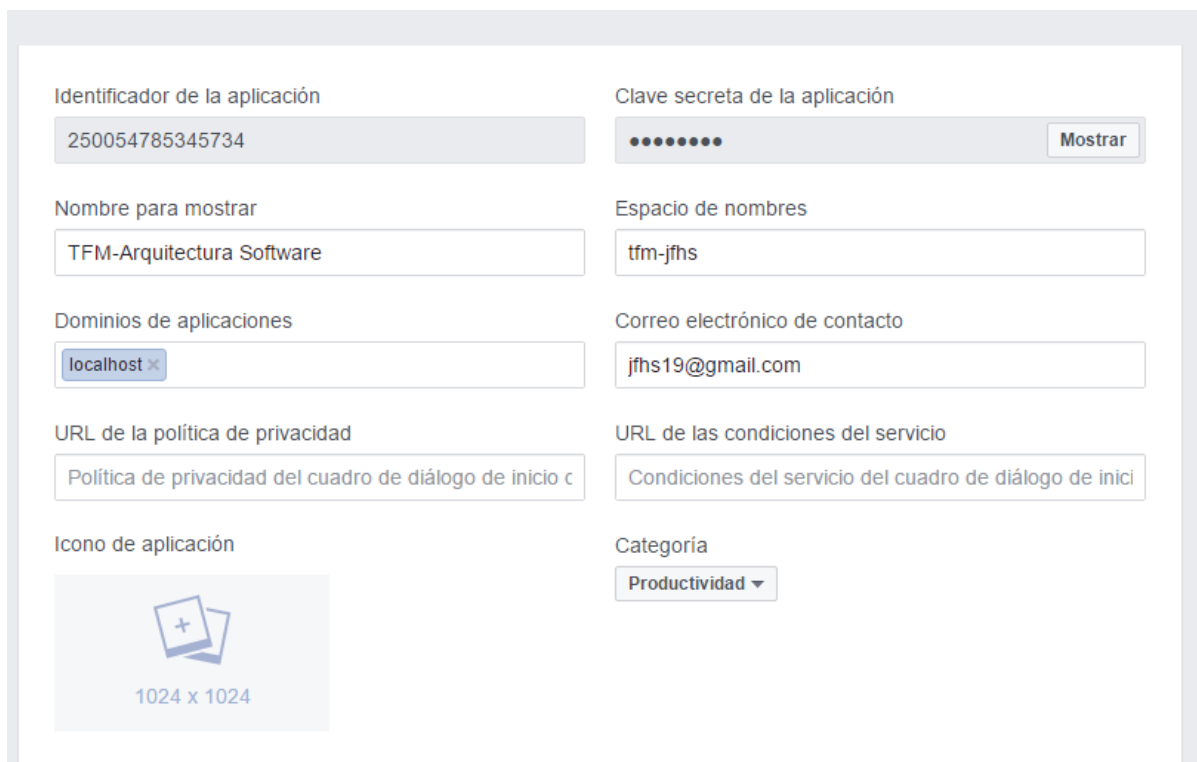
Para ello, vamos a la siguiente url:

<https://developers.facebook.com/quickstarts/?platform=web> y rellenamos los siguientes datos:

- Display Name: Nombre de la aplicación. En nuestro caso es: TFM-Arquitectura Software.
- App Domains: introduciremos localhost, ya que se trata de una aplicación de prueba.

- NameSpace: tfm-jfhs. Se ha simplificado debido a que la descripción utilizada en Twitter se excede de los 20 caracteres definidos.
- Site URL: http://localhost:5000

Una vez creada y configurada la app se queda del siguiente modo:



The image shows the Facebook App configuration page. It contains the following fields and values:

Field	Value
Identificador de la aplicación	250054785345734
Clave secreta de la aplicación (with a 'Mostrar' button)
Nombre para mostrar	TFM-Arquitectura Software
Espacio de nombres	tfm-jfhs
Dominios de aplicaciones	localhost x
Correo electrónico de contacto	jfhs19@gmail.com
URL de la política de privacidad	Política de privacidad del cuadro de diálogo de inicio c
URL de las condiciones del servicio	Condiciones del servicio del cuadro de diálogo de inici
Icono de aplicación	1024 x 1024 (with a placeholder icon)
Categoría	Productividad ▼

Ilustración 50: App Facebook

Los datos que necesitaremos para el login con la red social de Facebook son el identificador de la aplicación y la Clave secreta de la aplicación.

3.3.1.3.2 Login con Twitter

La configuración que utiliza Passport a la hora de realizar el login con las diversas redes sociales es a través de diversas “estrategias” que vienen por defecto en a la hora de la instalación del módulo Passport en NodeJS.

Para utilizar estas estrategias predefinidas hay que personalizar diversos valores, como las claves de la aplicación creada previamente en Twitter, las urls, la información a obtener de los diversos usuarios,...

La configuración de la estrategia utilizada con el login en Twitter es la siguiente:

```
passport.use(new TwitterStrategy({
    consumerKey : config.twitter.key,
    consumerSecret: config.twitter.secret,
    callbackURL : '/auth/twitter/callback'
}, function(accessToken, refreshToken, profile, done) {
//Busca en la BD si existe ese profile.id
    User.findOne({provider_id: profile.id}, function(err, user) {
        if(err) throw(err);
        // Si existe en la Base de Datos, lo devuelve
        if(!err && user!= null) return done(null, user);
        // Si no existe crea un nuevo objeto usuario
        var user = new User({
            provider_id : profile.id,
            provider : profile.provider,
            name : profile.displayName,
            photo : profile.photos[0].value //Twitter
            devuelve la foto en la primera posición
        });
        //Y lo almacena
        user.save(function(err) {
            if(err) throw err;
            done(null, user);
        });
    });
});
```

```
});  
  
});
```

Ilustración 51: Estrategia Passport Twitter

Necesita una serie de parámetros básicos a la hora de hacer el login, que son:

- **ConsumerKey:** Es una variable de la aplicación de Twitter que se genera al darla de alta en la red social. Hay que almacenarla en un lugar seguro, ya que, con ella, pueden acceder a la aplicación, por ese motivo se encuentra en el fichero config, asegurándonos que no se vea desde el código.
- **ConsumerSecret:** Al igual que consumerKey, es una variable que nos genera la API de Twitter al registrar la aplicación
- **CallbackURL:** Url a la que accederá una vez se haya realizado el login.

Una vez que se ha hecho la llamada a la API de twitter, y tenemos la información del usuario. Buscamos en la base de datos, si existe ese usuario, con el campo provider_id el valor de profile.id (que es el que nos ha devuelto twitter). Si existe en la base de datos ese registro, nos lo devuelve, sino existe lo crea, y se devuelve.

De la información que nos devuelve Twitter, se almacena el campo profile.id, profile.provider(Twitter) y la foto de perfil, que se encuentra en profile.photos[0], en el atributo value.

3.3.1.2 Login con Facebook

Al igual que con la red social de Twitter, Passport tiene predefinida una estrategia que hay que personalizarla a la hora de realizar el login.

La configuración de la estrategia utilizada es la siguiente :

```
passport.use(new FacebookStrategy({  
    clientID: config.facebook.id,  
    clientSecret: config.facebook.secret,  
    callbackURL: '/auth/facebook/callback',
```

```
        profileFields : ['id', 'displayName', 'photos']

    }, function(accessToken, refreshToken, profile, done) {

        User.findOne({provider_id: profile.id}, function(err, user) {

            if(err) throw(err);

            if(!err && user!= null) return done(null, user);

            //Si no lo encuentra, lo crea y lo almacena

            var user = new User({

                provider_id : profile.id,

                provider : profile.provider,

                name: profile.displayName,

                photo: profile.photos[0].value

            });

            user.save(function(err) {

                if(err) throw err;

                done(null, user);

            });

        });

    });

});
```

Ilustración 52: Estrategia Passport Facebook

Como podemos ver es muy similar a la utilizada con Twitter, diferenciando que Facebook nos lo devuelve como un array la información, y obtenemos solo la que queremos que es displayName, el nombre de la cuenta y la foto de perfil.

3.3.1.3 Obtención de los datos

Una vez que se ha definido como se realiza el login, hay que definir como se obtienen los datos desde el servidor. Para ello hay que realizar una petición HTTP GET a la URL donde está almacenado el XML que vamos a utilizar.

El método principal que se encarga de la obtención de la información, tiene la siguiente estructura:

```
//EndPoint encargado de devolver los datos.
app.get('/data', function(req, res) {
    //Generamos el Path del fichero
    var path = generatePath();
    console.log(path);
    //obtemenos el fichero original con getXML
    getXML(path, function(respuesta){
        var jsReturn = new Array();
        //Parseamos el xml
        parsearXML(path, function(resultado){
            //Eliminamos el fichero.
            deleteXML(path);
            //Devolvemos el JSON
            res.setHeader('Content-Type', 'application/json');
            res.json(resultado);
        })
    });
});
```

Ilustración 53: Obtención datos (1)

Como se puede ver, se realizan llamadas a diversos métodos con funcionalidades perfectamente definidas.

