

Desarrollo Web con Angular 5 (front) y API REST con NodeJS (back)

Desarrollo Web con Angular 5 (front) y API REST con NodeJS (back)



Angular 5 + Node

MI1519

Introducción

El objetivo del curso es presentar a los alumnos las características de desarrollo Web utilizando Angular 5 como biblioteca JavaScript para la parte front y la creación de API REST con NodeJS para la parte back. Durante el curso se verá la integración de ambas partes y las mejores prácticas de desarrollo y puesta en marcha.

Objetivos

Al finalizar este curso los alumnos podrán:

- Crear componentes web con Angular 5
- Crear API REST con NodeJS
- Consumir API desde Angular 5
- Probar aplicaciones

Dirigido a

- Desarrolladores

Requisitos

- Ninguno

Profesorado

Contamos con un equipo de instructores altamente cualificados que combinan la actividad formativa con el desarrollo de su actividad profesional como expertos en el campo de las TIC. Profesionales certificados por los principales fabricantes del sector capaces de transferir de forma amena y entendedora los conceptos técnicos más abstractos.

Metodología

Curso presencial, activo y participativo. El docente introducirá los contenidos haciendo uso del método demostrativo, los participantes asimilarán los conocimientos mediante las prácticas de aplicación real.

Barcelona Carrer Almogàvers 123 / T. +34 933 041 720 / F. +34 933 041 722 / 08018 Barcelona
Madrid Plaza de Carlos Trías Bertrán 7, 1^a Planta (Edificio Sollube) / T. +34 914 427 703 / 28020 Madrid
Bilbao San Vicente 8, 6^a Planta (Edificio Albia I) / T. +34 944 354 982 / 48001 Bilbao

Documentación

Cada alumno recibirá un ejemplar de la Documentación acorde a los contenidos del curso.

Contenidos

1. Herramientas

- 1.1. Visual Studio Code
- 1.2. Herramientas para proyectos:
 - 1.2.1. Node
 - 1.2.2. NPM
 - 1.2.3. Protractor
 - 1.2.4. Grunt
 - 1.2.5. Gulp
 - 1.2.6. Bower

2. Angular 5 – Front

- 2.1. Estructura de una aplicación en Angular 5
- 2.2. Angular-cli
- 2.3. Directivas y plantillas
- 2.4. Componentes y servicios
- 2.5. Navegación y enrutamiento
- 2.6. Enlace a datos y Pipes
- 2.7. Componentes anidados
- 2.8. Emisión y captura de eventos
- 2.9. HTTP

3. Angular Material

- 3.1. Características
- 3.2. Componentes
- 3.3. Instalación y uso

4. NodeJS – Back

- 4.1. Primeros pasos con NodeJS
- 4.2. Instalación y uso de express
- 4.3. Plantillas
- 4.4. Servicios REST y protocolo http
- 4.5. Web Sockets y soluciones con socket.IO

5. Testing

- 5.1. Pruebas JavaScript con Protractor
- 5.2. Pruebas JavaScript con Chai y Mocha

Duración del curso

25 horas distribuidas en 4 sesiones de 6,25 horas.

Apuntes

Desarrollo Web con Angular 5 (front) y API REST con NodeJS (back)

Índice

- 1. Herramientas**
- 2. Angular 5 – Front
- 3. Angular Material
- 4. NodeJS – Back
- 5. Testing



Herramientas

Visual Studio Code

Herramientas para proyectos

Node

NPM

Protractor

Grunt

Gulp

Bower

NotePad++

➤ Lo que necesitamos:

- Herramientas básicas de edición de código
- Monitorizar y comprobar el funcionamiento (depuración en editores y en navegadores)

➤ Herramientas para la edición de código (todas gratuitas)

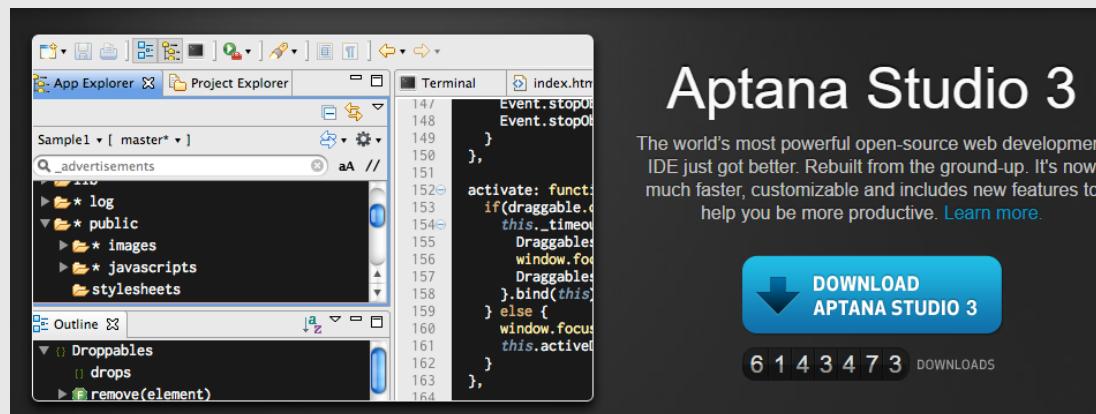
- **NotePad++**: Es gratuito y Open Source (Windows/Versión 7.4.1)
 - Sintaxis en colores (definibles)
 - Estructuración de código (con despliegue opcional de zonas)
 - Soporte de otros lenguajes de programación, etc...
 - Disponible en <https://notepad-plus-plus.org/download/v7.5.8.html/>



Aptana Studio

➤ Aptana Studio (versión 3.6.1- Windows y otras plataformas)

- Versión "light" de **Eclipse**
- Soporte de los lenguajes principales de la Web.
- Montones de complementos y posibilidad de manejar proyectos (no solo edición de archivos)
- Depuración incluida
- <http://www.aptana.com/>



Brackets

Brackets

- Actualmente en la versión 1.13 (<http://brackets.io/>)
- Especialmente pensado para páginas Web. (*Open Source*)
- Sincroniza el código con la vista en el navegador (vista previa dinámica)
- Versión en castellano
- Utiliza el motor V8 de Chrome (y permite editar directamente para Node.js)



The screenshot shows the Brackets IDE interface. On the left, the 'Working Files' sidebar lists files such as index.html, brackets.io (with a dropdown menu), gitignore, 404.html, blog.html, CNAME, contribute.html, css, docs, favicon.ico, favicon.png, and humans.txt. The main workspace shows the content of index.html:

```
103 <div id="hero-wrapper">
104   <div id="hero" class="row">
105     <div class="large-12 columns">
106       <h1>Brackets is an open source code editor for web designers and front-end developers. <a class="hero-cta-button" href="https://github.com/adobe/brackets/releases/latest" class="large radius">Get Started</a>
107     <div id="download">
108       <a id="hero-cta-button" href="https://github.com/adobe/brackets/releases/latest" class="large radius">Latest Version</a>
109     </div>
110   </div>
111 </div>
112 <div id="os-alert" class="alert-box radius" data-alert>
113   </div>
114 </div>
```

Below the code editor, a status bar indicates 'index.html — Brackets'.

Sublime Text

➤ Sublime Text:

- Es multiplataforma y una sola licencia sirve para cualquier S.O. en que lo usemos (Windows, OS X y Linux- Version 3 Build 3176).
- Dispone de una extensión de este editor especialmente pensada para Angular. Montones de "plug-ins" adicionales.
- Dispone de versiones instalables en "pen-drive" (32/64 bits)
- Sitio: <https://www.sublimetext.com/> (No hay depuración)



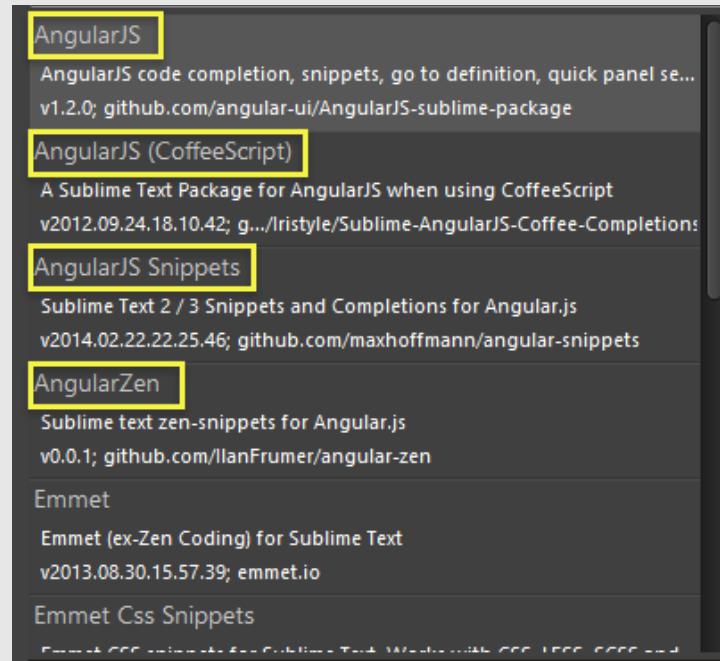
Sublime Text

➤ Instalación de *Sublime Text*

- Dos versiones: 2.0 y 3.0 (32/64 bits)
 - Se distinguen por el número de "*Build*"
 - Evaluación gratuita sin limitación de funcionalidad ni tiempo.
- La única diferencia es que -después de grabar varias veces un archivo- muestra un pequeño mensaje en pantalla.
- Desde un principio, se dispone de la posibilidad de editar código en un montón de lenguajes: HTML, CSS, JavaScript, PHP, C, C++, C#, Java, Objective-C, etc.
- En el sitio Web Package Control (<https://sublime.wbond.net/>) dispone de docenas de extensiones para este editor.
- Igualmente, desde la opción "*Preferences/Package Control*" aparecerá una ventana especial donde se ofrecen diversas opciones para el manejo de paquetes

Sublime Text

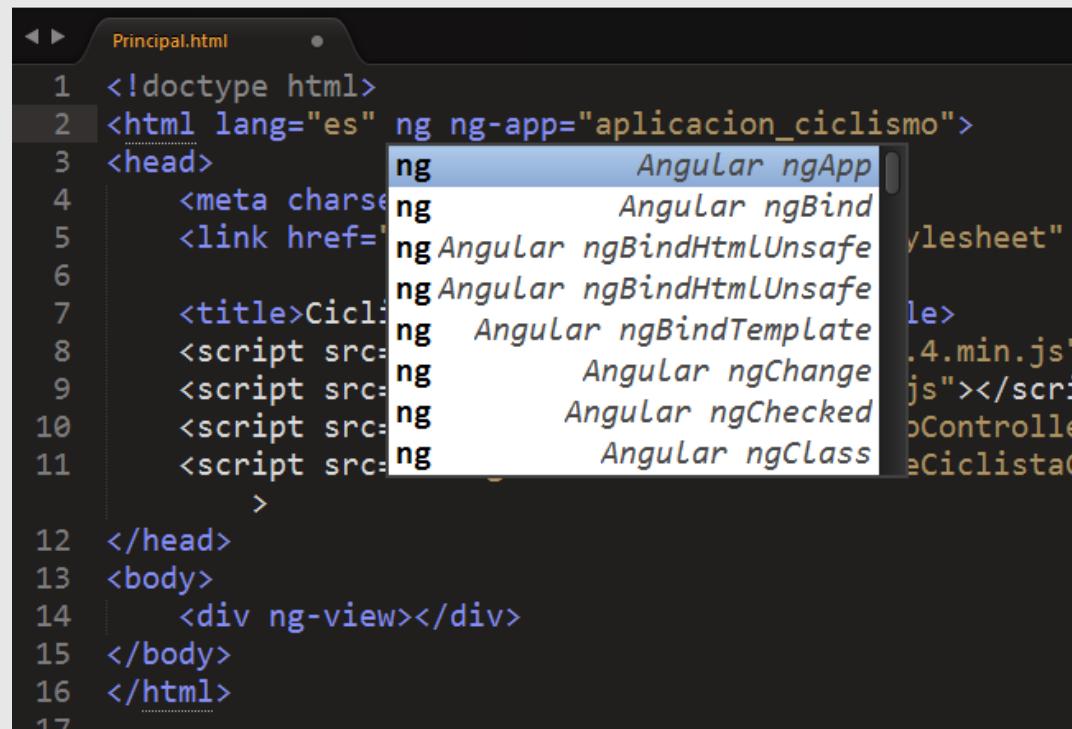
- Si seleccionamos la opción "*Install Package*", aparece un listado con la lista completa de paquetes disponibles.



- Con pulsar sobre cualquiera de ellas, se producirá la descarga e instalación inmediata (Normalmente, para que los cambios tengan efecto, tendrás que reiniciar el editor).

Sublime Text

- Si, una vez instalado, abrimos cualquiera de nuestros ejemplos anteriores, veremos algunas características inmediatamente.



A screenshot of the Sublime Text code editor showing a file named "Principal.html". The code is an HTML document with an Angular application setup. At line 2, there is a call to 'ng ng-app="aplicacion_ciclismo"'. A cursor is positioned at the start of the word 'ng' in this call. A dropdown menu is open, listing several Angular directives starting with 'ng', such as 'Angular ngApp', 'Angular ngBind', 'Angular ngBindHtmlUnsafe', 'Angular ngBindTemplate', 'Angular ngChange', 'Angular ngChecked', and 'Angular ngClass'. The background of the code editor is dark, and the code is written in white and light blue syntax highlighting.

```
<!doctype html>
<html lang="es" ng ng-app="aplicacion_ciclismo">
  <head>
    <meta charset="UTF-8">
    <link href="css/style.css" rel="stylesheet" type="text/css">
    <title>Ciclista</title>
    <script src="js/angular.min.js"></script>
    <script src="js/app.js"></script>
    <script src="js/controllers.js"></script>
    <script src="js/services.js"></script>
  </head>
  <body>
    <div ng-view></div>
  </body>
</html>
```

Sublime Text

- Package Control
- <https://packagecontrol.io/installation>
 - Emmet: http://docs.emmet.io/cheat-sheet/
 - **Angular JS**, Angular snippets, Angular material snippets, Angular Jasmine Boilerplate, etc.
- Aparte, interesa instalar NPM para configurar los directorios de trabajo con las herramientas directamente

A screenshot of the Sublime Text interface. The menu bar at the top includes 'Tools', 'Project', 'Preferences', and 'Help'. Below the menu is a tab bar with the active tab labeled 'AngularSublime1.html'. The main editor area shows the following code:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Demo</title>
```

A floating command palette is open in the bottom right corner, displaying the results of a search for 'npm'. The top result is 'npm' with a description 'npm commands within Sublime Text'. Below it is a link 'install v0.0.9; github.com/PixnBits/sublime-text-npm'.

Visual Studio Community Edition

- **Visual Studio 2017 Community Edition**
- Totalmente gratuito: para cualquier actividad y – en desarrollo profesional- para equipos de hasta 5 personas.
 - Idéntico a la versión **Professional**, pero con muchas opciones añadidas.
 - Soporta todas las opciones de desarrollo Web actuales: Apache/Cordova, Node.js, PHP, HTML5, CSS3, JavaScript 5 y 6, TypeScript, LESS, SASS, etc.
 - Es de muy rápida instalación
 - Gratuita



Visual Studio Community Edition

- Un paquete que instala un grupo de extensiones seleccionadas como las mejores para el desarrollo web
- Al igual que muchas otras extensiones, están disponibles en el sitio: <https://marketplace.visualstudio.com/>

- **Add New File**
- **Browser Reload on Save**
- **Browser Sync**
- **Glyphfriend**
- **Image Optimizer**
- **JavaScript Snippet Pack**
- **Open Command Line**
- **Package Installer**
- **Suggested Extensions**
- **Web Accessibility Checker (*)**
- **Web Analyzer**
- **Web Compiler**
- **Web Essentials**
- **...etc.**



Visual Studio Code

- **Visual Studio Code** es el "compañero cross-platform" de Visual Studio
- Totalmente Open Source
 - <http://code.visualstudio.com>
- Funciona en cualquier plataforma
- Personalizada a los más importantes idiomas (Español entre ellos).
- Linux, Mac y Windows
- Está basado en los motores de Javascript de:
 - NodeJS
 - Electrón
- Permite distintos niveles de soporte de código fuente: Básico, Mejorado y Óptimo
- **Es la que está usando el propio equipo de desarrollo de Angular en Google.**

Herramientas

editor

- Ligero y rápido
- Ficheros y carpetas
- Varios lenguajes
- Varios Workflows
- Basado en teclado



IDE

- Proyectos
- Comprensión del código
- Depuración
- "Build" integrado
- "File>New", Asistentes
- Diseñadores
- Integración ALM
- Herramientas de plataforma
- ...

- Ligero y rápido
- Ficheros y carpetas con contexto de proyecto
- Varios lenguajes
- Basado en teclado
- Comprensión del código (algunos lenguajes)
- Depuración
- Ejecución de tareas

Visual Studio Code

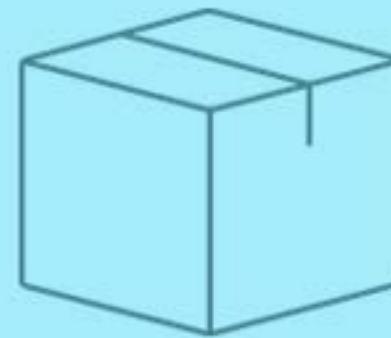
Runtimes	Node.js, ASP.Net 5
Lenguajes - básico* coloring, brackets, indent	CoffeeScript, Python, Ruby, Jade, Clojure, Java, C++, R, Go, makefile, shell, powershell, bat, XML, Markdown, ...
Lenguajes - mejorado + IntelliSense, linting, outline	JavaScript, HTML, CSS, LESS, SASS, JSON
Lenguajes - Óptimo + buscar referencias, refactoring, etc.	C#, TypeScript
source control	git
task running	gulp, grunt ...

Visual Studio Code



Web Tech

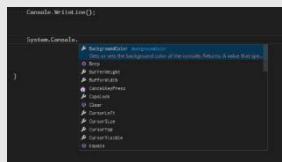
Use HTML, CSS, and JavaScript with Chromium and Node.js to build your app.



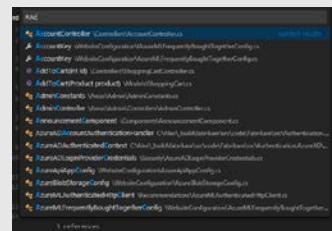
Cross Platform

Electron apps build and run on Mac, Windows, and Linux.

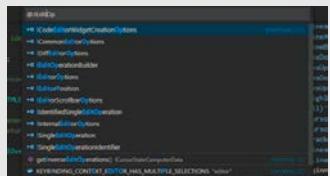
Visual Studio Code



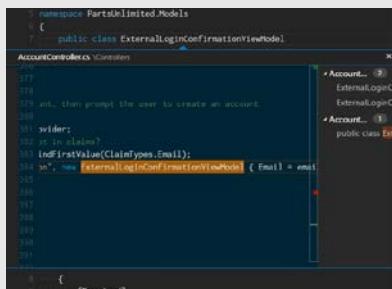
IntelliSense



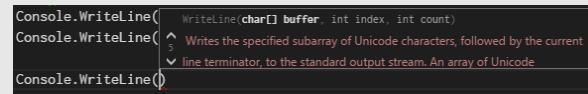
goto any symbol



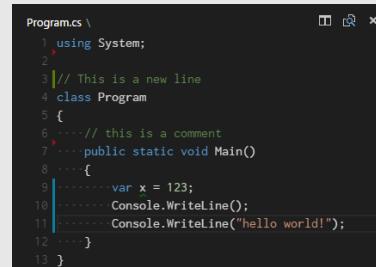
goto symbol in file



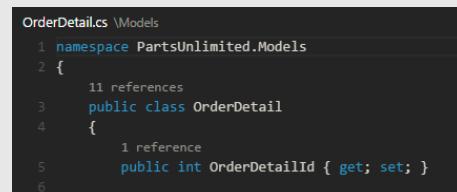
find all references



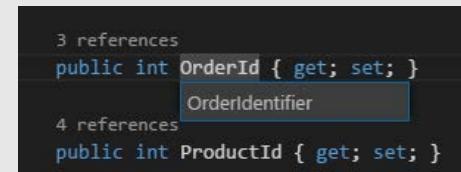
parameter hints



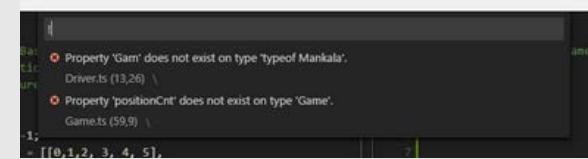
code actions



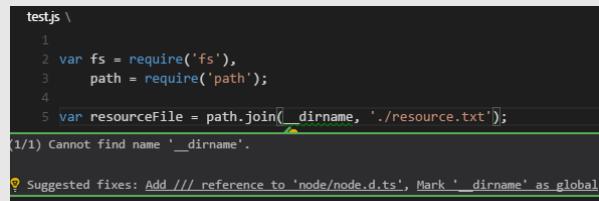
inline references



rename symbol



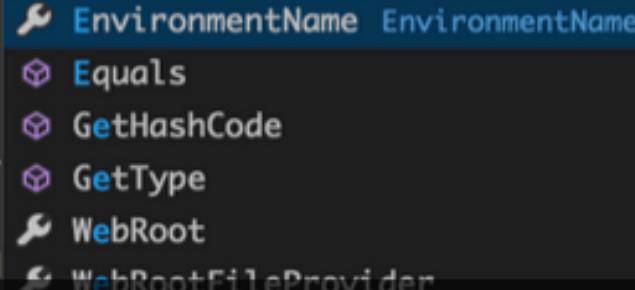
error/warnings



inline errors

Intellisense

```
0 references
public Startup(IHostingEnvironment env)
{
    // Setup configuration sources.
    Configuration = new Configuration()
        .AddJsonFile("config.json")
        .AddEnvironmentVariables();
    env.e
}
3 references
public ICon
    // This met
0 references
```



INTELLIGENT CODE AUTHORING, UNDERSTANDING AND NAVIGATION

Rich, multi-file IntelliSense for ASP.NET v5 application development using C#, JavaScript, TypeScript, HTML, CSS, LESS, and more. Easily navigate and understand your code with features traditionally only found only in IDEs, such as Outlining, Find All References, and Go To Definition.

```
// Add EF services to the services container if not using Mono
// consider using Azure SQL or non local store for xplat until EF7 has
```

Depuración

The screenshot shows a Node.js debugging session in the Netmind IDE. The left sidebar has sections for VARIABLES (Local), CALL STACK, and BREAKPOINTS. The VARIABLES section shows local variables like __dirname and __filename. The CALL STACK shows the stack trace from module compilation down to the app.use call. The BREAKPOINTS section shows two breakpoints set at app.js:10 and app.js:18. The right pane displays the app.js code with these breakpoints marked by red dots. The code sets up an Express app, configures ports, views, engines, and routes.

```
app.js /  
10 var app = express();  
11 // all environments  
12 app.set('port', process.env.PORT || 3000);  
13 app.set('views', path.join(__dirname, 'views'));  
14 app.set('view engine', 'jade');  
15 app.use(express.favicon());  
16 app.use(express.logger('dev'));  
17 app.use(express.json());  
18 app.use(express.urlencoded());  
19 app.use(express.methodOverride());  
20 app.use(app.router);  
21 app.use(express.static(path.join(__dirname,  
22 "/User...  
23 // development only  
24 if ('development' == app.get('env')) {  
25   app.use(express.errorHandler());  
26 }  
27 }  
28  
29 app.get('/', routes.index);  
30 app.get('/users', user.list);
```

Control de código fuente (Git)

The screenshot shows the GitHub desktop application interface. On the left, there's a sidebar with icons for file browser, search, and notifications (9). The main area has a 'GIT' header with a checkmark, refresh, and three-dot menu buttons. A message box says 'Moving/renaming site assets'. Below it, the 'STAGED CHANGES' section shows four items: 'M bower.json /', 'A editcloud9.png /site-assets2', 'R bower.json /template2 ← bower...', and 'R readme_template.md /template...'. The number '4' is in a circle next to the section title. At the bottom, there's a 'CHANGES' section with a number '5' in a circle and a 'SIMPLIFIED GIT SOURCE CONTROL' section with a 'M knownIssues.is /browser-tests' item. The right side of the screen is a code editor for 'base.css' in the '/examples/typescript-ang' directory. The code is as follows:

```
base.css /examples/typescript-ang
1 html,
2 body {
3     margin: 0;
4     padding: 0;
5 }
6
7 button {
8     margin: 0;
9     padding: 0;
10    border: 0;
11    background: none;
12    font-size: 100%;
13    vertical-align: baseline;
14    font-family: inherit;
15    color: inherit;
```

Control de código fuente (OneDrive)

The screenshot shows the OneDrive web interface. In the top navigation bar, there's a cloud icon, the text "OneDrive |", and dropdown menus for "Rename", "Download", "Share", and "Manage". On the right, it shows "Alex Dima" and a profile icon. Below the bar, the path "greeter" is shown under "Alex's OneDrive > greeter.ts".

The main area contains a code editor with the following TypeScript code:

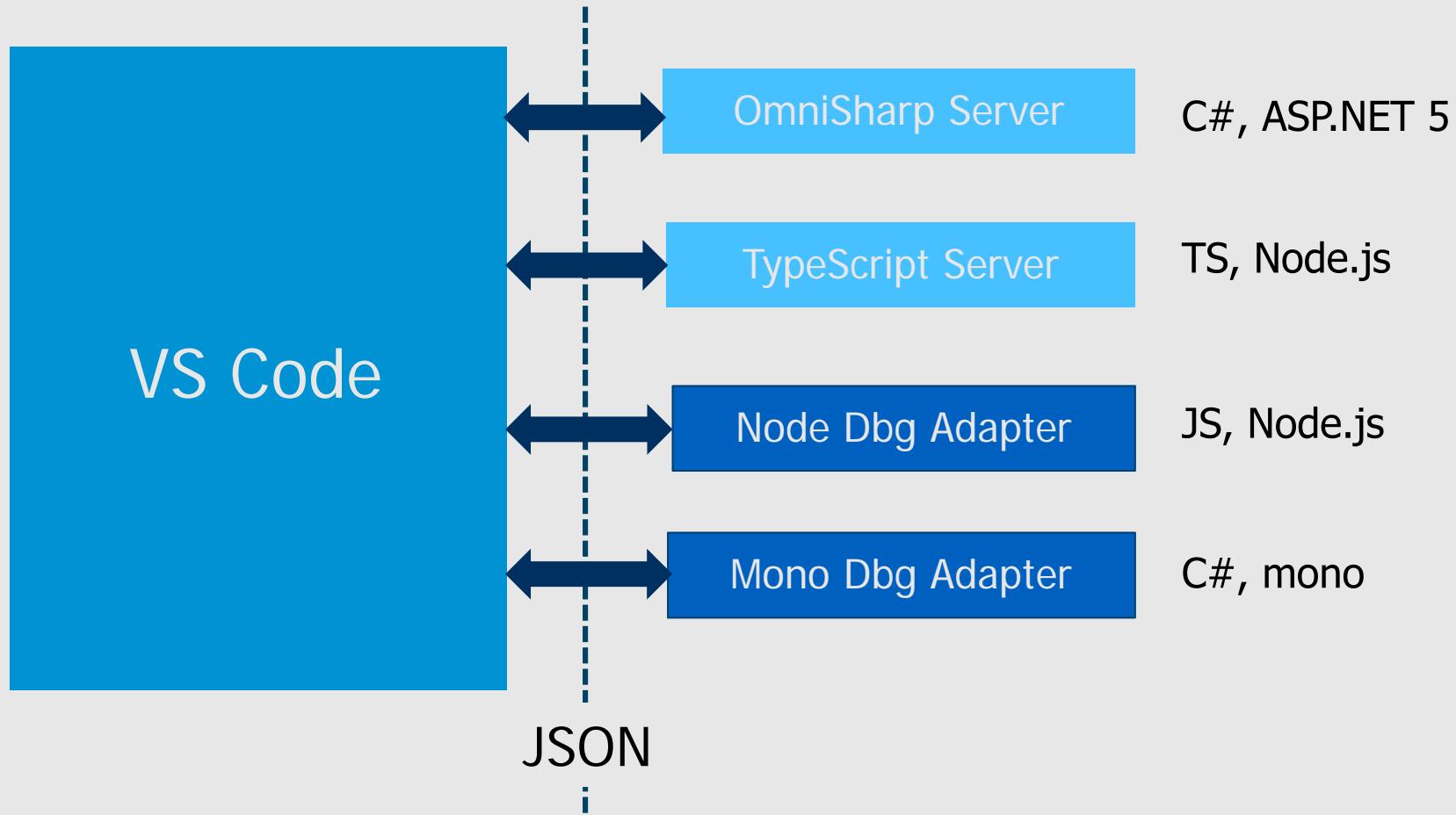
```
i class Greeter {  
  greeting: string;  
  constructor(message: string) {  
    this.greeting = message;  
  }  
  greet() {  
    return "Hello, " + this.greeting;  
  }  
}  
var greeter = new Greeter("world");  
  
var button = document.createElement('button');  
button.textContent = "Say Hello";  
button.onclick = function() {  
  alert(greeter.greet());  
}  
  