El funcionamiento general de este método, es la creación de un fichero temporal donde almacenaremos la información del XML, que utilizaremos para obtener la información necesaria de dicho XML y lo almacenaremos en un JSON que devolveremos como respuesta a la petición.

Una vez que ya no necesitamos el fichero XML temporal que hemos creado, lo eliminaremos del sistema.

3.3.1.4 getXML

Lo primero que realizamos, es generar una variable única para almacenar el fichero temporalmente, posteriormente lo eliminaremos. Esta variable es el timestamp:

```
function generatePath(){  
    //Genera el path del fichero  
  
    //Es el timestamp para asegurarnos de que es unico  
  
    console.log("Timestamp generated");  
  
    var timestamp = new Date().getTime();  
  
    var path = timestamp+".xml";  
  
    console.log(path);  
  
    return path;  
}
```

Ilustración 54: Obtención datos (2)

Una vez que tenemos el path donde vamos a escribir el fichero, lo que hacemos es obtenerlo desde la URL donde la fuente lo publica. Como una de las características de NodeJS, es su asincronidad, y no sabemos cuánto tiempo no llevará obtener el fichero, incluimos el flujo del proceso en una función callback, que solamente se ejecutará cuando esté finalizado el proceso asíncrono.


```
function getXML(path, callback){  
  //Funcion encargada de obtener el xml desde la ruta original.  
  //Recibe y devuelve el path donde se almacena el fichero  
  var file = fs.createWriteStream(path);  
  var request = http.get("http://www.pamplona.es/xml/parkings.xml",  
function(response) {  
    
    response.pipe(file);  
    console.log("File XML obtain");  
    callback(path);  
  });  
}
```

Ilustración 55: Obtención datos(3)

Este método lo que hace es generar una petición http GET del fichero y almacenándolo en el fichero que es la ruta path.

Una vez que ha finalizado, hace el callback al método que le ha llamado, devolviéndole la path.

3.3.1.5 ParsearXML

Este es quizá el método más importante, ya que es el encargado de pasar la información que tenemos en el XML, a un JSON que crearemos y que devolveremos.

Este JSON estará formado por una lista de todos los parkings, con la información necesaria.

Una vez que está el fichero en nuestro servidor, lo tratamos para poder obtener la información. Lo hacemos en el método `parsearXML`, que es el que contiene la lógica de la transformación de los datos en XML a JSON que leerá el Front.

```
function parsearXML(path, callback){  
    console.log(path + " desde parsear");  
    var parser = new xml2js.Parser();  
    //Leemos el fichero con la codificación de UTF-16  
    var fileContents = fs.readFileSync(path, 'ucs2');  
    var JSONreturn = [];  
    var libxml = require("libxmljs");  
    var xmlDoc = libxml.parseXmlString(fileContents);  
    //Obtenemos todos los nodos hijos  
    var parkings = xmlDoc.root().childNodes();  
    for(var i=1; i<parkings.length; i++){  
        //El primero lo ignoramos porque es datos de ultima actualización  
        var parkingBucle = parkings[i];  
        var parkingObjeto = getJSONFromParking(parkingBucle);  
        //Añadimos el objeto creado al array  
        JSONreturn.push(parkingObjeto);  
    }  
    console.log("Return: " + JSONreturn);  
    callback(JSONreturn);  
}
```

Ilustración 56: Obtención datos(4)

Este método, lee el fichero con la codificación 'ucs2', que es la correspondiente a UTF-16.

Utiliza el módulo `libxmljs`, para poder trabajar con XML en NodeJS. Una vez que ya tenemos acceso al XML, leemos los hijos, ignorando el primero, ya que es la fecha de la última actualización, y el resto son los parkings que hay. Vamos iterando sobre el bucle, llamando al método `getJSONFromParking`, enviándole el nodo correspondiente. Con el JSON obtenido, lo añadimos al array de Objetos que forman el JSON de retorno.

Una vez que tenemos el JSON formado lo enviamos por el callback al método que ha realizado la llamada.

3.3.1.6 DeleteXML

Al igual que el método `generatePath`, es un método muy simple, se limita a eliminar el fichero creado, para que no ocupe espacio en el filesystem:

```
function deleteXML(path){  
  //Elimina el fichero generado  
  //Recibe el path a borrar  
  fs.unlinkSync(path);  
  console.log("File deleted");  
}
```

Ilustración 57: Obtención datos (6)

3.3.2 Gestión de las URL

La gestión de las URL las realiza con `express`. Es necesario gestionar las URLs, para poder ofrecer un funcionamiento correcto de la aplicación, re direccionando las peticiones a los métodos previamente definidos para su funcionamiento.

Se encarga de generar los recursos necesarios para la gestión, indica quien se encarga de la gestión de la sesión, puerto de escucha, rutas, el motor de la vista, que se explicará detalladamente en otro apartado.

También se definirá el uso o no de cookies, en encode y demás configuración técnica necesaria para su correcto funcionamiento

```
// Iniciamos la aplicación Express
var app = express();

// Configuración (Puerto de escucha, sistema de plantillas, directorio de vistas,...)

app.set('port', process.env.PORT || 5000);
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');
app.use(express.logger('dev'));

//Configuramos App de Express
app.use(express.cookieParser());
app.use(express.urlencoded());
app.use(express.json());
app.use(express.methodOverride());

// Ruta de los archivos estáticos (HTML estáticos, JS, CSS,...)
app.use(express.static(path.join(__dirname, 'public')));

//Usaremos Sessions, con el valor de la clave secreta en config
app.use(express.session({ secret: config.secretkey }));

//Configuramos para que use passport para la gestión de la sesion
app.use(passport.initialize());
app.use(passport.session());
app.use(app.router);
```

Ilustración 58: Configuración Express (1)

Una vez definido las características de express, empieza la gestión de las peticiones URL. Según el tipo de petición GET, POST, ... llama al método correspondiente.

En este apartado de la configuración, se definen las URLs que va a utilizar el usuario y las llamadas internas de la aplicación entre sí.

Y finalmente indica que se encuentra levantado el servidor a la espera de peticiones en el puerto correspondiente:

```
//Login a twitter (enlace de login)
app.get('/auth/twitter', passport.authenticate('twitter'));

// Login a facebook(enlace de login)
app.get('/auth/facebook', passport.authenticate('facebook'));

// Ruta de callback del login de twitter. Si falla a raiz
app.get('/auth/twitter/callback', passport.authenticate('twitter',
  { successRedirect: '/', failureRedirect: '/' }
));

// Ruta de callback del login de facebook. Si falla a raiz.
app.get('/auth/facebook/callback', passport.authenticate('facebook',
  { successRedirect: '/', failureRedirect: '/' }
));

//EndPoint encargado de devolver los datos.
app.get('/data', function(req, res) {
  //Generamos el Path del fichero
  var path = generatePath();
  console.log(path);
  //obtemenos el fichero original con getXML
  getXML(path, function(respuesta){
    var jsReturn = new Array();
    //Parseamos el xml
    parsearXML(path, function(resultado){
```

```
//Eliminamos el fichero.  
  
deleteXML(path);  
  
//Devolvemos el JSON  
  
res.setHeader('Content-Type', 'application/json');  
  
res.json(resultado);  
  
})  
  
});  
  
})  
  
// Inicio del servidor  
app.listen(app.get('port'), function(){  
  console.log('App listening in: ' + app.get('port'));  
});
```

Ilustración 59: Configuración Express (2)

3.3.3 Front con Jade y AngularJS

Para desarrollar la parte FrontEnd, se han utilizado las siguientes tecnologías:

- Jade: a la hora de escribir el código front
- AngularJS: Para la gestión de la app
- AngularMaterial: Para el apartado visual de la App
- Css y BootStrap: Para el apartado visual de la ventana de Login

3.3.4 Jade

Como se ha explicado en apartados anteriores, Jade es un motor de plantillas que tiene su propio lenguaje de generación de código HTML, por lo tanto, al usar el motor Jade generamos el código HTML sin necesidad de utilizar las etiquetas propias de HTML, como son los típicos <body>, <a>, ...

También permite el uso de variables propias, como se verá a continuación, que según sus valores, se genera un código HTML distinto.