document.body.appendChild(button);
```

To the right of the code editor is a preview pane showing a white document with the text "greeter.ts". Below the preview, there's a "Save" button and a "Share" icon. A "Description" field is present with the placeholder "Add a description".

On the far right, there are two sections: "Sharing" and "Information". The "Sharing" section has a collapsed "Share" menu with the option "Only me". The "Information" section is expanded and displays the following details:

Type	Text Document
Modified	A moment ago Alex Dima
Added	A moment ago Alex Dima
Path	Alex's OneDrive > greeter.ts
Size	410 bytes

Arquitectura de los servicios



Visual Studio Code

Request	Response
definition	Returns the file location of the symbol at the given file location
references	Returns the file locations that reference the symbol at the given file location
completions	Returns possible completions at a given file location
symbolDetails	Return a type information and documentation for the symbol at a given file location
format	Returns edit instructions to format a file
outline	Returns list of navigation targets for a particular file
open	Notifies the server that the editor host has opened a file
close	Notifies the server that a previously opened file is now closed
change	Notifies the server that a range of a file has changed

Servicio *Language Worker*

Servicio *Debugger Adapter*

Request	
launch	Launch a debuggee
setBreakpoints	Sets multiple break points
continue, next, stepIn, stepOut	Execution control
stackTrace	Returns the stack trace from the current execution state
variables	Returns the children of all variables given a variable reference
Event	
stopped	The execution of the debuggee has stopped
exited	The debuggee has terminated

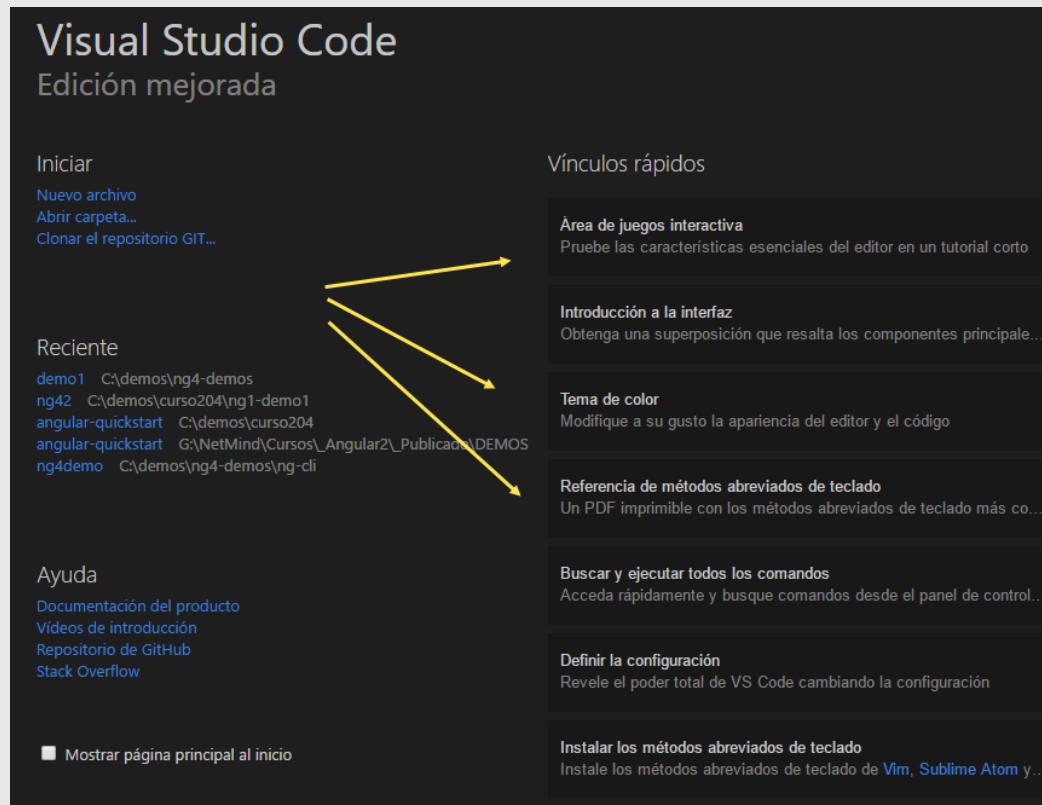
Personalización

- Todas las acciones y comandos son configurables
 - Se usan ficheros JSON para establecer la configuración
 - Para personalizarlo, sobrescribimos el fichero predeterminado (**read-only**) con uno propio.
 - Vale para configuraciones de trabajo y atajos de teclado (ficheros distintos)
 - También disponemos de un listado extenso de temas (themes) para personalizar el aspecto visual.
 - Finalmente, las extensiones llegan a superar el millar (de todo tipo)

```
{} keybindings.json x
1 // Coloque sus enlaces de teclado en este archivo
2 [
3   {
4     "key": "shift+alt+down",
5     "command": "cursorColumnSelectDown",
6     "when": "editorTextFocus"
7   },
8   {
9     "key": "shift+alt+left",
10    "command": "cursorColumnSelectLeft",
11    "when": "editorTextFocus"
12  },
13 ]
```

Introducción "in-line": "Bienvenidos"

- Las últimas versiones, incluyen una "mini-aplicación" que explica el funcionamiento y los cambios más importantes: "Ayuda/Bienvenidos"



Introducción "in-line": "Bienvenidos"

- Cada entrada de la izquierda, despliega una nueva solapa con explicaciones, algunas de ellas, con Intellisense incorporado

The screenshot shows a code editor interface with two floating toolbars. The top toolbar is titled 'Errors and Warnings' and displays a tooltip for the 'Number' type, stating: '[js] 'types' can only be used in a .ts file.' Below this, the code editor shows a snippet of JavaScript:

```
1 // This code has
2 Console.log(add(1));
3
4
5 function Add(a : Number, b : Number) : Int {
6     return a + b;
7 }
```

The bottom toolbar is titled 'Rename Refactoring' and provides instructions for renaming symbols. Below it, another code editor window shows a snippet of TypeScript:

```
1 // Reference the function
2 new Book("War of the Worlds", "H G Wells");
3 new Book("The Martian", "Andy Weir");
4
5 /**
6  * Represents a book.
7 */
8 function Book(title, author) {
9     this.title = title;
10    this.author = author;
11 }
```

Visual Studio Code

F1: paleta de comandos
Configuración
Personalización
Terminal

Herramientas – Otras opciones

➤ *Fiddler*

- Herramienta "Open Source" desarrollada por *Eric Lawrence*
- Existen dos versiones de Fiddler: Fiddler 2 (COM) y Fiddler 4, realizada en .NET 4.0
- Se trata de un "*sniffer*" de tráfico de internet, capaz de monitorizar cualquier tipo de tráfico
- Registro de todo el tráfico de tipo HTTP/HTTPS
- Gestión y control de la sesión
- Depuración Web (con un nivel de configuración que le hacen una herramienta única)
- Pruebas de seguridad
- Pruebas de rendimiento: Permite simular pruebas de carga real configurando el número de usuarios y obteniendo resultados tanto en formato estadístico, como gráfico.

Herramientas – Otras opciones

The screenshot shows the Fiddler Web Debugger interface. The main pane displays a list of network requests and responses. The selected request is for `/browserconfig.xml` from `localhost`, which resulted in a `404 Not Found` error. The right-hand panel provides detailed information about this request, including the Request Headers (Client and Transport), the raw response body (an HTML error page from IIS 10.0), and various tabs for viewing the response in different formats (Raw, JSON, XML, etc.).

Request Headers (Client):

- Accept: */*
- Accept-Encoding: gzip, deflate
- User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0

Request Headers (Transport):

- Connection: Keep-Alive
- Host: localhost

Raw Response Body:

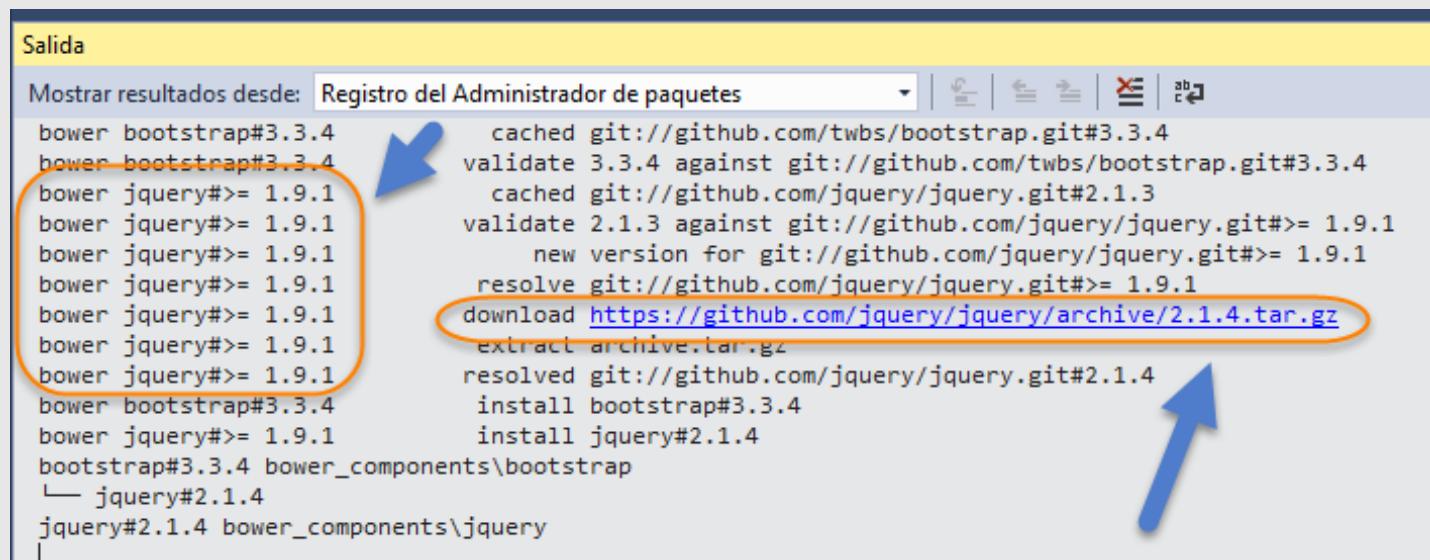
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/1999/xhtml">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Error detallado de IIS 10.0 - 404.0 - Not Found</title>
<style type="text/css">
<!--
body{margin:0;font-size:.7em;font-family:Verdana,Arial,Helvetica,sans-serif;}
code{margin:0;color:#006600;font-size:1.1em;font-weight:bold;}
.config_source code{font-size:.8em;color:#000000;}
pre{margin:0;font-size:1.4em;word-wrap:break-word;
    padding:0 10px 0 10px; font-family: monospace; border: 1px solid black; border-radius: 5px; background-color: #f9f9f9; }
-->
</style>
<body>
<h1>404.0 - Not Found</h1>
<p>The resource you are looking for was not found on this server. If you entered the URL manually, please check your spelling and try again or contact the administrator if you believe this is a mistake.</p>
<hr>
<small>Microsoft Internet Information Services 10.0</small>
</body>
</html>
```

Nuevas herramientas

Nombre	Función	Observaciones
Node.js	Servidor Web, multiplataforma, no-bloqueante, de instalación ligera.	Soporta un motor de JavaScript interno para su configuración
NPM	Manejador de paquetes para Node. Se especializa en instalación de <u>Herramientas</u> .	Compiladores LESS, SASS, TypeScript, CoffeeScript, Bundling & Minifying, ...
NuGet	Instalador de librerías externas en aplicaciones cliente: incluye DLL (.NET), y librerías (JavaScript, CSS, etc.).	Sólo DLL para .NET
Bower	Similar a NuGet, especializado en instalación de paquetes "script" en cliente. El acceso pasa a ser nativo en V. Studio 2015.	Pasa a jugar el papel de NuGet en lo referente a bibliotecas de "scripts". Dispone de muchas más bibliotecas que NuGet
Grunt	Ejecución de tareas en aplicaciones cliente. Se instalan vía NPM y por aplicación (dif. Versiones)	
Gulp	Competidor del anterior. Realiza tareas como compilación, B&M	
Git	Repositorio de código fuente	

Bower en acción

- Tras la grabación, Bower nos ofrece un “histórico” de los procesos que han tenido lugar como consecuencia de la grabación
- ...que incluyen las dependencias de otras bibliotecas y las descargas adicionales

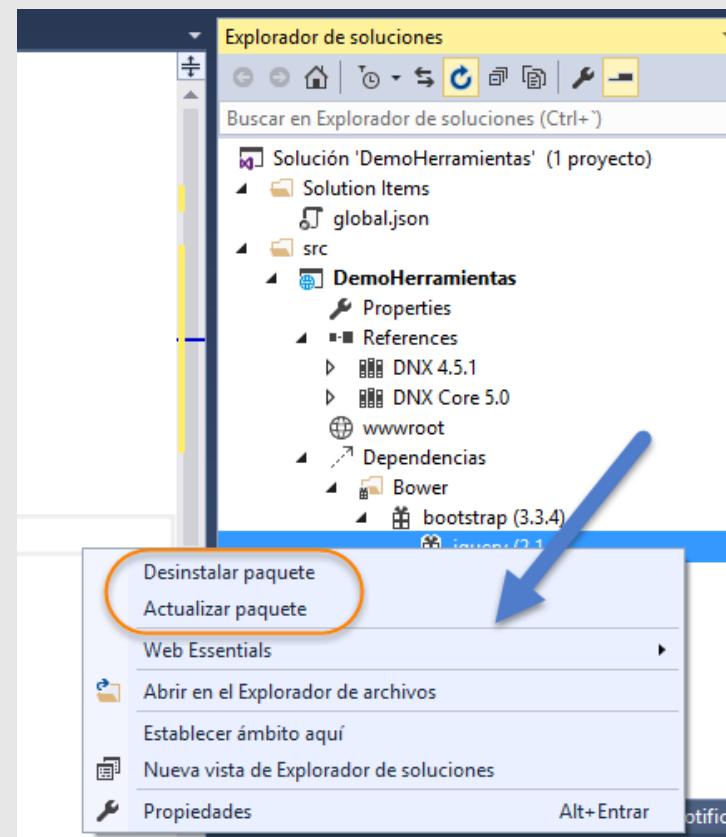


```
Salida
Mostrar resultados desde: Registro del Administrador de paquetes
bower bootstrap#3.3.4
bower bootstrap#3.3.4
bower jquery#>= 1.9.1
bower bootstrap#3.3.4
bower jquery#>= 1.9.1
bootstrap#3.3.4 bower_components\bootstrap
└── jquery#2.1.4
jquery#2.1.4 bower_components\jquery
cached git://github.com/twbs/bootstrap.git#3.3.4
validate 3.3.4 against git://github.com/twbs/bootstrap.git#3.3.4
cached git://github.com/jquery/jquery.git#2.1.3
validate 2.1.3 against git://github.com/jquery/jquery.git#>= 1.9.1
new version for git://github.com/jquery/jquery.git#>= 1.9.1
resolve git://github.com/jquery/jquery.git#>= 1.9.1
download https://github.com/jquery/jquery/archive/2.1.4.tar.gz
extract archive.tar.gz
resolved git://github.com/jquery/jquery.git#2.1.4
install bootstrap#3.3.4
install jquery#2.1.4
```

El Explorador de Soluciones ofrece menús inteligentes

- ...como hemos instalado un paquete nos sugiere actualizarlo o desinstalarlo entre otras opciones.
- Y podemos ver lo instalado en la carpeta física de la solución

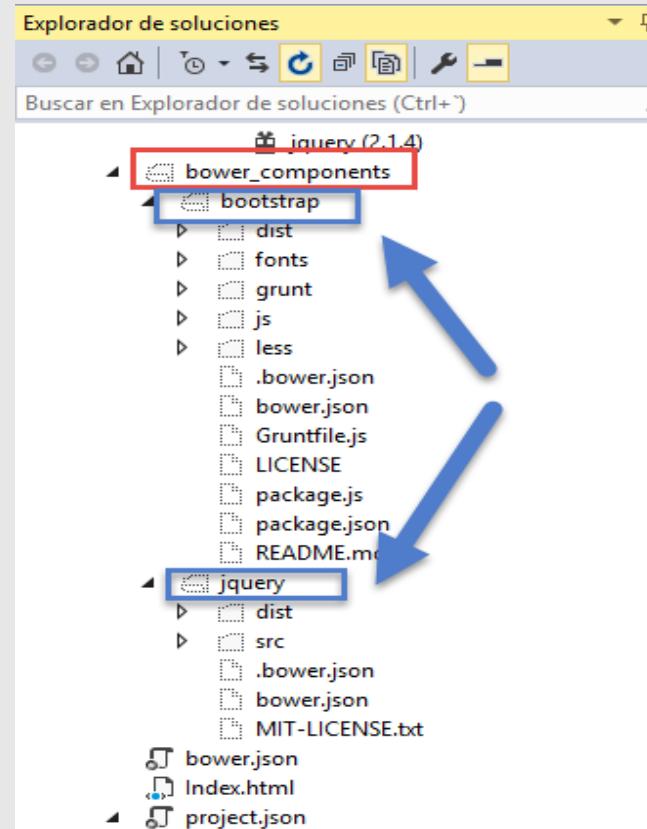
Carpeta		
Nombre	Fecha de modificación	Tipo
bootstrap	23/05/2015 19:55	Carpeta de archivos
jquery	23/05/2015 19:55	Carpeta de archivos