3.3.5 Ventana Login

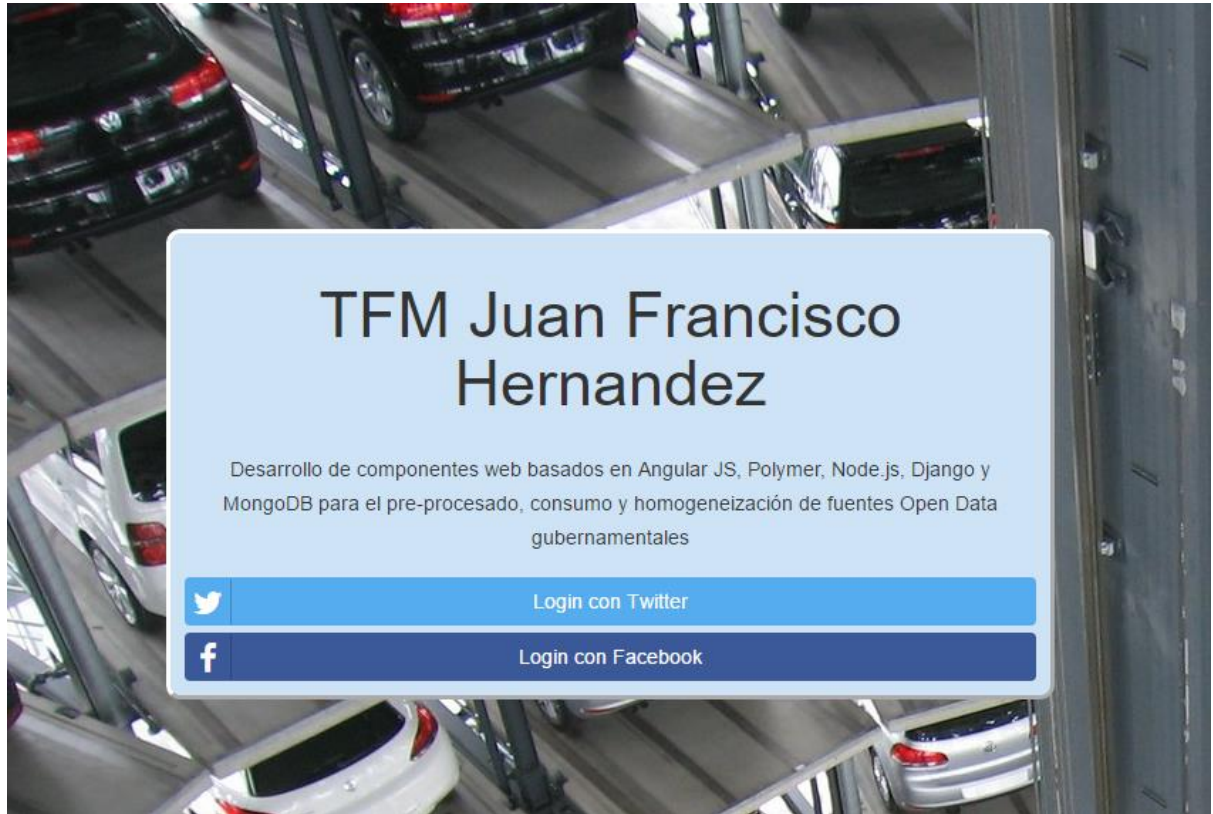


Ilustración 60: Captura Login

El fondo elegido es un parking de coches, acorde con la temática del desarrollo. Se muestra una información de la app y los diversos modos de login para la app.

El estilo de los botones, es Bootstrap.

El código en Jade es el siguiente:

```
#info
  h2 #{title}
  p #{description}
  a.btn.btn-block.btn-social.btn-twitter(href='auth/twitter')
    span.fa.fa-twitter
    | Login con Twitter
  a.btn.btn-block.btn-social.btn-facebook(href='auth/facebook')
```

span.fa.fa-facebook
| Login con Facebook

Ilustración 61: Jade login

Las variables con `#{variable}` están definidas en `/routes/index.js` y hacen referencia a las variables globales del sistema de rutas:

```
exports.index = function(req, res){  
  res.render('index', {  
    title: 'TFM Juan Francisco Hernandez',  
    description : 'Desarrollo de componentes web basados en Angular JS,  
Polymer, Node.js, Django y MongoDB para el pre-procesado, consumo y  
homogeneización de fuentes Open Data gubernamentales ',  
    user: req.user  
  });  
};
```

Ilustración 62: Variables index

El objeto 'user' que contiene toda la información del usuario y viaja en el 'request', y lo utilizamos a la hora de escoger que vista mostrar, si el login o la webApp. Este user es el que se almacena en la base de datos cuando se realiza el login a través de las redes sociales.

3.3.6 WebApp

Se ha querido diseñar la interfaz similar a las aplicaciones que se utilizan en los dispositivos Android, por ese motivo se ha utilizado AngularMaterial para la realización del menú superior y la interacción con el panel lateral, similar a cómo funcionan las aplicaciones Android.



Ilustración 63: Menú AngularMaterial (1)

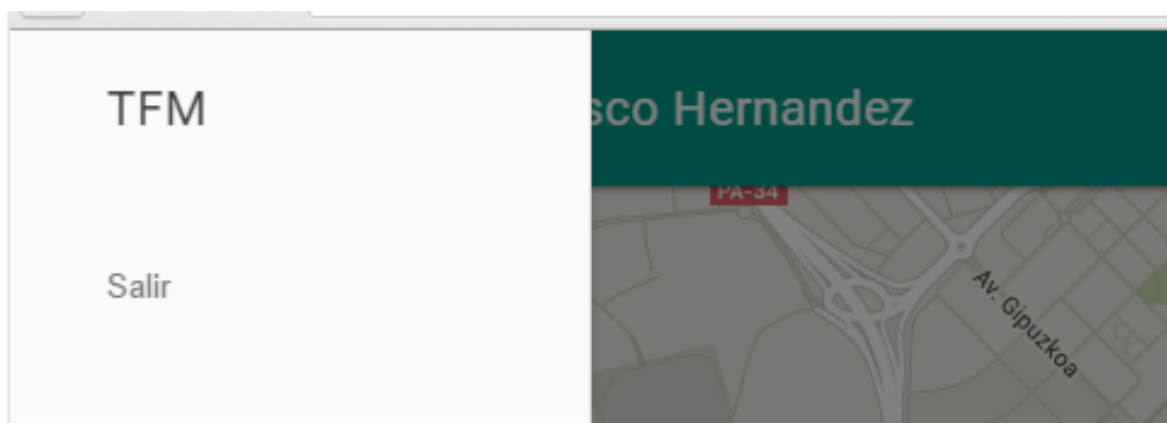


Ilustración 64: Menú AngularMaterial (2)

Se ha desarrollado con las directivas de AngularMaterial, que incluyen llamadas a directivas AngularJS para la interacción con el usuario. Cuando el ancho es poco, se redibuja, eliminando la barra, y dejando únicamente el menú para aprovechar más la pantalla.

Este funcionamiento es similar en los dispositivos Android, y se utiliza para obtener más visión de la aplicación que es realmente lo importante en el desarrollo y no la barra de menús, aun así, está accesible a través de las tres líneas horizontales

El menú queda de la siguiente manera cuando se reduce el ancho:

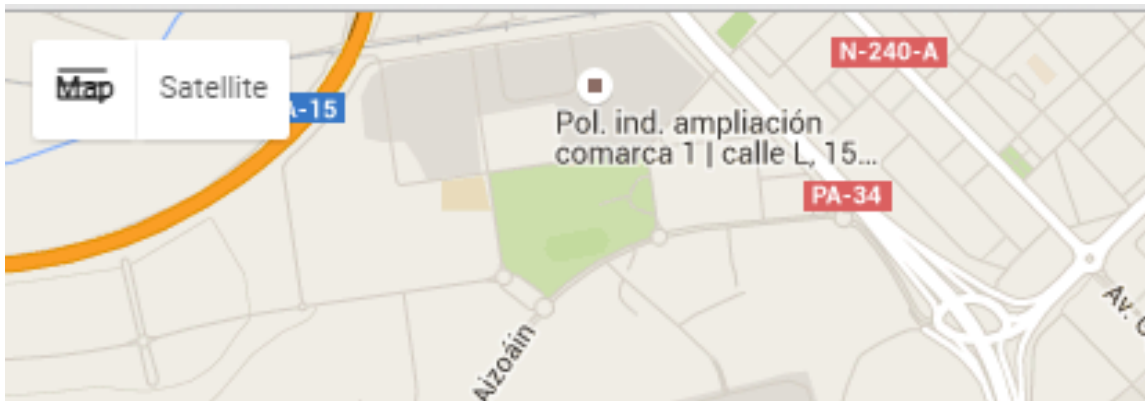


Ilustración 65: Menú AngularMaterial (3)

```
.mdl-layout.mdl-js-layout
  header.mdl-layout__header
    .mdl-layout__header-row
      span.mdl-layout-title #{title}
    .mdl-layout-spacer
  nav.mdl-navigation
    mdl-navigation #{user.name}
    mdl-navigation
      img(src="#{user.photo}")
  .mdl-layout__drawer
    span.mdl-layout-title TFM
    nav.mdl-navigation
      a.mdl-navigation__link(href='/logout') Salir
```

Ilustración 66: Jade Menú AngularMaterial

El resto de la webApp es un contenedor del map, con las directivas necesarias de AngularJS para su gestión.