Lo instalado por Bower podemos examinarlo

- Desde el mismo Explorador de Soluciones, con la opción "Ver todos los archivos"

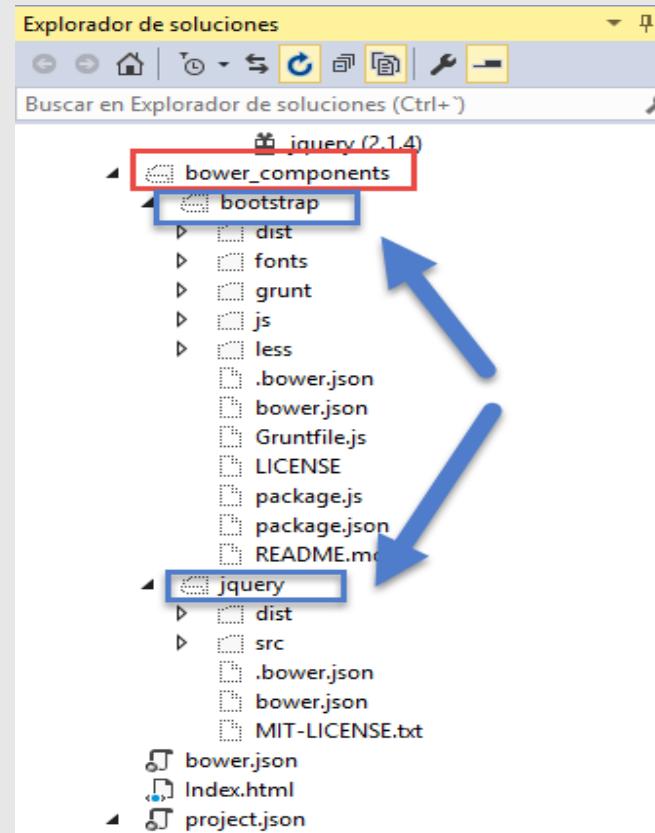
- Aparecerá en una carpeta individual de nombre predeterminado.



Gulp



- La labor de Gulp es establecer qué componentes instalados en nuestra aplicación deben de ser utilizados .

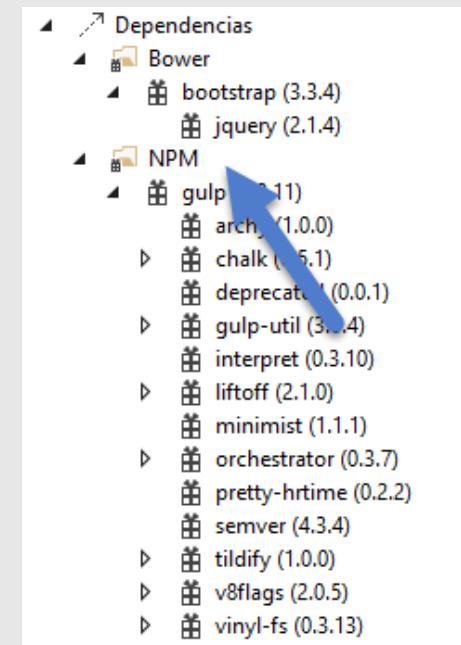


Gulp

- Para instalarlo, (junto a sus dependencias), lo declaramos en el fichero "**Package.json**".
 - Al grabar los cambios, igual que antes, se resolverán las dependencias y –como Gulp, depende de **NPM**- se instalará NPM igualmente.

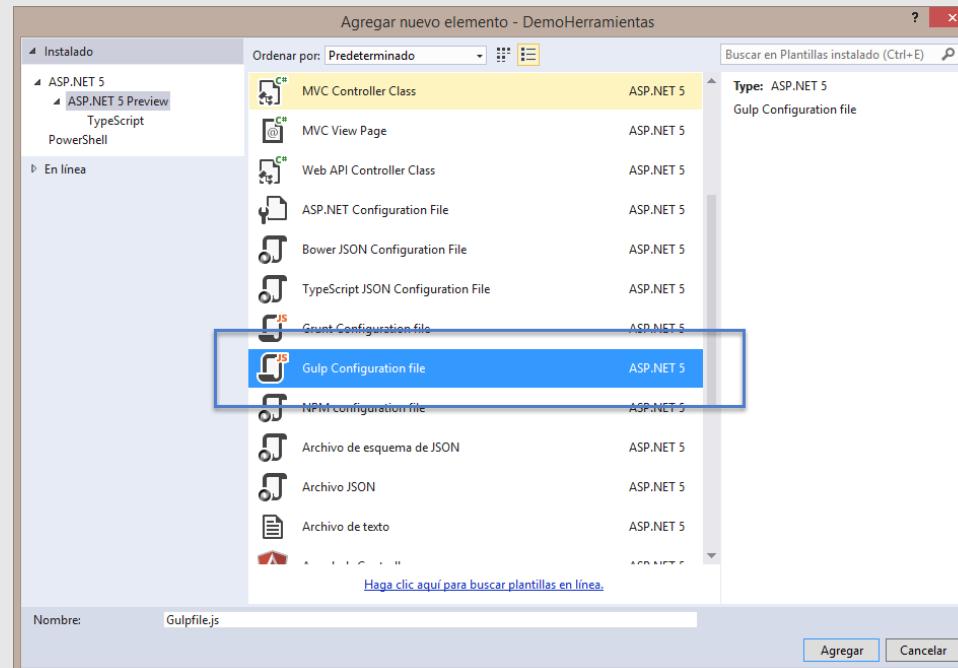
```
npm http 200 https://registry.npmjs.org/sigmund/-/sigmund-1.0.1.tgz
npm http 200 https://registry.npmjs.org/number-is-nan/-/number-is-nan-1.0.0.tgz
npm http 200 https://registry.npmjs.org/lru-cache/-/lru-cache-2.6.4.tgz
gulp@3.8.11 node_modules/gulp
├── fs.realpath@0.1.0
├── ty-hrtime@0.2.2
└── inherits@0.3.10
  ├── deprecated@0.0.1
  └── archy@1.0.0
  ├── minimist@1.1.1
  └── semver@4.3.4
  └── v8flags@2.0.5 (user-home@1.1.1)
  └── tildify@1.0.0 (user-home@1.1.1)
  └── chalk@0.5.1 (ansi-styles@1.1.0, escape-string-regexp@0.1.3, supports-color@0.2.0, strip-ansi@0.0.7)
  └── orchestrator@0.3.7 (stream-consume@0.1.0, sequencify@0.0.7, end-of-stream@0.1.5)
  └── liftoff@2.1.0 (extend@2.0.1, rechoir@0.6.1, flagged-respawn@0.3.1, resolve@1.1.6, findup-sync@0.2.9)
  └── gulp-util@3.0.4 (beeper@1.0.0, array-differ@1.0.0, array-uniq@1.0.2, object-assign@2.0.0, lodash@4.5.1)
  └── vinyl-fs@0.3.13 (graceful-fs@0.0.7, mkdirp@0.5.1, strip-bom@1.0.0, defaults@1.0.2, vinyl@0.4.6, es6-promise@3.0.2)

====npm command completed with exit code 0====
```



Gulp

- Ahora que disponemos de los servicios de Gulp, tenemos que configurarlo de manera similar
- Para ello creamos un fichero de nombre "**Gulpfile.js**"
- Aunque V. Studio nos lo ofrece como opción predeterminada...



Gulp

- Aquí configuramos las tareas necesarias para el desarrollo...
- También contamos con Intellisense de cualquier herramienta instalada

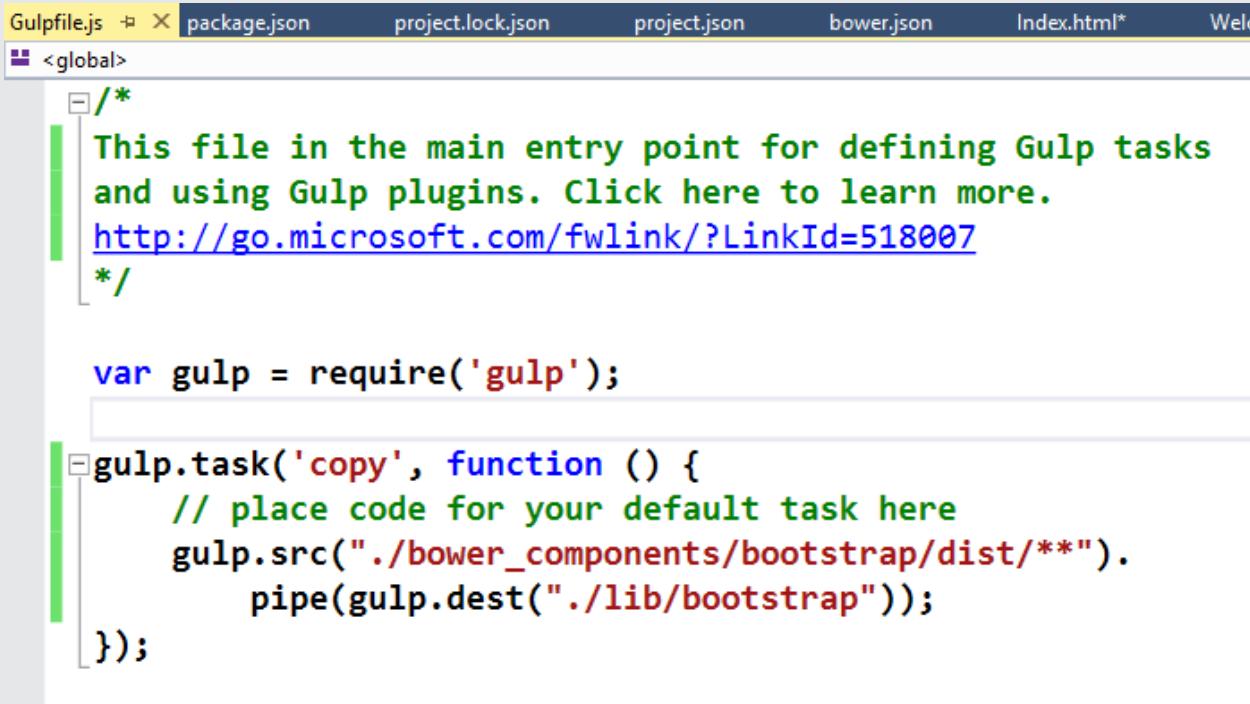
The screenshot shows an IDE interface with the following details:

- File Explorer:** Shows files like Gulpfile.js*, package.json, project.lock.json, project.json, bower.json, and Index.html*.
- Toolbar:** Includes "Welcome to ASP.NET 5" and a "gulp" button.
- Code Editor:** Displays the Gulpfile.js code:

```
/*  
 * This file is the main entry point for defining Gulp tasks  
 * and using Gulp plugins. Click here to learn more.  
 * http://go.microsoft.com/fwlink/?LinkId=518007  
 */  
  
var gulp = require('gulp');  
  
gulp.task('copy', function () {  
    // place code for your default task here  
    gulp.src()  
});
```
- Intellisense Pop-up:** A tooltip for the `gulp.src()` method is open, showing:
 - Signature:** `▲ 1 de 2 ▼ Stream src(String glob, [srcConfig options])`
 - Description:** Emits files matching provided glob or an array of globs. Returns a stream of Vinyl files that can be piped to plugins.
 - Parameter:** `glob: Glob or array of globs to read.`

Gulp

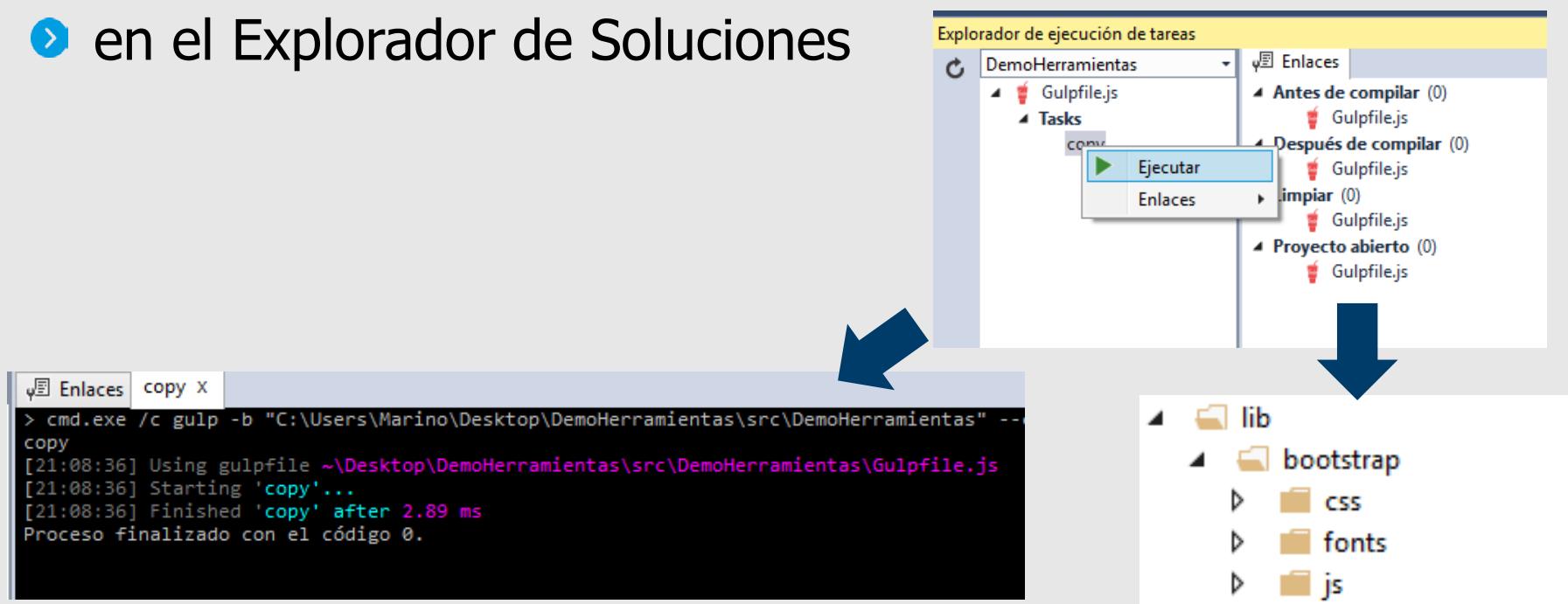
- Si, por ejemplo, indicamos a Gulp que extraiga un conjunto de archivos a una ubicación física de nuestro sitio, lo hacemos mediante llamadas (tipo "pipe") al objeto **gulp**:



```
Gulpfile.js ✘ package.json project.lock.json project.json bower.json Index.html* Welcome.html  
└─<global>  
  └─/*  
    This file is the main entry point for defining Gulp tasks  
    and using Gulp plugins. Click here to learn more.  
    http://go.microsoft.com/fwlink/?LinkId=518007  
  */  
  
  var gulp = require('gulp');  
  
  gulp.task('copy', function () {  
    // place code for your default task here  
    gulp.src("./bower_components/bootstrap/dist/**").  
        pipe(gulp.dest("./lib/bootstrap"));  
  });
```

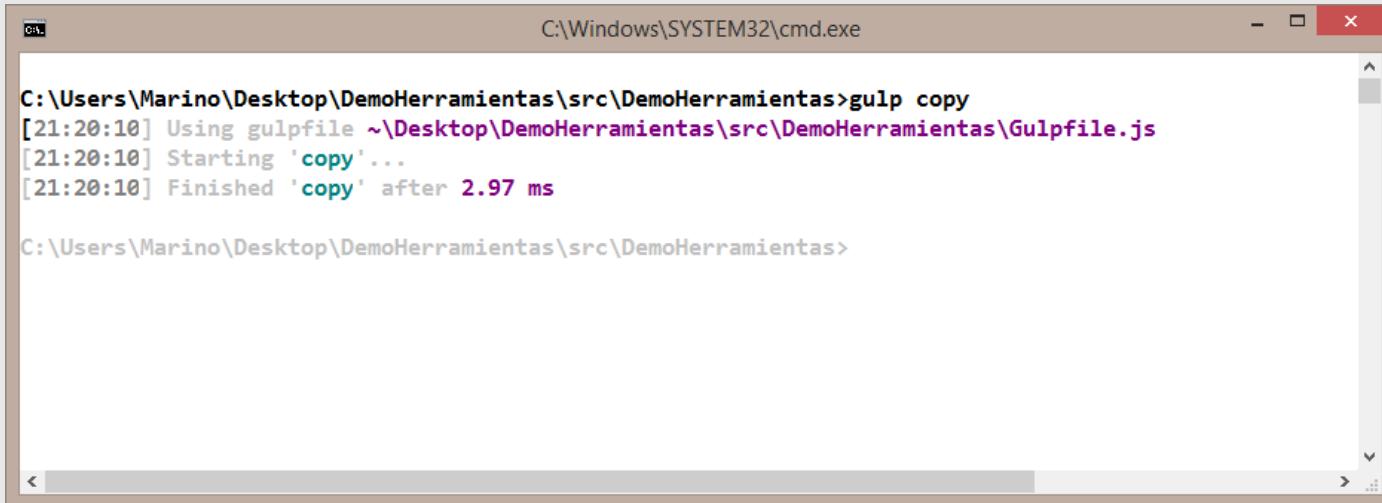
Gulp

- Automáticamente, aparece una nueva ventana, el "Explorador de ejecución de tareas", donde podemos ver lo configurado y seguir todos los procesos
- Y también, lanzar la ejecución de cualquier tarea definida, y ver cuál es el resultado de la ejecución
- en el Explorador de Soluciones



Gulp

- Visual Studio no realiza ninguna tarea especial, solamente lanza la ejecución de estos comandos.
- Podemos probarlo lanzando nosotros mismos la tarea desde una ventana de comandos.
- Existen complementos para instalarlo en Visual Studio 2013/2015/2017



A screenshot of a Windows Command Prompt window titled "C:\Windows\SYSTEM32\cmd.exe". The window contains the following text output:

```
C:\Users\Marino\Desktop\DemoHerramientas\src\DemoHerramientas>gulp copy
[21:20:10] Using gulpfile ~\Desktop\DemoHerramientas\src\DemoHerramientas\Gulpfile.js
[21:20:10] Starting 'copy'...
[21:20:10] Finished 'copy' after 2.97 ms

C:\Users\Marino\Desktop\DemoHerramientas\src\DemoHerramientas>
```

Gulp

- Mediante Gulp podemos "orquestar" nuestras aplicaciones.
- Es decir, podemos indicarle la secuencia exacta en que queremos que se trate cada parte de la aplicación y en qué forma debe hacerlo.
- Una secuencia puede ser, por ejemplo:
 - Buscar un fichero .LESS
 - Ejecutar una "transpilación" del fichero en otro de tipo CSS
 - Enviar esa salida a otra herramienta (por ejemplo, de minimización del código)
 - Colocar el resultado en una determinada ubicación de nuestro proyecto.

Grunt

- Es la primera herramienta popular que apareció para este tipo de tareas.
- Realiza básicamente las mismas acciones que Gulp
- Visual Studio 2017 soporta ambas indistintamente.



Referencias

URL	Título
https://github.com/DanTup/TestAdapters/blob/master/README.jasmine.md	Jasmine Test Adapter
http://www.emezeta.com/articulos/15-aplicaciones-para-montar-servidores-web-en-local	15 aplicaciones para montar servidores web en local

We Know IT

Barcelona

C. dels Almogàvers 123
08018 Barcelona
Tel. +34 933 041 720
Fax. +34 933 041 722

Madrid

Plaza Carlos Trías Bertrán,
7 Edificio Sollube, 1^a Planta
28020 Madrid
Tel. +34 914 427 703
Fax +34 914 427 707

Bilbao

C/ San Vicente, n^o 8
Edificio Albia I,
6^a planta - dpto. 8
48001 – Bilbao
Tel. +34 944 354 982

www.netmind.es

Desarrollo Web con Angular 5 (front) y API REST con NodeJS (back)

Índice

1. Herramientas
- 2. Angular 5 – Front**
 3. Angular Material
 4. NodeJS – Back
 5. Testing



Angular 5 - Front

Estructura de una aplicación Angular 5

Angular-cli

Directivas y plantillas

Componentes y servicios

Navegación y enrutamiento

Enlace a datos y pipes

Componentes anidados

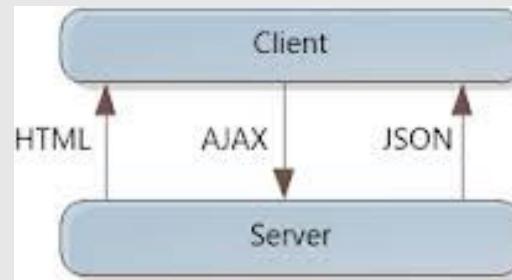
Emisión y captura de eventos

HTTP

El modelo actual de aplicaciones Web

➤ Nuevos supuestos para Aplicaciones Web

- Se tiende a asumir que el navegador es la nueva plataforma
- La nube permite acceso a contenidos desde no importa qué dispositivo
- HTML5 y sus tecnologías son la base de este "framework"
- El nuevo modelo de aplicaciones SPA (Single Page Applications), se adapta mejor a esta arquitectura
- Se utiliza un patrón de arquitectura tipo MVC, que facilita la separación de responsabilidades
- Los formatos de transferencia de información tienden a optimizarse, utilizando propuestas como JSON (fundamentalmente)
 - Los cambios en las páginas *responsive* se producen principalmente sobre la información modificada (peticiones AJAX).



El Modelo actual de aplicaciones Web

➤ Nuevos supuestos para Aplicaciones Web

- La experiencia de usuario (UX) tiene que ser fluida
 - Las animaciones y transiciones ayudan a interpretar la IU
 - Las nuevas propuestas de diseño, permiten la creación de sitios con una experiencia muy similar en móvil y en escritorio.
 - La velocidad actual de los motores de JavaScript (V8/Chrome, Chakra/Edge) permite la ejecución de código aprovechando todos los recursos de hardware del dispositivo.
- La adaptación al dispositivo es fundamental para que la programación sea única
- Las nuevas API están disponibles desde JavaScript: **Web Workers, Web Sockets, localStorage, sessionStorage, Notifications, File API, Drag&Drop**, etc.
- Estas API pueden integrarse con los servicios propios de AngularJS (1.x) y Angular (2+) para obtener soluciones extendidas más efectivas.

Razones para utilizar Angular (cualquier versión)

- Se pueden aportar varias según el perfil:
 - Para nuevos desarrolladores:
 - Popularidad
 - Demanda
 - Soporte y recursos
 - Front-End
- Para programadores expertos:
 - Marco de trabajo estructurado y bien probado.
 - Productividad
 - Consistencia (y soporte multiplataforma y multi-navegador)
- Para Jefes de Proyecto
 - Eficiencia
 - Longevidad (Google/Microsoft han anunciado su soporte continuado en el tiempo)

Razones para utilizar Angular

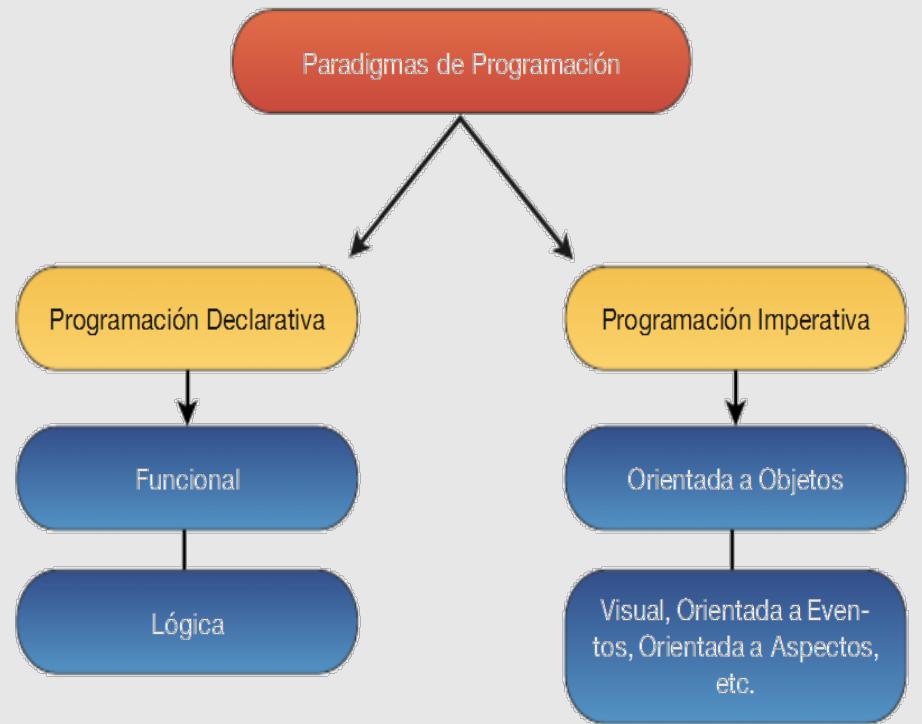
➤ Tenemos un gran conjunto de ventajas con esta aproximación:

- Visualización óptima de la información
- Facilidad de lectura y navegación
- Independencia del dispositivo
- Compatibilidad hasta IE9
- "Feedback" continuo (cualquier acción supone una respuesta)
- Acceso a datos de forma simplificada, mediante servicios Web
- Posibilidad de realizar Test Unitarios
- Escalabilidad y mantenimiento

Los inicios de AngularJS (v 1.x)

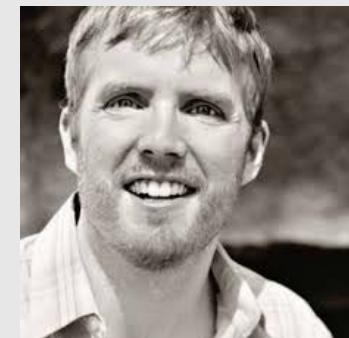
- Un grupo de desarrollo de Google, liderado por el arquitecto de software **Misko Hevery** (junto a **Brad Green** e **Igor Minar**), se planteó la necesidad de unificar los esfuerzos previos en este sentido.

- Principios a seguir:
 - La programación declarativa es ideal para la construcción de interfaces de usuario
 - Mientras que la programación imperativa lo es de cara a la creación de la lógica de negocio



Los inicios de Angular (2+): acuerdo con Microsoft

- En 2015 el equipo de desarrollo se da cuenta de que el volumen y la complejidad de Angular 2 necesita un meta-lenguaje de JavaScript más avanzado que **AtScript**.
- Conecta con el equipo de desarrollo de **Hejlsberg**, que construía **TypeScript** y –por primera vez- se hace un acuerdo de colaboración.
- Comienzan a usar V. Studio Code como herramienta de desarrollo
- El propio Hejlsberg, junto a y Jonathan Turner y James Foster participa habitualmente en eventos Google y al revés...



Diferencias principales entre Angular 1.x y Angular 2+

➤ Angular 1.x

- Marco estructurado
- Separación de HTML y Lógica de negocio
- Plantillas del lado del cliente.

➤ Angular 2+

- UI Basada en componentes
- Diseño más modular
- Construida con TypeScript
- Compatibilidad hacia atrás
 - Se puede añadir código Angular 2+ a una aplicación Angular 1.x.
- Rapidez: de 2 a 3 veces más rápido que Angular 1.x

Diferencias principales entre Angular 1.x y Angular 2+

➤ Comparativa inicial de código fuente

```
import { Component } from '@angular/core'

angular.module('myModule')
  .controller('myController',function(){
    })
<body>
  <div ng-controller="myController">
    </div>
</body>

@Component({
  selector: 'my-app',
  template: ``
})
export class AppComponent { }
```

Cuestiones a considerar en la decisión de marcos de trabajo

- Angular 2+ (mejor para...)
- Proyectos grandes
- Equipos de desarrollo numeroso
- Marco de trabajo "dogmático"
- Más complejo.
- Angular 1.x o incluso otros (jQuery, BootStrap, etc.)
- Proyectos más pequeños
- Equipos de desarrollo con pocas personas.
- Libertad total de abordaje
- Rapidez: de 2 a 3 veces más rápido que Angular 1.x
- Aplicaciones multi-página

Principios SOLID (recordatorio)

Inicial	Significado (acrónimo)	Concepto
S	SRP	Principio de Única Responsabilidad (<i>Single responsibility principle</i>). La noción de que un objeto solo debería tener una única responsabilidad. (usado en Angular)
O	OCP	Principio Abierto/Cerrado (<i>Open/Close Principle</i>). La noción de que las "entidades de software..." deben estar abiertas para su extensión, pero cerradas para su modificación".
L	LSP	Principio de sustitución de Liskov (<i>Liskov Substitution Principle</i>). La noción de que los "objetos de un programa deberían ser reemplazables por instancias de sus subtipos sin alterar el correcto funcionamiento del programa".
I	ISP	Principio de Segregación de la Interfaz (<i>Interface Segregation Principle</i>). La noción de que "muchas interfaces cliente específicas son mejores que una interfaz de propósito general."
D	DIP	Principio de Inversión de Dependencia (<i>Dependency inversion principle</i>). La noción de que uno debería "Depender de Abstracciones. No depender de concreciones." La Inyección de Dependencias es uno de los métodos que siguen este principio. (AngularJS la utiliza en su arquitectura)