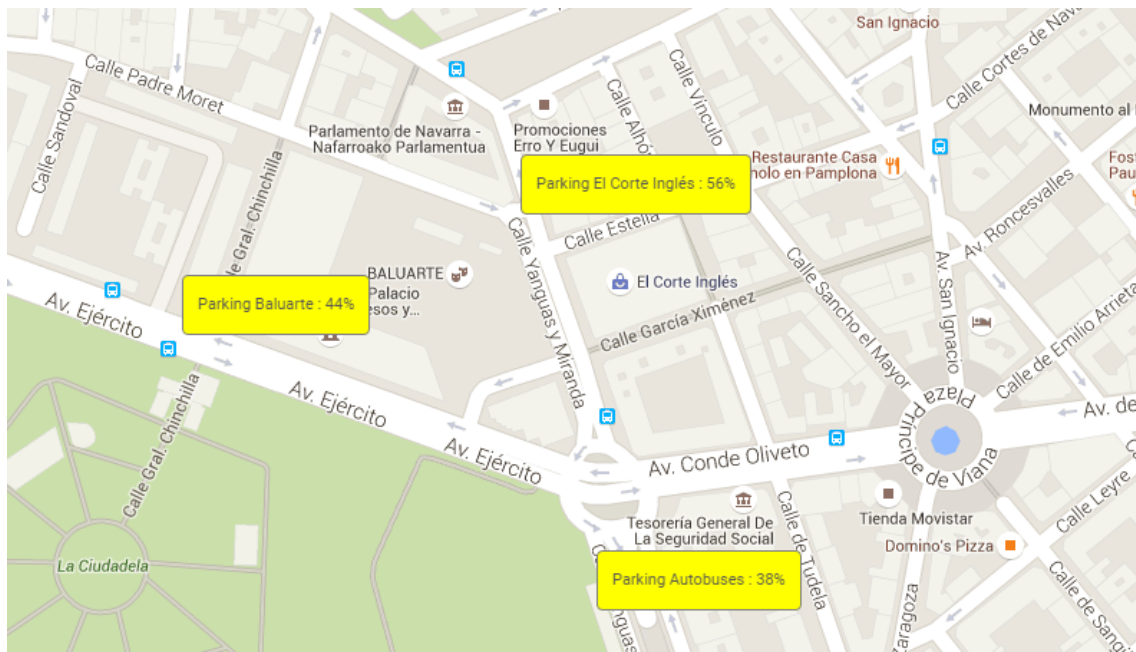


Ilustración 67: Captura WebApp

Se define un ng-controller y la etiqueta ng-map, que es el modo de cargar mapas con AngularJS.

Se crean tantos custom-markers según variables haya en el array positions del controller, de donde se obtiene el valor de la latitud, longitud y una breve descripción, del nombre y el porcentaje de ocupación, y varía la clase del marker, según el estado.

Lo que se hace en este fragmento del código es la estructura de la información que se va a mostrar, y que se rellenará individualmente cuando recorremos la lista de los parkings en el controlador del mapa, que se explicará a continuación.

```
#map-container(ng-controller='mapController as vm')
  ng-map(zoom='14', center='[42.8169, -1.6432]', default-style="false")
    custom-marker(ng-repeat='p in vm.positions',
position='{{p.latitud}}, {{p.longitud}}')
```

```
#descripcion(class= '{{p.estado}}') Parking {{p.nombre}} :  
{{p.porcentaje}}
```

Ilustración 68: Jade ng-map

El controlador del mapa se encuentra en el JavaScript.

```
var app = angular.module('myApp', ['ngMap']);  
  
app.controller('mapController', function($interval, $http) {  
  
    var vm = this;  
  
    vm.positions = [];  
  
    var generateMarkers = function() {  
  
        $http.get("data").  
  
        success(function(data) {  
  
            console.log(data);  
  
            console.log("length array" + data.length);  
  
            vm.positions=data;  
  
            console.log("vm.positions", vm.positions);  
  
        }).  
  
        error(function (data) {  
  
            console.log("fallo");  
  
        });  
  
    };  
  
    $interval(generateMarkers, 120000);  
  
});
```

Ilustración 69: Controlador AngularJS

El controlador se encarga de realizar peticiones a la URL /data, configurada en el BackEnd, para que devuelva la información en el formato adecuado para utilizarlo, eliminando toda la lógica de parte del cliente, para que sea lo más ligero posible.

Una vez que se ha recibido correctamente la información, se actualiza el array, y con el doble binding de AngularJS se actualiza en el mapa.

Este proceso se realiza cada dos minutos, que es cuando se refresca la información en el servidor donde obtenemos la información.

Al actualizarse la información en el mapa con el doble binding de AngularJS , es cargar la lista de los parkings, y así, como se ha definido la estructura que tendrán(nombre, color de fondo,...), se mostrará la información automáticamente la información que hemos recibido al tener la petición.

3.4 Desarrollo Django y Polymer

3.4.1 BackEnd con Django

Como se ha visto en el ejemplo del hola mundo, hay ficheros que son realmente importantes a la hora de configurar una aplicación, settings.py, urls.py y views.py.

3.4.2 Settings.py

Este es el archivo de configuración principal de cualquier desarrollo de Django, ya que tiene la información sobre las aplicaciones instaladas, donde buscar los ficheros estáticos,...

Por defecto viene configurado para generar la estructura básica de una aplicación, por este motivo se han realizado modificaciones en los siguientes apartados:

```
INSTALLED_APPS = (  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
  
    #APLICACIONES PROPIAS
```

```
'webapp',  
  
'djangobower',  
  
'social.apps.django_app.default',  
  
)
```

Ilustración 70: Settings.py INSTALLED_APPS

Se han añadido las siguientes apps:

- Webapp: Es el desarrollo como tal, y es el encargado de devolver la información.
- Djangobower: Es la versión de bower para poder utilizarse con django.
- Social.apps.django_app.default: Biblioteca que se encarga de la gestión de los usuarios a través de las redes sociales. Como se puede ver, a pesar de ser una sola “gran” aplicación django, está compuesta de diversas aplicaciones.

```
TEMPLATES = [  
  
    {  
  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
  
        'DIRS': [BASE_DIR + "/templates", ],  
  
        'APP_DIRS': True,  
  
        'OPTIONS': {  
  
            'context_processors': [  
  
                'django.template.context_processors.debug',  
  
                'django.template.context_processors.request',  
  
                'django.contrib.auth.context_processors.auth',  
  
                'django.contrib.messages.context_processors.messages',  
  
            ],  
  
        },  
  
    ],  
]
```

```
#TEMPLATES DE SOCIAL LOGIN

        'social.apps.django_app.context_processors.backends',
        'social.apps.django_app.context_processors.login_redirect',

    ],

},

},

]
```

Ilustración 71: Settings.py TEMPLATES

En este apartado se indica de donde cargar las plantillas para su funcionamiento. Se destaca, que se indica la ruta /templates, donde se encuentran las templates html que se le ofrece al usuario, y las templates de social, que forman parte de authSocial.

```
STATIC_URL = '/static/'

STATICFILES_DIRS = (

    os.path.join(BASE_DIR, "static"),

)
```

Ilustración 72: Settings.py STATICS

En estos apartados se indica donde se almacenan los ficheros estaticos. Estos ficheros suelen ser css, javascripts,... que se utilizan en los diversos htmls. Se almacenan en la ruta /static/ para que sean accesibles.

```
AUTHENTICATION_BACKENDS = (

    'social.backends.facebook.FacebookOAuth2',

    'social.backends.twitter.TwitterOAuth',

)
```

```
# Facebook Keys

SOCIAL_AUTH_FACEBOOK_KEY = 'XXX'

SOCIAL_AUTH_FACEBOOK_SECRET = 'XXX'

# Twitter Keys

SOCIAL_AUTH_TWITTER_KEY = 'XXX'

SOCIAL_AUTH_TWITTER_SECRET = 'XXX'
```

Ilustración 73: settings.py AUTHENTICATION

En este apartado se indica con que backends se va a realizar la autenticación de los usuarios. Entre los muchos que se puede, se realiza con Facebook y Twitter.

Es el nexo de unión entre la red social y nuestro desarrollo. Es similar al uso de Passport en el desarrollo con NodeJS. Esta aplicación no necesita generar ningún esquema del usuario, ni nada similar, ya que ella se encarga de obtener y almacenar la información de modo persistente en el sistema a través del gestor de identidades que ofrece django.

Este gestor de identidades se encarga de almacenar en la base de datos que gestiona automáticamente Django la información necesaria. Es una base de datos de tipo SQL, esto simplifica la gestión de los usuarios.

También se indican las claves de las apps que se han generado, que son necesarias para la autenticación.

3.4.3 urls.py

Este archivo contiene los patrones de las urls a las que da servicio, y como se debe comportar, que vista mostrar, que modulo llamar, etc.

```
urlpatterns = [

    url('', include('social.apps.django_app.urls', namespace='social')),

    url(r'^data/$', 'webapp.views.webapp'),
```



```
url(r'^$', TemplateView.as_view(template_name="home.html"),
name="home"),

url(r'^users/login/$', 'django.contrib.auth.views.login', {'next_page':
'/mapa'}),

url(r'^users/logout/$', 'django.contrib.auth.views.logout',
{'next_page': '/'}, name="user-logout"),
]
```

Ilustración 74: urls.py URLPATTERNS

Se incluyen todas las urls del paquete 'social.apps.django_app.urls' para poder gestionar los usuarios con el framework instalado para ello.