Principios SOLID

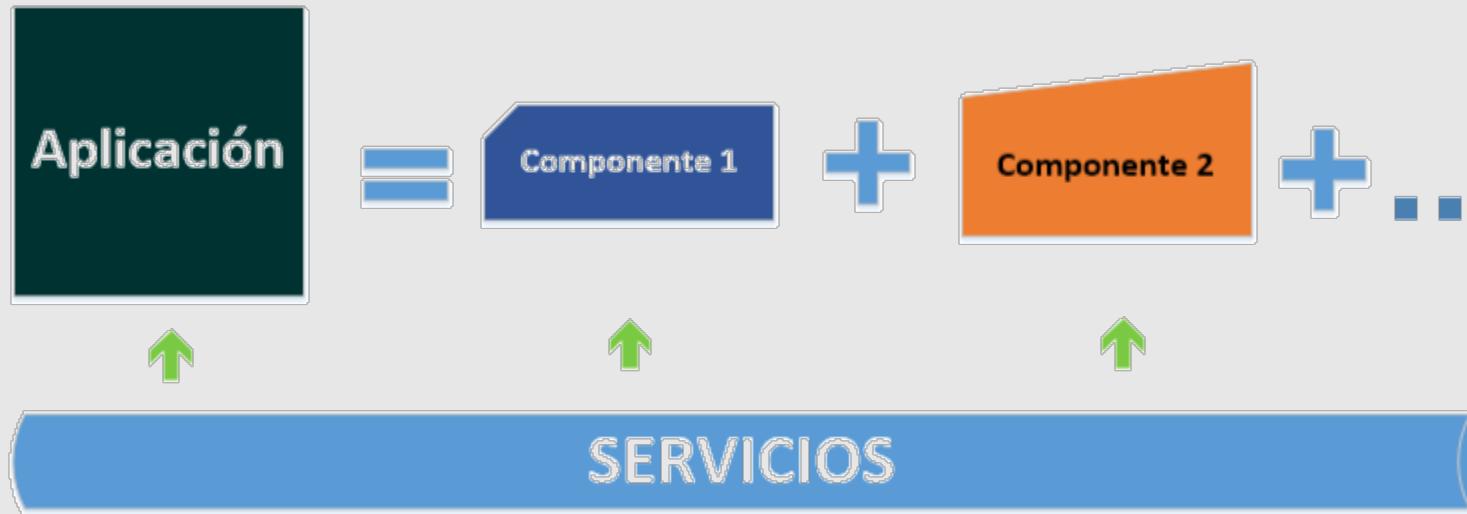
- El **Principio de Única Responsabilidad** enseña que cada objeto debe tener un propósito exclusivo.
 - Si usamos el ejemplo de un controlador, su responsabilidad es, esencialmente, controlar el alcance que tienen los modelos y sus vistas, y la forma en que ambos se relacionan.
 - Si el controlador también es responsable de hacer llamadas AJAX y buscar y actualizar datos, se estaría violando el principio SRP.
 - La lógica de ese tipo (y otras lógicas de negocio) deberían ser abstraídas en un servicio independiente, e inyectadas en los objetos que necesitan utilizarlo.

Principios SOLID

- El **Principio de Inyección de Dependencias** enseña que cada objeto debe tener un propósito exclusivo.
 - Éste indica que los objetos deben de depender de abstracciones, no de elementos concretos.
 - En lenguajes como C# o Java, esto significa dependencia de interfaces en lugar de clases concretas.
 - En JavaScript, las abstracciones serían los parámetros de cualquier función (función constructora o de otra clase), ya que podemos pasar cualquier objeto como parámetro, con tal de que el código interno de la función realice alguna operación con él (le "sirva" para su operativa).
 - Pero, en este caso, la clave es la capacidad de ser "inyectado" en otros objetos.
 - Esto significa que todos nuestros controladores, directivas, filtros y otros servicios deberían admitir cualquier dependencia externa como parámetro durante su construcción.
 - Esto tiene la doble ventaja de disponer de código débilmente acoplado (*loosely coupled*), y de favorecer los test unitarios.

Anatomía de una aplicación Angular 5

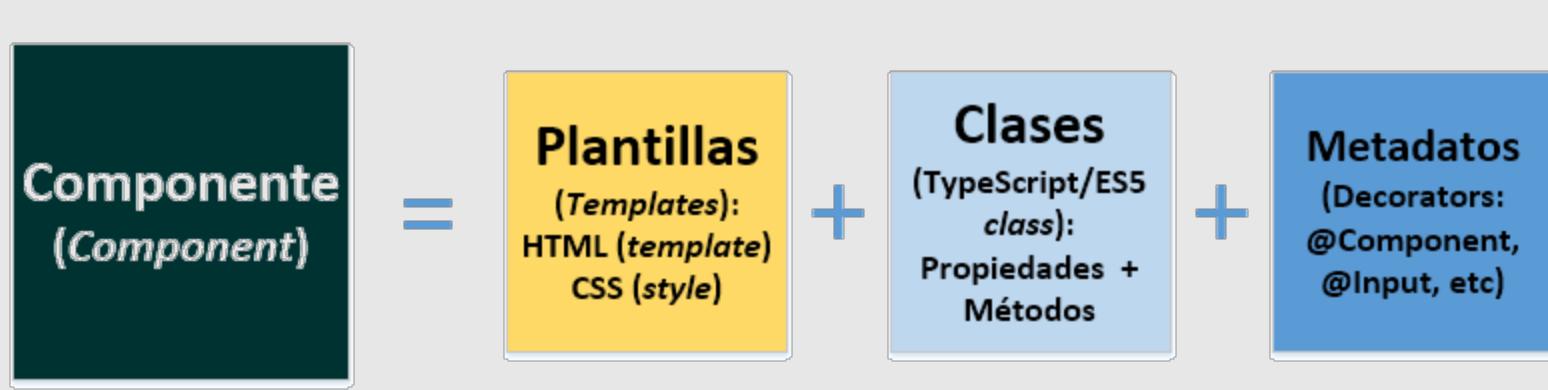
- Las aplicaciones Angular 5 están compuestas de componentes individuales, construidos de forma independiente.



- Estos componentes obtienen funcionalidad de los servicios disponibles en Angular 5
 - El desarrollador, puede construir sus propios servicios.

Anatomía de una aplicación Angular 5

- A su vez, un componente está constituido de 3 bloques principales:



- Una plantilla, que contiene la parte HTML (las vistas) y opcionalmente, estilos CSS
- Una clase, que aporta la parte funcional, y dispone de propiedades y métodos.
- Metadatos, que configuran el comportamiento de ambos y su relación entre sí.

Estándares Web en los que se basa Angular 5

- Para poder construir Angular 5 con esta arquitectura, el equipo de Angular se ha basado fundamentalmente en dos estándares Web de reciente creación (además de otros aspectos existentes en la versión 5 de JavaScript):
- **Web Components:** Permitir la creación de *widgets* o componentes reutilizables en los documentos web y aplicaciones web.
 - La intención detrás de ellos es llevar la ingeniería de software basado en componentes a la Web.
 - El modelo de componentes permite la encapsulación y la interoperabilidad de los elementos HTML individuales.
 - <http://webcomponents.org/>
 - <https://www.w3.org/standards/techs/components#completed>
- **Web Workers:** Permiten la ejecución de cálculos de pesados en el interior del contexto del subproceso diferente, lo que deja el hilo principal de ejecución libre, y capaz de manejar las entrada del usuario y hacer que la interfaz de usuario no se bloquee.

Web Components

- A su vez, este estándar habilita el uso de 3 características principales, a las que se añadió una 4^a finalmente:
 - **Shadow DOM** (la combinación de múltiples árboles DOM en una jerarquía y su interacción entre sí dentro de un documento, permitiendo así una mejor composición del DOM).
[\(https://www.w3.org/TR/2016/WD-shadow-dom-20160830/\)](https://www.w3.org/TR/2016/WD-shadow-dom-20160830/)
 - **Custom Elements** (elementos personalizados):
<https://www.w3.org/TR/2016/WD-custom-elements-20160830/>
 - **HTML Imports** (bloques de HTML reutilizables):
<https://www.w3.org/TR/2016/WD-html-imports-20160225/>
 - **HTML Templates** (plantillas HTML):
<https://www.w3.org/TR/2014/NOTE-html-templates-20140318/>

Comparativa: controladores

- Angular 1.x utiliza el patrón MVC.
- Las buenas prácticas recomiendan que no se manipule el DOM desde los controladores, sino a través de directivas.
- Si la aplicación requiere de cierta funcionalidad que puede necesitarse en varias partes, lo adecuado es crear un servicio y obtener esa funcionalidad mediante Inyección de Dependencias.
 - Por tanto, esa funcionalidad, puede moverse a los controladores internos de las directivas
 - Cuando una directiva recibe una entrada puede delegar su resolución a un servicio inyectado previamente.
 - Por eso, Angular 5 elimina la directiva **ng-controller**.

Comparativa: \$scope

- Debido al patrón MVC, en Angular 1.x se expone una jerarquía de modelos a través de **\$scope**.
- Esa jerarquía es responsable del enlace a datos (**one-way** o **two-way**).
- Pero también lo es de detectar los cambios y lanzar eventos que actualizan la IU
- En Angular 5 se eliminan estos objetos y todas las expresiones se evalúan en el contexto de un componente de IU concreto.
 - Las propiedades del componente se convierten en su estado, y ese estado es el \$scope del componente.
 - Más simple, y más natural desde el punto de vista de la OOP.

Comparativa: Inyección de dependencias

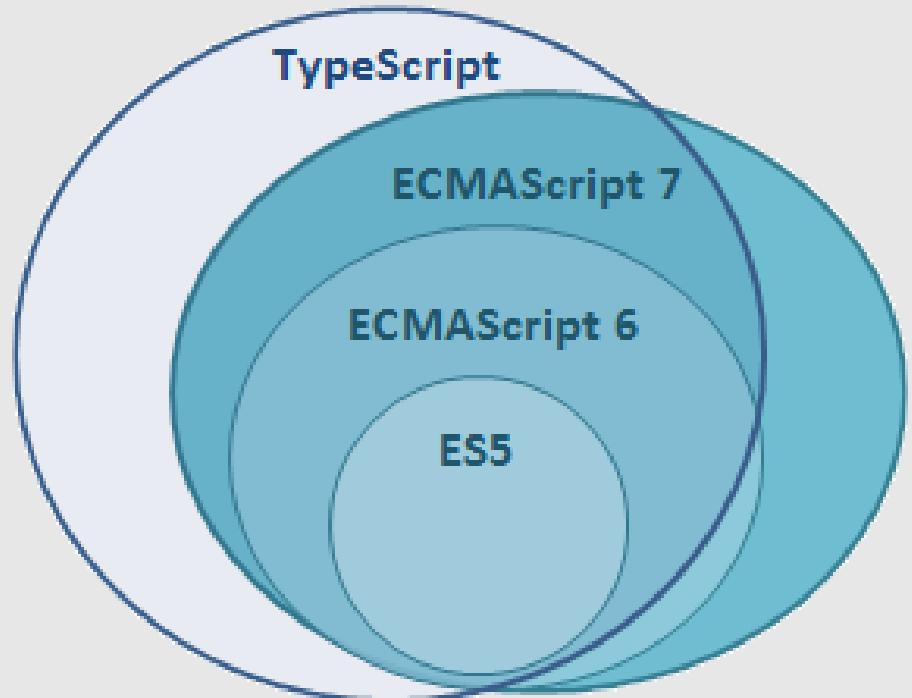
- Angular 1.x fue el primer *framework* en implantar la IoC (Inversión de control) a través de DI (*Dependency Injection*).
- Los metadatos de los componentes (y otros recursos), son injectados en Angular 5 mediante el uso de anotaciones (*Decorators*).
- La existencia de *Decorators* en TypeScript permite expresar estos metadatos de forma muy conveniente y clara en el código fuente.
- Angular 5 utiliza estas API para eliminar inconsistencias en la forma de aplicar la inyección.

Comparativa: One-way binding

- Angular 1.x utiliza un sistema de *binding* simple o doble (siendo el doble *binding* el predeterminado).
- Esto, muchas veces genera ciclos **\$digest**, que pueden provocar lentitud u otros problemas.
- NG-2 elimina el doble *binding* predeterminado, lo que consigue:
 - Un flujo de datos más explícito.
 - Elimina dependencias entre distintos *bindings*.
 - Mejora el rendimiento:
 - El bucle **\$digest** solo se ejecuta una vez.
 - Podemos crear aplicaciones que manejen modelos **inmutables/observables**., que nos permitan hacer optimizaciones posteriores.

Comparativa: Entorno fuertemente tipado

- En el JavaScript tradicional, es muy complicado realizar cosas como la refactorización (y más todavía la llamada refactorización inteligente).
- El sistema fuertemente tipado de TypeScript, reconoce las definiciones de tipos, facilitando la labor de los IDE's y editores.
- TypeScript es un superconjunto de JavaScript (que incluye ECMAScript 2015/2016/5 como subconjuntos)
- Disponemos de definiciones para todo tipo de librerías (incluso jQuery, Bootstrap, Node, etc.) en **DefinitelyTyped.com**)



Comparativa: Plantillas (Templates)

- Las plantillas se construyen con HTML simple, sin necesidad de motores especiales de interpretación, como **Moustache**, **Jade**, etc.
- Utilizan el estándar **HTML Templates**, creando un DSL (*Domain Specific Language*), que permite la creación de elementos y atributos personalizados.
- En esta versión se aprovecha lo mejor de la anterior, pero reduce algunas de las redundancias o complejidades que se daban en algunos escenarios complejos en 1.x
- La nueva sintaxis utiliza corchetes y paréntesis para indicar valores de atributos y asignaciones de eventos:

```
<li *ngFor="let elem of colección"
     (click)="selectElem(elem)">
    {{elem.propiedad}}
</li>
```

Comparativa: Detección de cambios

- Siguiendo el principio de separación de responsabilidades, NG-2 separa los motores de detección de cambios en capas distintas.
- Eso permite utilizar 2 mecanismos de detección:
 - **Detección de cambio básica**, similar a la utilizada en Angular 1.x, que se usa en contextos en que `eval()` está deshabilitado.
 - **Detección de cambios JIT**: El código que ejecuta la detección de cambios es generado en tiempo de ejecución (**Just-In-Time**), lo que permite a la máquina virtual de JavaScript ejecutar optimizaciones posteriores.
 - Al final, se traduce en menos código en ejecución y mejor rendimiento.