Se han definido las siguientes urls:

- /data: Cuando recibimos una petición a esta url, se llama al método webapp de la app webapp, que se explicará más adelante su funcionalidad.
- /: Es la raíz, cuando recibe una petición, muestra la plantilla de home.html, que contiene lo necesario para el login
- /users/login: Se encarga de realizar el login de los usuarios.
- /users/logout: Se encarga de realizar el logout de los usuarios.

El otro apartado del fichero urls.py, es el encargado de redirigir las peticiones a los ficheros estáticos del sistema, para poder utilizarlos en el servidor.

```
from django.contrib.staticfiles.urls import staticfiles_urlpatterns

urlpatterns += staticfiles_urlpatterns()
```

Ilustración 75: urls.py STATICFILES

Sin esta redirección sería imposible usar los css, javascripts y sobre todo las bibliotecas de Polymer que se utilizan para la interfaz.

3.4.4 views.py

Es el fichero que tiene la lógica de la webapp, que se le llama a través de la url data. Es el siguiente método:

```
def webapp(request):  
#GENERAMOS EL PATH  
    fileName=generatePath()  
    return_json = []  
#OBTENEMOS EL FICHERO DESDE LA URL  
    getXML(fileName)  
#TENEMOS EL FICHERO. LO PARSEAMOS  
    return_json=parseXML(fileName)  
#CODIFICAMOS EN JSON LOS DATOS A DEVOLVER  
    json_return = demjson.encode(return_json)  
    print(json_return)  
#ELIMINAMOS EL FICHERO XML DEL FILESYSTEM  
    deleteXML(fileName)  
    return HttpResponse(json_return, content_type="application/json")
```

Ilustración 76: views.py webapp(request)

Es el método general que contiene la lógica para la obtención y tratamiento de los datos.

A grandes rasgos, el funcionamiento es similar al desarrollado anteriormente, realiza una petición HTTP para obtener los datos, los almacena en un fichero temporal para obtener la información y enviarla en formato JSON como respuesta de la petición.

Finalmente borra el fichero XML temporal, porque ya no es necesario.

Está modularizado para que sea más sencilla su lectura y la implementación.

La primera llamada del método es a `generatePath`, donde generamos un path donde almacenar el fichero. Para asegurarnos de que es único utilizamos el timestamp en milisegundos al igual que en el desarrollo de `node.js`. Se devuelve el valor para utilizarlo en el resto del programa:

```
def generatePath():  
    path = int(round(time.time() * 1000))  
    path = str(path) + ".xml"  
    print path  
    return path
```

Ilustración 77: views.py generatePath

Una vez que tenemos la ruta donde almacenaremos el fichero XML, llamamos al método `getXML` con la ruta como parámetro. Este método se encarga de realizar la petición a la url donde obtenemos el fichero con la información (<http://www.pamplona.es/xml/parkings.xml>) y escribirla en el fichero correspondiente:

```
def getXML(path):  
    url = 'http://www.pamplona.es/xml/parkings.xml'  
    response = urllib2.urlopen(url)  
    html = response.read()  
    file = open(path, 'wb')  
    file.write(html)  
    file.close()
```

Ilustración 78: views.py getXML

Una vez que ya tenemos el fichero nos toca obtener la información de él y parsearla a un formato JSON, para ello llamamos al método `parsearXML`, con el path del fichero XML.

Este método, lee la información del fichero con el método `parse` del modulo `etree` incluido en Python, obteniendo todos los nodos 'APARCAMIENTO' que tiene la información de los aparcamientos. Y se recorre con un bucle `for` todos los elementos, llamando al método `getJSONFromParking`, que devuelve la información ya en formato JSON. Esta información la añadimos al array que estamos generando de objetos JSON, que devolveremos al método principal.

```
def parsearXML(path):  
    tree = etree.parse(path)  
    root = tree.getroot()  
    parkings_array = root.findall('APARCAMIENTO')  
    parkings_json=[]  
    for parking in parkings_array:  
        parking_json=[]  
        parking_json=getJSONFromParking(parking)  
        parkings_json.append(parking_json)  
    return parkings_json
```

Ilustración 79: views.py parsearXML(path)

El método `getJSONFromParking` es el método que de verdad se encarga de obtener la información. Extrae la información buscando por el nombre de la variable que queremos y obteniendo el texto. Como la información de las plazas libres y totales se encuentra en un nivel inferior, se hace el mismo procedimiento, se busca el nodo y sobre ese nodo, obtenemos los datos.

Una vez que tenemos los datos, hacemos los cálculos necesarios para obtener la información que queremos mostrar. Para ello, comprobamos antes de todo, que los valores de las plazas son dígitos, sino es así las igualamos para que no nos muestre información incompleta.

Calculamos el estado del parking, con los valores que tenemos, y generamos el json que devolvemos con la información:

```
def getJSONFromParking(parking):  
    parking_json= []  
    print parking  
    estado=''  
    codigo = parking.find('CODIGO').text  
    latitud = parking.find('LATITUD').text  
    longitud = parking.find('LONGITUD').text  
    nombre = parking.find('NOMBRE_CORTO').text  
    #OBTENEMOS LOS VALORES DE OCUPADO Y TOTAL  
    plazas = parking.find('PLAZAS')  
    total = plazas.find('TOTAL').text  
    libre = plazas.find('LIBRES').text  
    #COMPROBAMOS QUE SEA NUMERICO, PUEDE ENVIAR DATOS FALSOS  
    if libre.isdigit() is False or total.isdigit() is False:  
        libre=1  
        total=1  
    porc = float(libre)/float(total) *100  
    porcentaje_String = str(int(porc)) + "%"  
    if porc < 33 :  
        estado = 'bajo'  
    if porc > 34 and porc < 66:  
        estado = 'medio'  
    if porc > 67 :  
        estado = 'alto'
```

```
parking_json.append({'total' : total , 'libres' : libre , 'codigo' :  
codigo, 'latitud' : latitud, 'longitud' : longitud, 'nombre' : nombre ,  
'porcentaje':porcentaje_String, 'estado' : estado })  
  
return parking_json
```

Ilustración 80: views.py getJSONFromParking

Una vez que tenemos el JSON generado, eliminamos el fichero XML generado, con el método deleteXML, que simplemente lo eliminando del filesystem.

```
def deleteXML(path):  
  
    os.remove(path)
```

Ilustración 81: views.py deleteXML

Devolvemos el fichero JSON generando, especificando que el tipo de la información que devolvemos es de tipo json.

3.4.5 Front con Polymer

Se ha integrado Polymer con el sistema de templates, al igual que se hizo en el desarrollo con Node.JS para integrar las dos vistas (login y webapp) en un fichero html.

Se indica la carga de los ficheros estáticos almacenados en el servidor, con la siguiente directiva “{% load staticfiles %}” y haciendo haciendo referencia a que son ficheros estáticos a la hora de importación de los ficheros css, JavaScript y Polymer necesarios:

```
{% load staticfiles %}  
  
<script src="{% static 'bower_components/webcomponentsjs/webcomponents-  
lite.min.js' %}"></script>  
  
<link rel="import" href="{% static 'bower_components/google-map/google-  
map.html' %}">  
  
<script src="{% static 'js/map.js' %}"></script>  
  
<link rel="stylesheet" src="{% static 'css/map.css' %}"></script>  
  
<link rel="stylesheet" src="{% static 'css/bootstrap-social.css'  
%}"></script>
```

```
<link rel="import" href="{% static  
'bower_components/polymer/polymer.html' %}">  
  
<link rel="import" href="{% static 'bower_components/paper-toolbar/paper-  
toolbar.html' %}">
```

Ilustración 82: Carga ficheros estaticos

La diferencia a la hora de cargar el html reside en si existe la variable user o no. Si existe, se carga la parte de la webapp y en el caso de que no exista, se carga la ventana de login.