Resumen

- Hemos visto los principios detrás de la construcción de las dos versiones de Angular y, en especial, de Angular 2.
- La nueva arquitectura, propone un sistema totalmente modular, basado en las nuevas API de JavaScript
- Angular está construido con TypeScript, un superconjunto de JavaScript, orientado a objetos con una sintaxis similar a la que conocemos de C# o Java.
- Se ha conseguido la compatibilidad de navegadores hasta IE9 y además, es multi-plataforma.
- Su continuidad, está garantizada por Google y Microsoft.
- De cara al desarrollador, requiere conocimientos de ES5 o de TypeScript para la construcción de componentes.

angular-cli

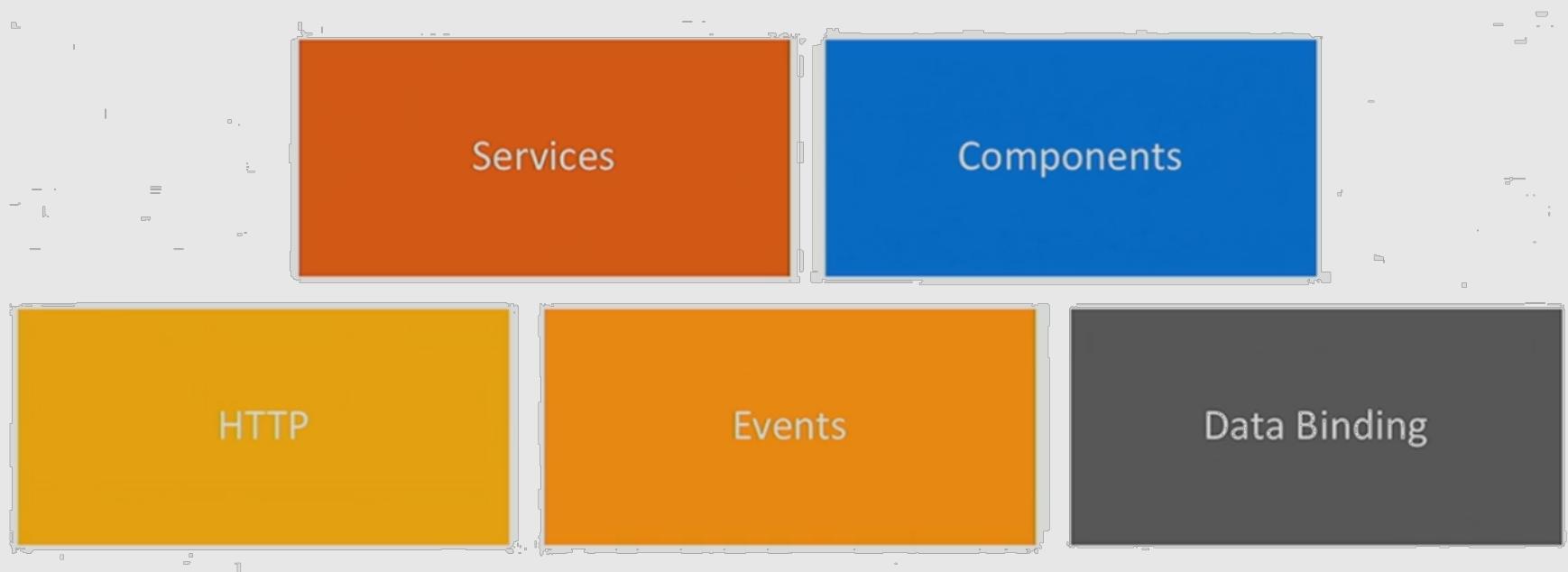
Arquitectura de Angular 5: Procesos de construcción

- En Angular 5 hay 3 procesos básicos que se siguen en la obtención de los entregables finales:
 - Compilación "off-line" (plantillas en el servidor)
 - **Tree-Shaking**: establece cuáles son los componentes imprescindibles para la distribución final
 - Minimización: produce un paquete final comprimido que contiene todo lo necesario.



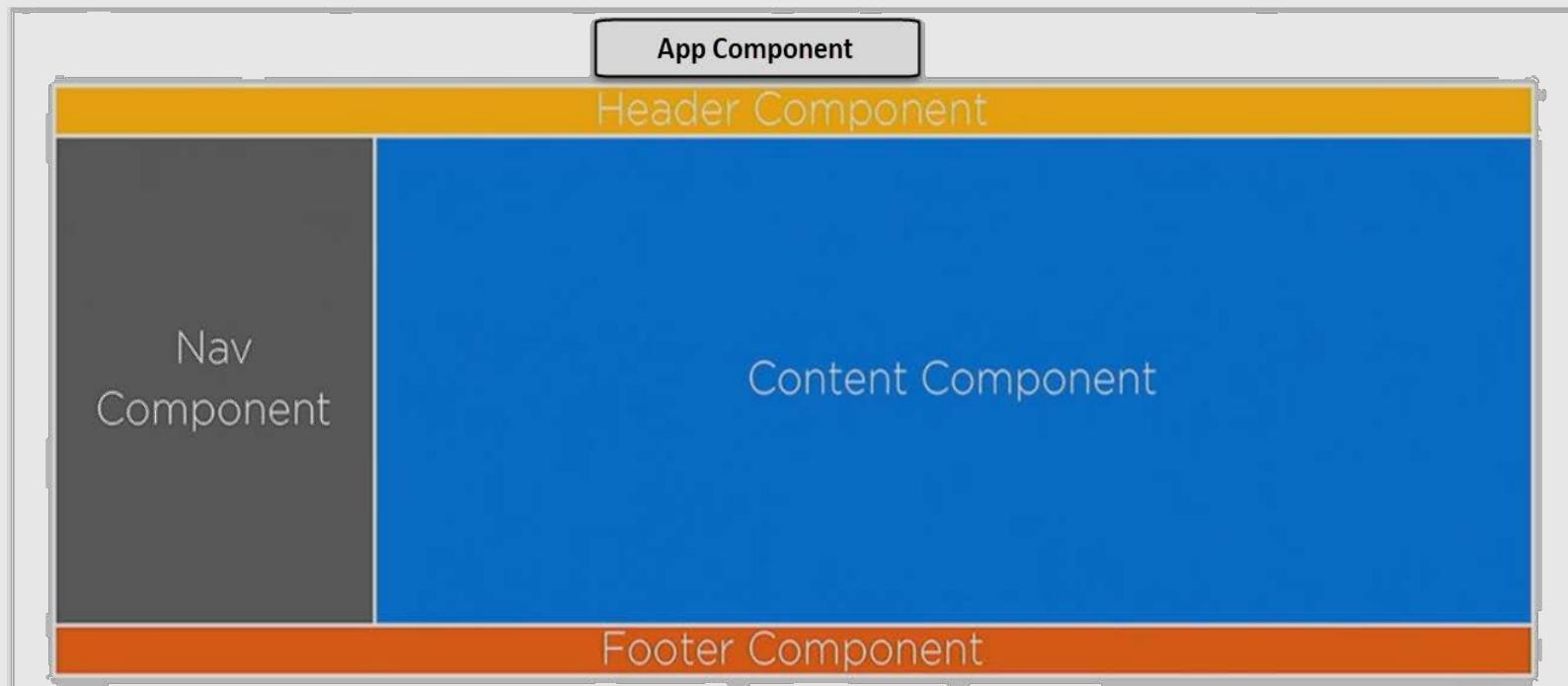
Arquitectura de Angular 5

- Existen 8 bloques de construcción básicos (*building blocks*) que pueden configurar una aplicación Angular 5:
 - **Modules, Components, Templates, Metadata, Data Binding, Directives, Services and Dependency injection**
- Decimos que los conceptos más habitualmente usados son:



Arquitectura de Angular 5

- Como hemos indicado, Angular 5 está compuesto de un conjunto de elementos diversos que se comunican e interactúan entre sí para formar una aplicación. Todos basados en la idea de componente.



Punto de entrada: Main.ts

- El punto de entrada de una aplicación angular, se establece en el fichero **main.ts**.
- Las partes mínimas que se requieren son estas simples líneas de código:



The screenshot shows a code editor with a dark theme. A blue header bar at the top has the text "main.ts". The main workspace contains the following TypeScript code:

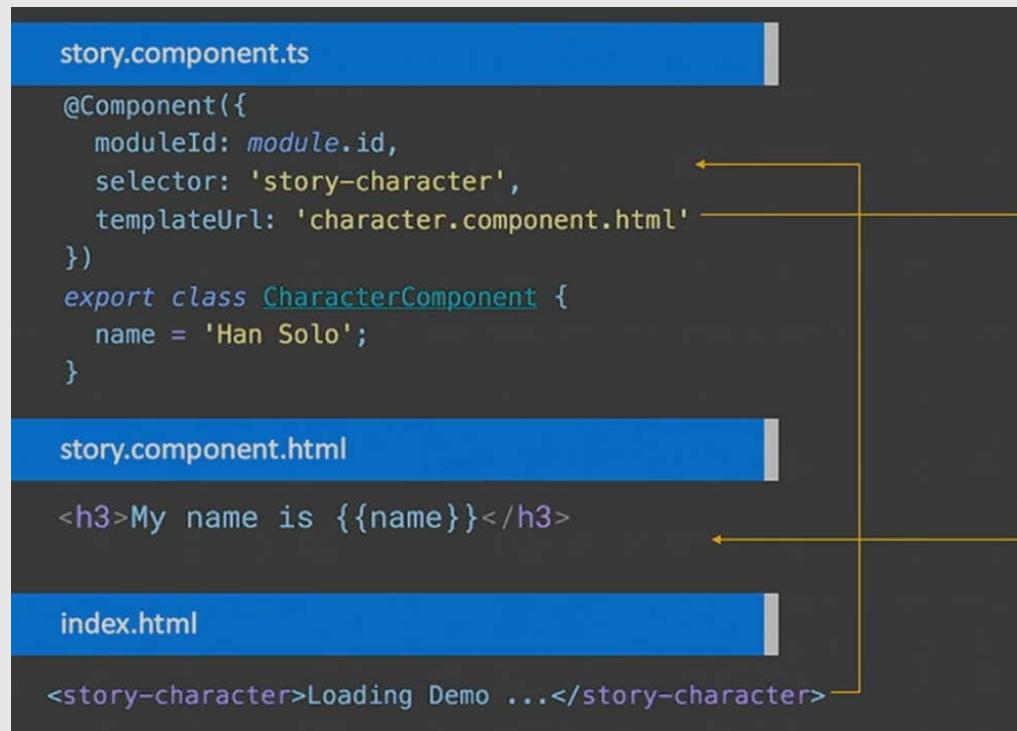
```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule);
```

- Observad que se le indica el tipo de plataforma (ahora pueden ser varias), y –sobre ella- se configura el módulo principal (AppModule) en el ejemplo, que ha sido previamente cargado.

Funcionamiento básico de un componente

- Todo componente, tendrá una referencia a una parte HTML y a un descriptor (*story-character*, en el ejemplo).
- Se usa en la página principal para indicar en qué parte del DOM debe de ir el resultado del proceso de construcción



Configuración del entorno

- Se necesita **Node.js**, y su gestor de paquetes **NPM**
- Si queremos empezar con una aplicación básica (el *QuickStart*), podemos instalarla desde NPM, o podemos trabajar en un proyecto desde cero, siguiendo las pautas de construcción.
- Lo recomendado en esta versión 4.0 es instalar la herramienta de línea de comandos Angular CLI

```
> npm install -g @angular/cli  
  
> ng new my-dream-app  
  
> cd my-dream-app  
  
> ng serve
```

- En principio, solo con esto, hay una aplicación muy básica, que funciona en cualquier navegador (<http://localhost:4200>, por defecto)

Configuración de Node y NPM

Comprobación de instalaciones

Creación de la aplicación inicial

Puesta en marcha

Ejecución en navegador

Configuración del entorno

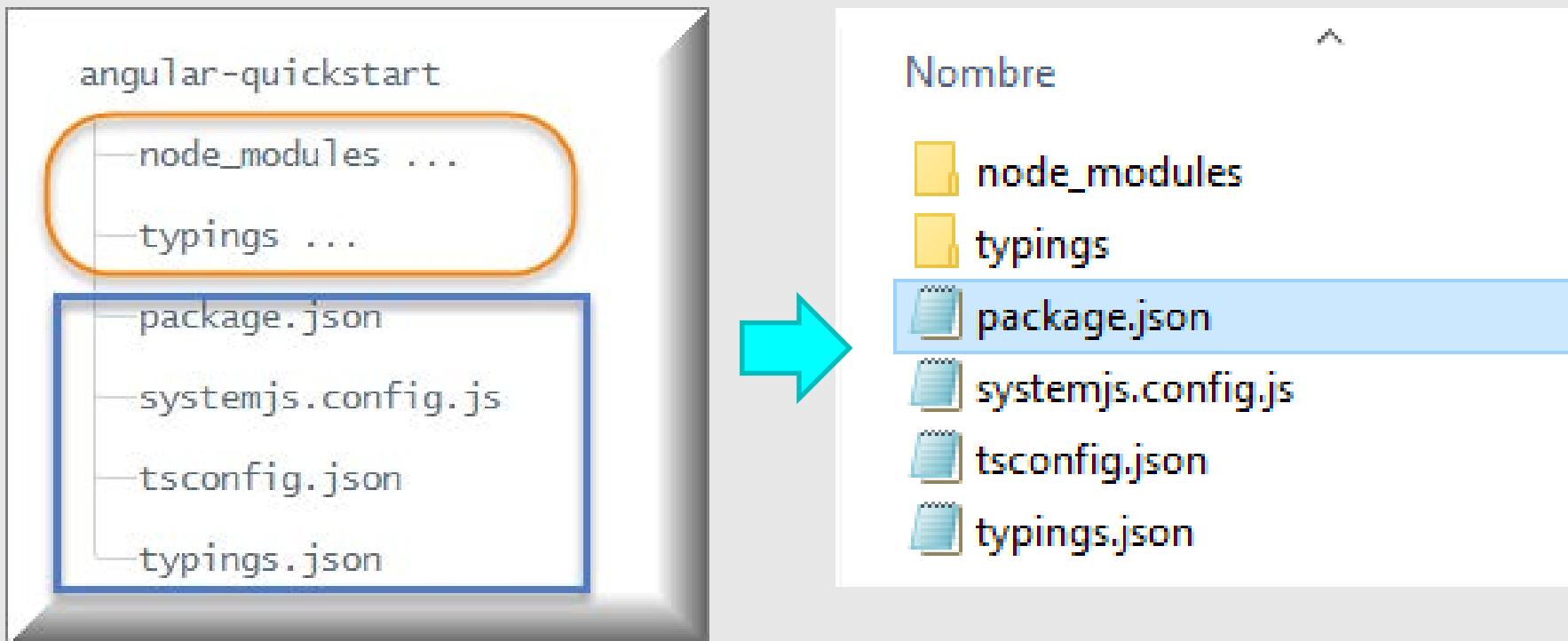
- El comando **ng serve** lanza el servidor, vigila los cambios en los archivos y recompila la aplicación instantáneamente.
- La configuración inicial requiere de un directorio exclusivo donde se ubique la aplicación, y la presencia de varios ficheros **.json** de configuración. Como mínimo, necesitamos:
 - **package.json**
 - **tsconfig.json**
 - **typings.json**
 - Un fichero de configuración para la carga de paquetes, que puede ser del tipo **systemjs** o **webpack**. P.e.; **systemjs.config.js**
- A partir de esas configuraciones, **npm** instalará las dependencias con una simple llamada a **npm install** que lee el fichero de configuración y descarga lo necesario.
 - Cada vez que ejecutamos **npm install** se recorren los archivos de configuración y se descargan las dependencias indicadas en ellos.

Instalación del entorno: archivos de configuración

- **package.json**: identifica los archivos de dependencias del proyecto que serán resueltos por NPM
- **tsconfig.json**: establece cómo se comporta el compilador de TypeScript
- **typings.json**: proporciona definiciones para las librerías que el compilador TypeScript no reconoce de forma nativa.
- Un sistema de carga de módulos: puede ser **Systemjs**, **WebPack**, o cualquier otro. El sitio oficial proponía **Systemjs.config.js** en la versión 2, *pero han cambiado a WebPack en esta versión.*
 - Además de esa labor, registra los paquetes necesarios, y contiene otros paquetes que pueden usarse posteriormente.