3.4.6 Ventana Login

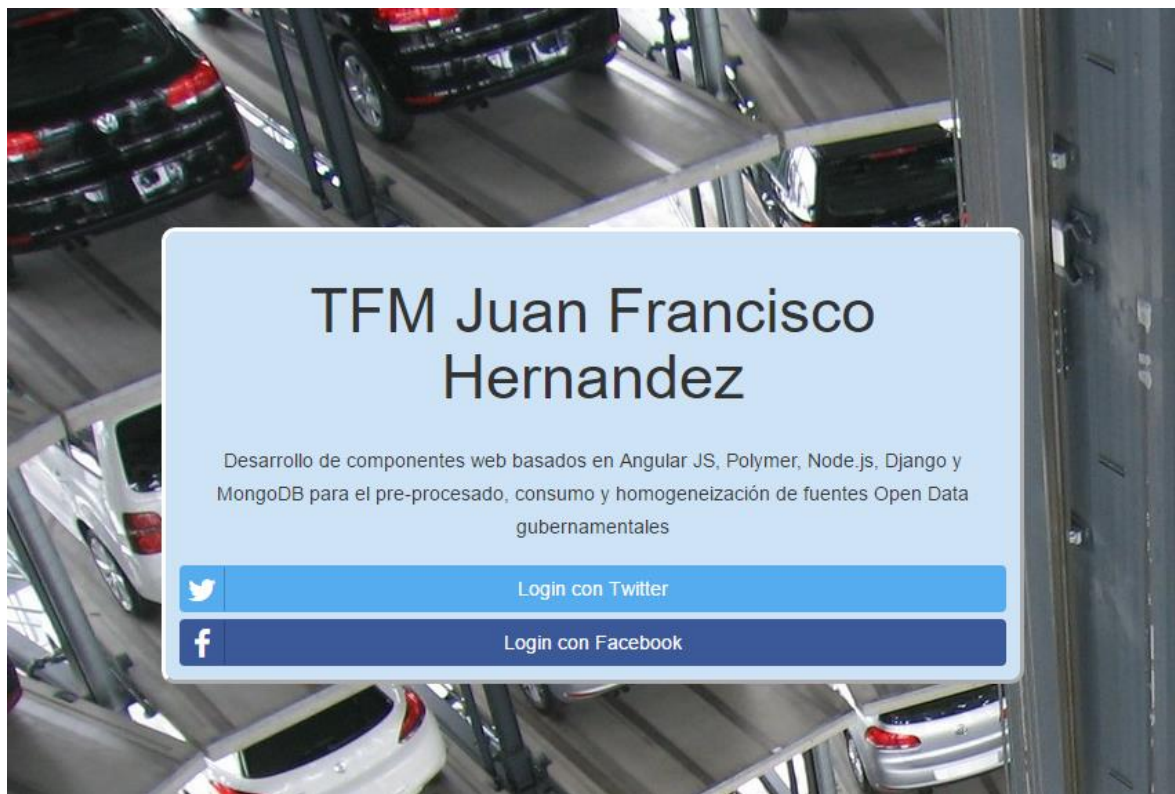


Ilustración 83: Ventana Login

Se ha decidido utilizar la misma interfaz que en el caso anterior, ya que el código html en este caso es el mismo, pero se ha desarrollado directamente en HTML:

```
<div id="info" class="info">
```

```
<h1>TFM Juan Francisco Hernandez</h1>

<p>Desarrollo de componentes web basados en Angular JS, Polymer,
Node.js, Django y MongoDB para el pre-procesado, consumo y homogeneización
de fuentes Open Data gubernamentales</p>

<a href="{% url 'social:begin' 'twitter' %}?next={{ request.path }}"
class="btn btn-block btn-social btn-twitter">

    <span class="fa fa-twitter"></span> Login con Twitter</a>

<a "{% url 'social:begin' 'facebook' %}?next={{ request.path }}"
class="btn btn-block btn-social btn-facebook">

    <span class="fa fa-facebook"></span> Login con Facebook</a>

</div>
```

Ilustración 84: Ventana login código

3.4.7 WebApp

Se ha utilizado la misma representación, una barra horizontal superior, para mostrar el usuario y poder realizar el logout siguiendo los principios de material desing. En este caso no se muestra ningún menú desplegable ni el icono del usuario, ya que el api de la gestión de usuarios solamente nos ofrece el nombre de usuario.



Ilustración 85: Menú

Se ha utilizado paper-toolbar que nos ofrece Polymer para generar estos menús. Se ha utilizado un icono de exit, almacenado en un cdn remoto.

```
<paper-toolbar>

    <paper-icon-button icon="menu"><a href="{% url 'user-logout' %}">
```



```


</a></paper-icon-button>

<span class="title">TFM</span>

<paper-icon-button icon="refresh"></paper-icon-button>

<paper-icon-button icon="add">{{ user.get_full_name }}</paper-icon-button>

</paper-toolbar>
```

Ilustración 86: WebApp código (1)

El resto es el contenedor de google Maps, que se utilizará para mostrar la información, indicándole las coordenadas donde centrar la imagen.

```
<google-map id='map' latitude="42.8169" longitude="-1.6432" zoom="15">

</google-map>
```

Ilustración 87: WebApp código (2)

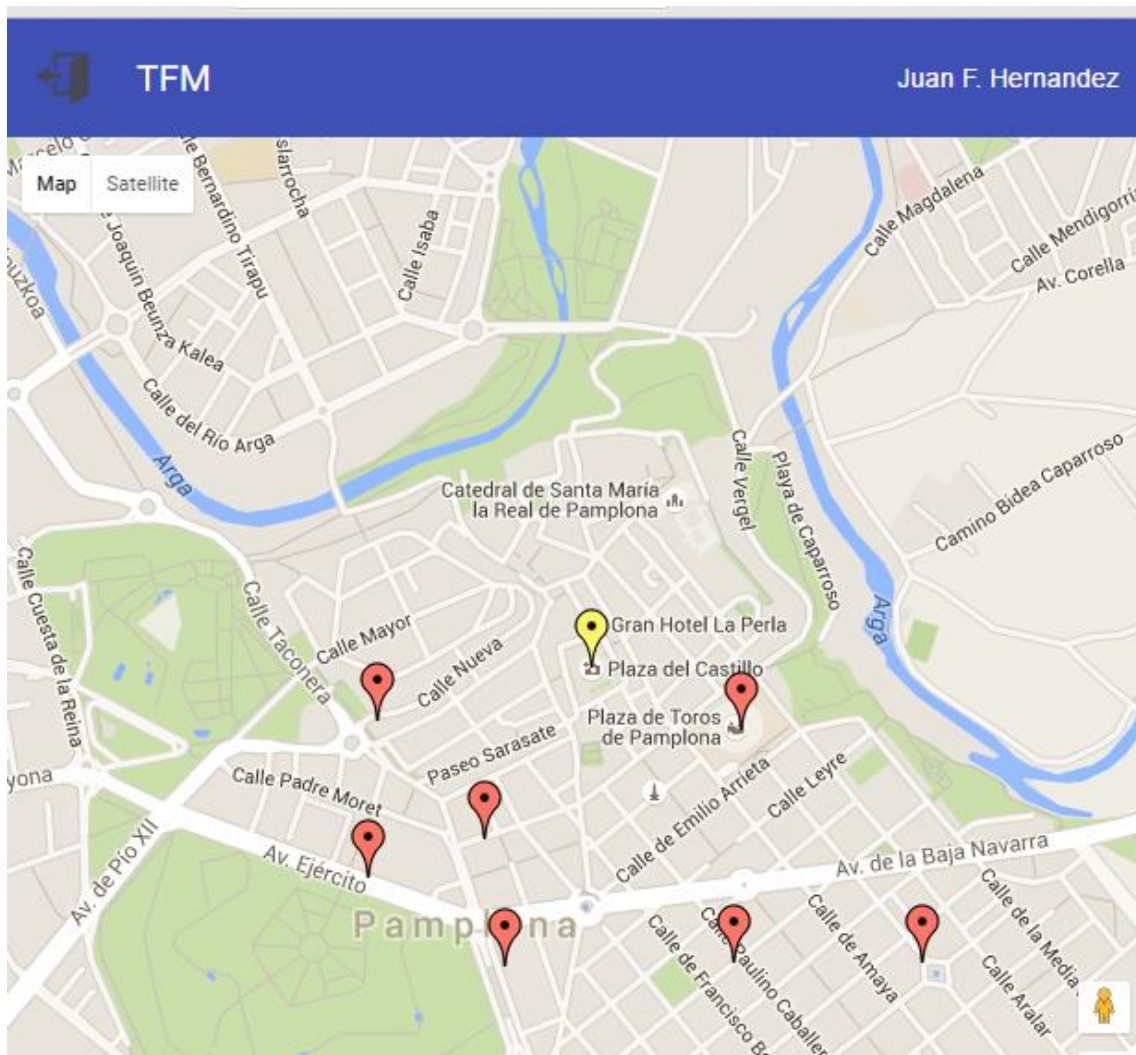


Ilustración 88: Captura WebApp

El controlador de la información se encuentra en el fichero map.js

Se define una función que se realice cada 2 minutos, que es cuando se recarga la información, realizando una llamada por AJAX al endpoint data del servidor que nos devuelve la información en JSON.

```
setInterval(function getData(){
    petición();
}, 120000);
```

Ilustración 89: Controlador JavaScript (1)

Se llama al método petición, que es el encargado de realizar las peticiones. También se le llama al cargar la página, para obtener la información:

```
function peticion(){  
    var xhttp = new XMLHttpRequest();  
    xhttp.onreadystatechange = function() {  
        if (xhttp.readyState == 4 && xhttp.status == 200) {  
            console.log(xhttp.responseText);  
            var data = JSON.parse(xhttp.responseText);  
            pintarMarca(data);  
        }  
    };  
    xhttp.open("GET", "/data", true);  
    xhttp.send();  
}
```

Ilustración 90: peticion()

Se llama a la función pintarMarcar, que carga la información del mapa, y recorre el array de los objetos.

```
var map = new google.maps.Map(document.getElementById('map'), {  
    zoom: 15,  
    center: new google.maps.LatLng(42.8169, -1.6432),  
    mapTypeId: google.maps.MapTypeId.ROADMAP  
});  
  
var infowindow = new google.maps.InfoWindow();
```

Ilustración 91: pintarMarca(1)

Según el estado del parking (bajo, medio, alto), se carga un icono de color diferente (verde, amarillo, rojo), que se almacena en la variable color_icon

```
for(var i=0; i< data.length; i++) {  
    var parking = new Object();  
    parking = data[i][0];  
    var color_icon;  
  
    if(parking.estado == 'bajo'){  
        console.log("bajo");  
        color_icon=new  
google.maps.MarkerImage('http://maps.google.com/mapfiles/ms/icons/green-  
dot.png');  
    }  
    if(parking.estado=='medio'){  
        console.log("medio");  
        color_icon= new  
google.maps.MarkerImage('http://maps.google.com/mapfiles/ms/icons/yellow-  
dot.png');  
    }  
    if(parking.estado=='alto'){  
        console.log("alto");  
        color_icon= new  
google.maps.MarkerImage('http://maps.google.com/mapfiles/ms/icons/red-  
dot.png');  
    }  
}
```

Ilustración 92: pintarMarca (2)

Una vez que está decidido el icono, se crea el marker correspondiente, indicándole la posición con las coordenadas, y el código HTML a mostrar cuando se interactúa con el.

En este caso, es el nombre del parking y el porcentaje de ocupación obtenido en la llamada a /data.

```
markers[i] = new google.maps.Marker({  
    position: {lat:parseFloat(parking.latitud),  
    lng:parseFloat(parking.longitud)},  
    map: map,  
    html: '<div id="contentInfoWindow" class="contentMap"><div  
class="contentTxt">Parking '+parking.nombre +' : '+parking.porcentaje  
+'</div></div>',  
    id: parking.codigo,  
    icon:color_icon,  
});
```

Ilustración 93: pintarMarca(3)

Una vez que se ha añadido el marker al array, se genera un listener para la interacción con el usuario (click en él para mostrar la información):

```
google.maps.event.addListener(markers[i], 'click', function(){  
    var infowindow = new google.maps.InfoWindow({  
        id: this.id,  
        content:this.html,  
        position:this.getPosition(),  
    });  
    google.maps.event.addListenerOnce(infowindow, 'closeclick', function(){  
        markers[this.id].setVisible(true);  
    });  
    this.setVisible(false);  
    infowindow.open(map);  
});
```

```
}
```

Ilustración 94: pintarMarca(4)

El resultado de añadir el listener es el siguiente:



Ilustración 95: captura WebApp (2)

4 CONCLUSIONES Y LÍNEAS FUTURAS

En cuanto a conclusiones, se ha realizado una comparativa sobre la complejidad a la hora de desarrollar las dos diversas aplicaciones.

4.1 Comparativa BackEnds

Se va a realizar una comparativa centrada en los backends. La primera parte, se selecciona con cual ha sido más sencillo realizar el desarrollo, y en la segunda parte puntos fuertes y débiles de cada una de las opciones:

Tabla 4: Comparativa BackEnds

	NodeJS	Django
Peticiones HTTP		X
Tratamiento XML		X
Generación JSON	X	
Tratamiento ficheros	X	
Gestión de URLs		X
Gestión de Templates		X
Uso de Módulos Externos	X	
Cantidad y calidad de modelos externos	X	
Login Social		X

Facilidad de desarrollo	Es en JavaScript, por lo que es fácil familiarizarse con él.	Es en Python, es complicado acostumbrarse al estilo de programación, pero una vez que se controla, es muy cómodo.
Documentación en la red	Existe documentación de sobra, con ejemplos y soluciones para la mayoría de los problemas más comunes.	Existe documentación con ejemplos y soluciones para los problemas más comunes.
Módulos externos	Existen muchos, bien documentados y con alternativas para realizar la misma función.	Existen muchos, bien documentados pero con pocas alternativas para realizar la misma función.
Generación del proyecto	Sencilla, con un fichero .js se puede comenzar sin problemas.	Compleja, te genera el proyecto entero nada más comenzar.
Asincrónica	Complejo, ya que hay que introducir el código siguiente dentro del método.	Gestionado por Python.

4.2 Comparativa FrontEnds

4.2.1 Comparativa técnica.

Tabla 5: Comparativa FrontEnds

	AngularJS + Jade + MaterialDesing	Polymer
Construcción del menú		X
Generación Mapa	X	
Control del Mapa	X	
Uso de plantillas	Complejo con el uso de Jade, ya que toca modificar la carga de los scripts, css,..	Sencillo con Django, solamente la carga de los estaticos es compleja.
Documentación	Existe mucha documentación y variada para cada problema planteado.	Escasa y ejemplos muy breves.
Realización de Peticiones AJAX	Algo complejo con Angular.	Sencilla
Binding JS & HTML	Sencillo con el doble binding de Angular.	Complejo, al ser directamente JS y HTML.

4.2.2 Comparativa visual.

En este apartado se hace una comparativa visual de las dos aplicaciones desarrolladas.

4.2.2.1 Menús

En el desarrollo con AngularJS se dispone del menú superior y un menú deslizante que aparece en el lateral con la opción de salir. Es una de las características que ofrece AngularJS sin tener que introducir código en exceso para ello. En cuanto al desarrollo en Polymer, realizar esta funcionalidad sería demasiado complejo, por lo que se sustituyó por un icono de una puerta, que hace referencia a Salir. Este icono está relacionado en todos los lugares donde se utiliza, bien en aplicaciones o en otros lugares, como museos, por ejemplo, que representa el hecho de salir.



Ilustración 96: Menú Angular



Ilustración 97: Menú Polymer

4.2.2.2 Representación de la información

El módulo en AngularJS utilizado permite recorrer arrays con facilidad, generar letreros con la información de modo sencillo, y por ese motivo se eligió representar la información así. Al tener el fondo de un color diferente se distingue a golpe de vista el estado de todos los parkings.

También esta representación reduce la interacción por parte del usuario, ya que se muestra siempre la información en pantalla, sin tener que seleccionar la localización específica del mapa.

Esta representación es recomendable para el grupo de usuarios que, conociendo la localización de los parkings, quiere saber el estado de ellos, sin tener que seleccionarlos individualmente.

Lo malo de esta representación, es el hecho de que el letrero del estado se muestra en la posición del parking, pero al ser de un tamaño “considerable” no se puede apreciar exactamente la posición del parking.

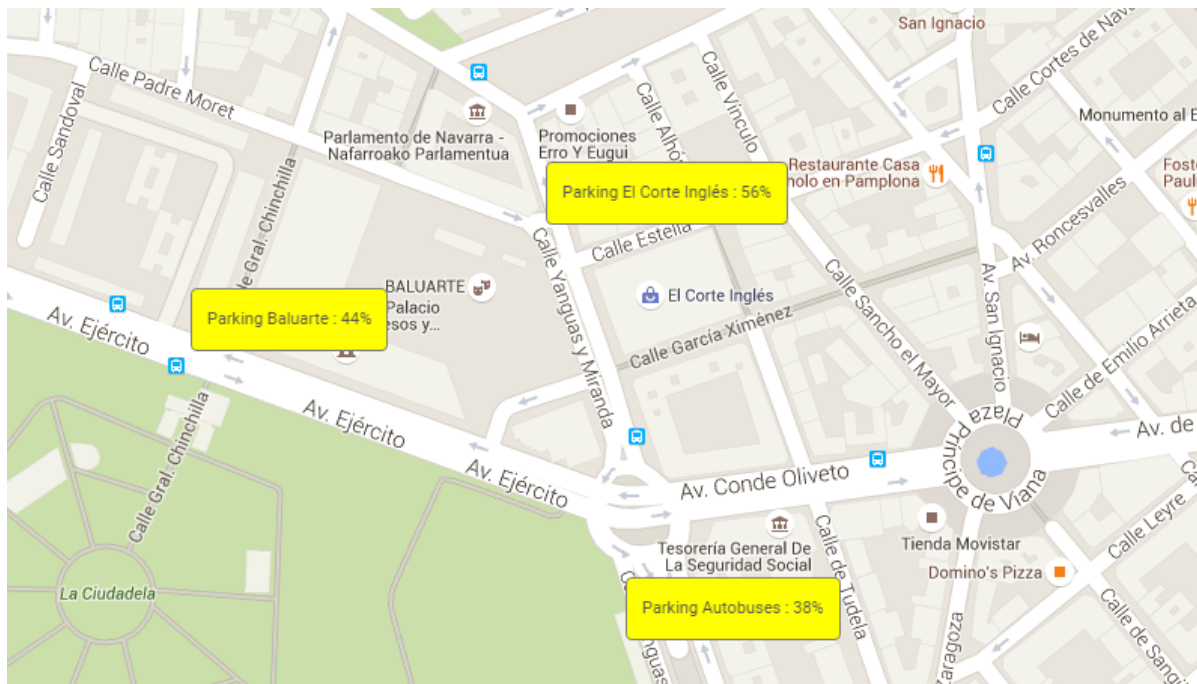


Ilustración 98: Comparativa (1)

En la aplicación desarrollada con Polymer, se ha querido mostrar la localización de los parkings, con la representación tradicional de las localizaciones en los mapas. Para distinguir el estado de las plazas, se ha escogido el modificar el color de cursor, pero sin mostrar la información.

Para mostrar la información el usuario tiene que interactuar con la aplicación, haciendo click sobre el parking elegido para saber su porcentaje de ocupación y el nombre del parking.

Esta interfaz es recomendable para la persona que quiere saber exactamente la posición del parking y el estado rápidamente, sin necesidad de saber el nombre del parking.

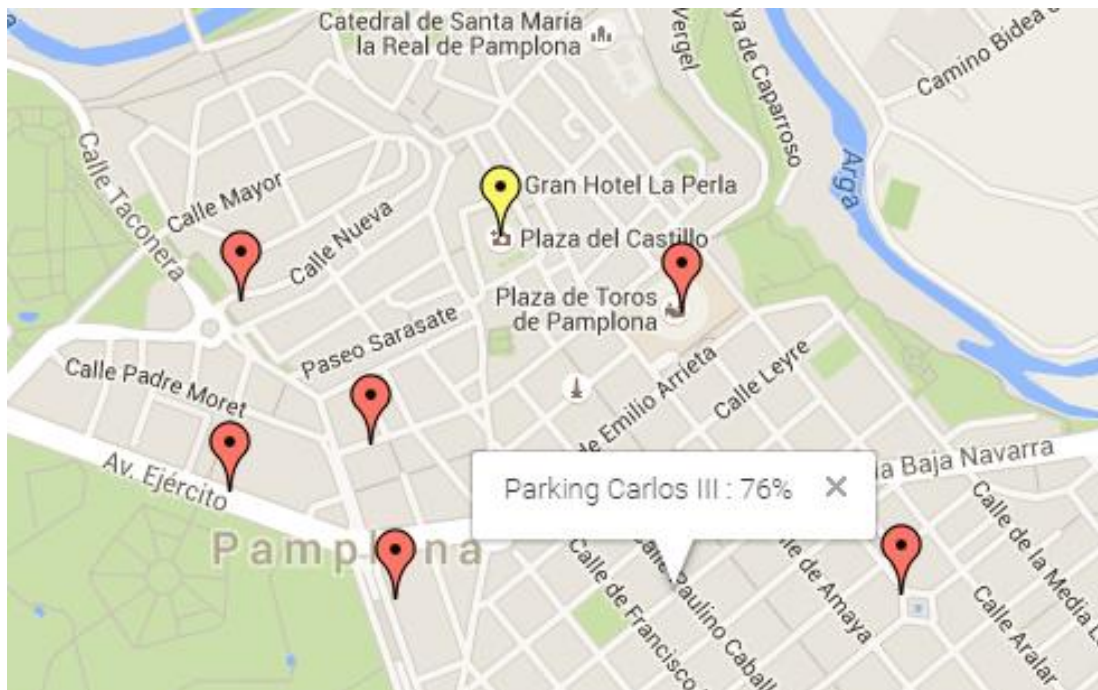


Ilustración 99: Comparativa (2)

4.3 Conclusiones

Con estas comparativas se pueden obtener las siguientes conclusiones:

- El desarrollo con Django, sin haber realizado nada anteriormente hace la curva de aprendizaje del framework sea muy fuerte al principio, pero una vez que vas controlando el lenguaje de Python, es realmente potente y se programa con sencillez debido a su sintaxis, como por ejemplo la ausencia de llaves ({ }). Sin embargo, eso hace que haya que tabular cuidadosamente el código.
- El desarrollo con NodeJS es complejo, debido a la asincronía que ofrece a la hora de tratar con tareas asíncronas como puede ser peticiones HTTP externas o el trabajo con ficheros.
- El desarrollo con la biblioteca de Angular de google Maps, hace sencillo que se trabaje con la información, debido al doble binding, pero es más complejo el realizar una interfaz bonita
- El uso de google Maps y los markers sin utilizar librerías externas para Polymer (solo se encarga de la carga del mapa), hace que sea más sencillo el representar

la información, pero es más complicado el trabajar con los arrays de información.

- La gestión del login social es mucho más sencilla en Django, ya que se integra con el sistema de gestión de usuarios propio del framework, sin embargo, está poco documentado.
- La gestión del login social es más compleja con Passport para NodeJS, sin embargo, está mucho mejor documentada que su homóloga en Django.
- El uso de Polymer es bastante similar al uso de AngularMaterial.

4.4 Líneas de trabajo futuras

Al realizar ambas aplicaciones la misma funcionalidad básica no se ha querido realizar más funcionalidades que podrían ser interesantes, ya que no es la finalidad propia de este trabajo, que está más centrado en la comparación entre los frameworks utilizados.

Algunas funcionalidades curiosas que se podrían son las siguientes:

- Una gráfica de la evolución de los aparcamientos a lo largo del día. Lo que llevaría a realizar peticiones al proveedor de los datos cada periodo de refresco de la información para poder almacenarla y tratarla.
- Añadir más ciudades a la aplicación. Esta nueva funcionalidad, depende más que de la programación y desarrollo propio, el poder obtener la información de “Open Data” de fuentes de otras ciudades. Se podría seleccionar la ciudad a través de un panel lateral.

5 REFERENCIAS BIBLIOGRÁFICAS Y RECURSOS ELECTRÓNICOS

- [1] IniciativaAporta. (s.f.). Obtenido de <http://datos.gob.es/acerca-de>
- [2] Torralbo, J. A. (s.f.). Tendencias en integración de bases de datos. En J. A. Torralbo, Tendencias en integración de bases de datos (págs. 168-171). Ediciones CEF.
- [3] opendatahandbook. (s.f.). Obtenido de <http://opendatahandbook.org/guide/es/what-is-open-data/>
- [4] JSON wiki. (s.f.). Obtenido de <https://es.wikipedia.org/wiki/JSON>
- [5] Zenika. (s.f.). Obtenido de <http://blog.zenika.com/2011/04/10/nodejs/>
- [6] algo3.uqbar-project. (s.f.). Obtenido de <http://algo3.uqbar-project.org/material/herramientas/angular/angularjs---resumen-de-la-arquitectura>
- [7] opendatahandbook. (s.f.). Obtenido de <http://opendatahandbook.org/guide/es/what-is-open-data/>
- [8] Obtenido de <https://www.mongodb.com/compare/mongodb-mysql>
- [9] Comparativa Mongo. (s.f.). Obtenido de <http://blog.eltallerweb.com/wp-content/uploads/2013/02/TablaComparativaMongo.jpg>
- [10] worldmaker. (s.f.). Obtenido de <http://blog.worldmaker.net/2006/feb/09/the-django-framework-architecture/>
- [11] chattyhive. (s.f.). Obtenido de https://blog.chattyhive.com/wp-content/uploads/2014/01/Django_mvc.png
- [12] platzi. (s.f.). Obtenido de <https://platzi.com/blog/web-components-polymer>
- [13] wikipedia. (s.f.). Obtenido de <https://en.wikipedia.org/wiki/Polyfill>
- [14] i.blogs. (s.f.). Obtenido de http://i.blogs.es/8a524d/web_components/650_1200.png
- [15] django-bower. (s.f.). Obtenido de <https://django-bower.readthedocs.io/en/latest/>
- [16] nodejs. (s.f.). Obtenido de <https://nodejs.org/es/>

[17] NPM. (s.f.). Obtenido de [https://en.wikipedia.org/wiki/Npm_\(software\)](https://en.wikipedia.org/wiki/Npm_(software))

[18] npmjs. (s.f.). Obtenido de <https://www.npmjs.com/>

[19] expressjs. (s.f.). Obtenido de <http://expressjs.com/es/>

[20] Passport. (s.f.). Obtenido de <http://passportjs.org/>

[21] jade-lang. (s.f.). Obtenido de <http://jade-lang.com/>

[22] angularjs. (s.f.). Obtenido de <https://angularjs.org/>

[23] meanjs. (s.f.). Obtenido de <http://meanjs.org/>

[24] genbeta. (s.f.). Obtenido de
http://i.blogs.es/235d57/650_1000_mean_stack/original.png

[25] newspindigital. (s.f.). Obtenido de http://www.newspindigital.com/wp-content/uploads/2014/08/NSD_Node.js-Stack.png

[26] djangoproject. (s.f.). Obtenido de www.djangoproject.com

[27] polymer. (s.f.). Obtenido de www.polymer-project.org

6 AUTORIZACIÓN DE DIFUSIÓN

El/la abajo firmante, matriculado/a en el Máster Universitario en Arquitectura del Software, autorizará a la Universidad a Distancia de Madrid (UDIMA) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el Trabajo Fin de Máster: “Desarrollo de componentes web basados en Angular JS, Polymer, Node.js, Django y MongoDB para el pre-procesado, consumo y homogeneización de fuentes Open Data gubernamentales”, realizado durante el curso académico 2015-2016 bajo la dirección de David Lizcano Casas en el Departamento de Ingeniería en Informática y Organización Industrial y a la Biblioteca de la UDIMA a depositarlo en el Archivo Institucional con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.