Preparación de la demo inicial

- La estructura final de directorios debería tener esta disposición:

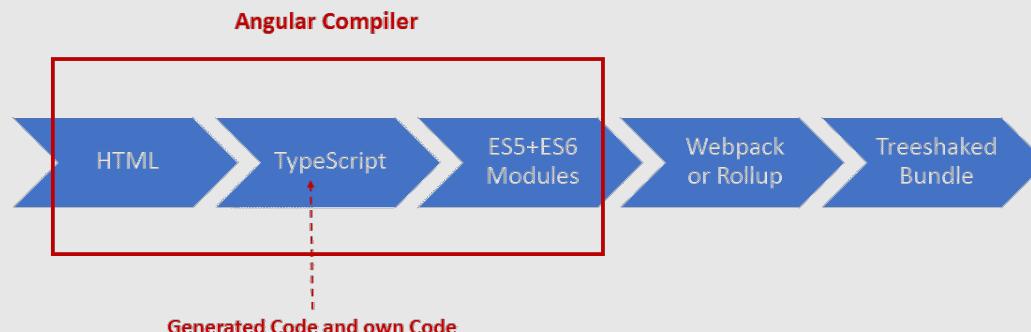


- Si la carpeta **typings** no apareciese, deberemos instalarla manualmente, mediante:

- **npm run typings install**

Crear la aplicación inicial

- Ten presente que todo el contenido de la carpeta **npm_modules** es el equivalente a lo que sería la instalación de .NET o J2EE con sus bibliotecas en una máquina cualquiera.
 - No merece la pena copiar su contenido para llevar el proyecto a otra ubicación: es preferible copiar el código y los ficheros de instalación y llamar a **npm install**.
- Además, en la fase de implantación/producción, la mayoría de estos componentes no se incluirán y lo necesario serán empaquetado utilizando cualquier herramienta de "*bundling and minifying*".
 - **Webpack, Gulp, Grunt**, etc.
- Este es el proceso conocido como "*Tree Shaking*"



Editar la aplicación inicial

- Podemos usar **Visual Studio Code** para editar el resultado
 - Recuerda que si no estuviera instalado en la máquina, deberías instalarlo desde el sitio oficial (<https://code.visualstudio.com>)
- Para abrir el proyecto seguimos estos pasos:
 - Si hemos seguido el proceso anterior, basta con movernos al nuevo directorio y teclear: **code .**
 - Esto abre la carpeta como un nuevo proyecto en VS Code.
 - También es posible que VS Code cree una subcarpeta para almacenar archivos propios de configuración del entorno (depende de los comandos CLI utilizados).

El módulo inicial

- Toda aplicación Angular 5 debe de tener –al menos- un módulo (el módulo raíz).
- Una clase se convierte en un módulo cuando esta "decorada" con el *decorator* ***NgModule***. Al importar ***AppComponent***, pone en marcha lo indicado en el componente.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/http';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, FormsModule, HttpClientModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

El componente inicial

- Definimos una clase **AppModule** y la exportamos para su uso por otros componentes.
- A su vez, esa clase importa **@angular/core** y **@angular/platform-browser** con la sintaxis que ya hemos visto en la parte de TypeScript.
- La clase contiene un **decorator (@NgModule)** que declara la importación de **BrowserModule** para uso por la clase haciendo referencia a él en el atributo **imports**
 - Este atributo define un array de elementos importados que serán usados después.
 - En aplicaciones más realistas, se necesitaran otros módulos, como **FormsModule**, **RouterModule** o **HttpModule**.

El componente inicial

- También hemos comentado que toda aplicación Angular 5 tiene, al menos, un componente (el componente raíz).
- Por convención, creamos en la misma carpeta ese componente raíz, con el nombre **app.component.ts**. Su contenido será un componente de nombre **AppComponent**.
- El código a utilizar será :

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Text changed!';
}
```

- También podemos copiar/pegar desde el sitio oficial
- Recuerda que si la plantilla requiere más de una línea debes usar la **tilde** del "francés" (en la tecla del corchete izquierdo).

El componente inicial

- En este caso, el componente importa la definición del *decorator* **@Component** de "**@angular/core**".
- A continuación define la IU del componente dentro del decorador. Esto lleva dos partes:
 - Una definición de directiva (lo que será usado en la página HTML)
 - Una plantilla HTML formada por una cadena con ese formato
 - O sea, usaremos la primera, y se mostrará la segunda.
 - Alternativamente, se puede utilizar la definición **templateUrl** apuntando a un fragmento HTML separado en un archivo (una vista).
- Por último, la propia definición de la clase, a la que se asocia el decorador (exportada, para poder ser usada externamente).
- Más adelante llevará su propia lógica de negocio, pero aquí su única función es servir de soporte al decorador.

El componente inicial

- Ahora, tenemos que añadir este componente a nuestro módulo inicial.
- Para ello modificamos el código

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Text changed!';
}
```

- Si se observa un subrayado rojo, pero al final compila, es un problema del editor, (no aparece a partir de la versión 2.0+ de TypeScript).
- Ahora tenemos un módulo que hace referencia a un componente con una pequeña interfaz de usuario

main.ts: revisión del bootstrapping (puesta en marcha)

- El último bloque TypeScript pone en marcha el motor de Angular para que reconozca nuestros módulos (y componentes): **main.ts**
- Hay que configurarlo para navegadores, con **platformBrowserDynamic**, que devuelve un objeto preparado para hacer el "bootstrapping" o proceso de carga inicial de angular
 - Hay que pensar que este código está preparado para ejecutarse en cualquier motor que lo soporte, de cualquier plataforma. Esto establece una referencia a una plataforma concreta.

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule);
```

Index.html (página principal)

- Por último, la página **index.html** es la que aprovecha esta funcionalidad y muestra el contenido.
- Se le injectarán las librerías necesarias e incluirá un elemento de la IU que coincide con el que hemos definido en nuestro componente:
`<app-root>Loading...</app-root>`
- En esta versión no aparecen referencias a librerías (que sí aparecían en versiones 2+), que son injectadas en tiempo de "*deployment*" por el entorno de desarrollo.
 - Eso es parte del "**bundling and minifying**" que ya hemos citado
- Sin embargo en tiempo de ejecución sí veremos esas referencias

```
<script type="text/javascript" src="inline.bundle.js"></script>
<script type="text/javascript" src="polyfills.bundle.js"></script>
<script type="text/javascript" src="styles.bundle.js"></script>
<script type="text/javascript" src="vendor.bundle.js"></script>
<script type="text/javascript" src="main.bundle.js"></script>
```

Index.html (página principal)

- El código fuente inicial (en edición), tendrá este aspecto:

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Ng41</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

Encadenamiento de procesos de carga

- Con todas estas configuraciones y relaciones, lo que ha sucedido es un encadenamiento de procesos de carga.
- Todo empieza con la definición incluida en el archivo **angular.cli.json** (o similar) sobre cuál es el módulo de *bootstrap* y cual la página de inicio:

```
  "apps": [
    {
      "root": "src",
      "outDir": "dist",
      "assets": [
        "assets",
        "favicon.ico"
      ],
      "index": "index.html", //
      "main": "main.ts",
      "polyfills": "polyfills.ts",
      "test": "test.ts",
      "tsconfig": "tsconfig.app.json",
      "testTsconfig": "tsconfig.spec.json",
      "prefix": "app",
      "styles": [
        "styles.css"
      ],
      "scripts": [],
      "environmentSource": "environments/environment.ts",
      "environmentMain": "environments/environment.ts"
    }
  ]
```

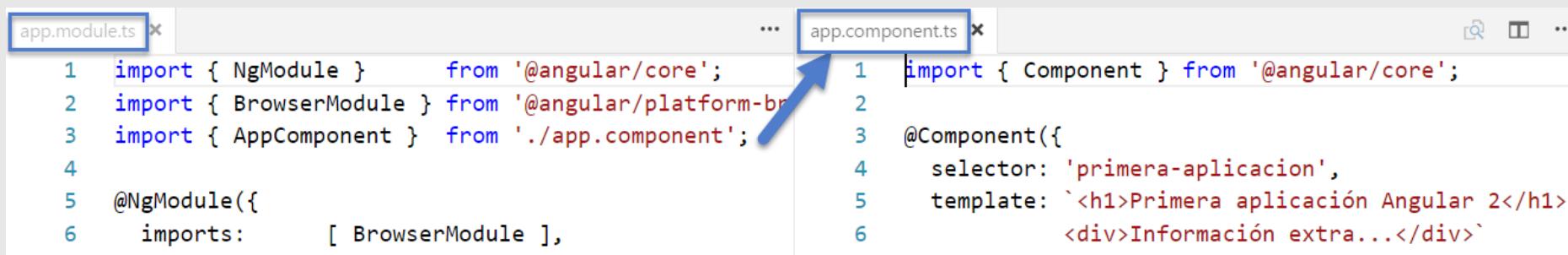
Encadenamiento de procesos de carga

- Hecho esto, Webpack entra en acción al llamar a los comandos
 - ng build
 - ng serve
- Ahí, se produce un proceso complejo de filtrado y empaquetado de fragmentos JavaScript, HTML y CSS
- Como, a su vez, **main.ts** carga **AppModule** y lo registra con:

```
platform.bootstrapModule (AppModule)
```
- ...que lanza toda secuencia.
- Cuando el archivo indicado (**index.html** u otro), ha sido "inyectado" con la información necesaria, ya podemos abrir la página en cualquier navegador sobre el puerto indicado (**4200**, por defecto).

Encadenamiento de procesos de carga

- Pero, **app.module**, a su vez, importa **app.component** (entre otras cosas), y es **app.component** el que define la interfaz de usuario.



```
app.module.ts x
1 import { NgModule }      from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3 import { AppComponent }  from './app.component';
4
5 @NgModule({
6   imports:      [ BrowserModule ],
```

```
... app.component.ts x
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'primera-aplicacion',
5   template: `<h1>Primera aplicación Angular 2</h1>
6   <div>Información extra...</div>`
```

- Aunque el proceso parece muy rebuscado, lo que sucede en realidad es que se aproxima mucho más al proceso de funcionamiento real (un tanto "escondido" en versiones anteriores de Angular).
- De esa forma tenemos más libertad a la hora de elegir la cadena de funcionamiento (incluso la inicial, asociada a la llamada a Platform.bootstrapModule (AppModule))

Puesta en marcha

- Podemos, antes de hacer la puesta en marcha, añadir algún estilo CSS o usar otras librerías como BootStrap.css.
- En este caso, añadimos un archivo del mismo nombre "Styles.css" en la raíz de la aplicación.
- Para ponerla en marcha hay múltiples opciones, ya que depende del servidor Web que queremos montar (vale cualquiera).
- De manera predeterminada, usa un servidor ligero (**lite-server**), que se instala dinámicamente desde **npm**, cuando lanzamos la aplicación con el comando **npm start**.
- Si estamos utilizando Visual Studio Code, podemos abrir una ventana terminal en la parte inferior del editor de código mediante CTL+Ñ o en el menú "*Ver/Terminal Integrado*".
- Desde ahí podemos lanzar cualquier comando, como desde la ventana exterior

Puesta en marcha

- Una vez en ejecución veríamos esto en la terminal:

```
webpack: Compiled successfully.  
webpack: Compiling...  
Hash: fb62c3f30bea50661c38  
Time: 113ms  
chunk {0} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 157 kB {4} [initial]  
chunk {1} styles.bundle.js, styles.bundle.js.map (styles) 65.2 kB {4} [initial]  
chunk {2} main.bundle.js, main.bundle.js.map (main) 3.73 kB {3} [initial] [rendered]  
chunk {3} vendor.bundle.js, vendor.bundle.js.map (vendor) 2.45 MB [initial]  
chunk {4} inline.bundle.js, inline.bundle.js.map (inline) 0 bytes [entry]  
webpack: Compiled successfully.
```

- El compilador se lanza en modo "watch" ("tsc -w")
 - Cualquier cambio en el código se reflejará inmediatamente en el navegador
- Podemos garantizar esto, configurando V.S.Code para grabar automáticamente después de una corta espera
 - En "*Preferencias/Configuración de usuario*", añadimos la entrada
 - **"files.autoSave": "afterDelay"**

Puesta en marcha

- Y se lanza la ejecución en el navegador predeterminado:



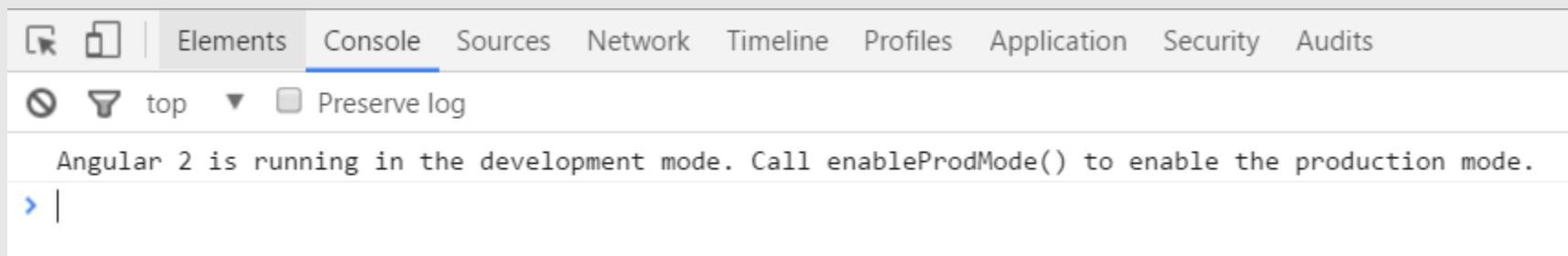
¡Primera Aplicación Angular 4!

Información extra...

- Naturalmente, también podemos indicar qué navegador queremos utilizar en los archivos de configuración de VSCode o en el proyecto.
- Para detener la ejecución del servidor pulsamos CTL+C en el terminal.
- Naturalmente, también podríamos usar el propio Node.js como servidor, o IIS, o incluso hacer una aplicación ASP.NET Core y auto-alojar la aplicación en una solución de consola u otra.
- Podemos probar a cambiar cualquier cosa de la interfaz y veremos cómo la terminal recompila y actualiza la IU.

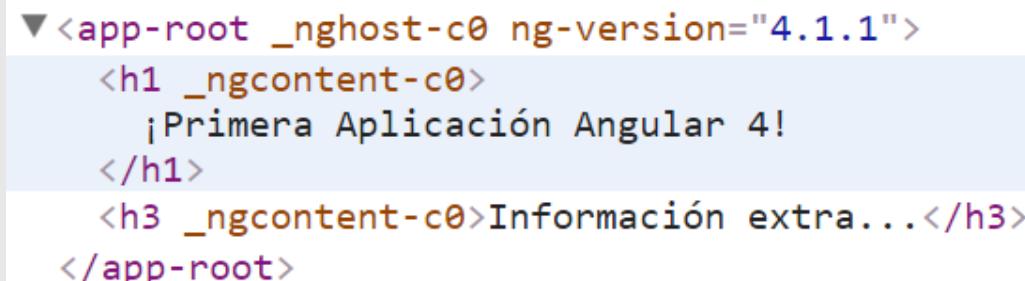
Puesta en marcha: Depuración

- Como siempre, podemos ver qué ha sucedido con las herramientas de desarrollador (F12):
- En la consola se nos avisa de que estamos en modo desarrollo:



A screenshot of the Chrome DevTools interface, specifically the 'Console' tab. The tab bar above shows 'Elements', 'Console' (which is selected), 'Sources', 'Network', 'Timeline', 'Profiles', 'Application', 'Security', and 'Audits'. Below the tab bar, there are filter icons for 'All' (unchecked), 'Logs' (checked), 'Top' (selected), and 'Preserve log' (unchecked). The main console area displays the following message:
Angular 2 is running in the development mode. Call enableProdMode() to enable the production mode.
A cursor icon is positioned at the end of the message line.

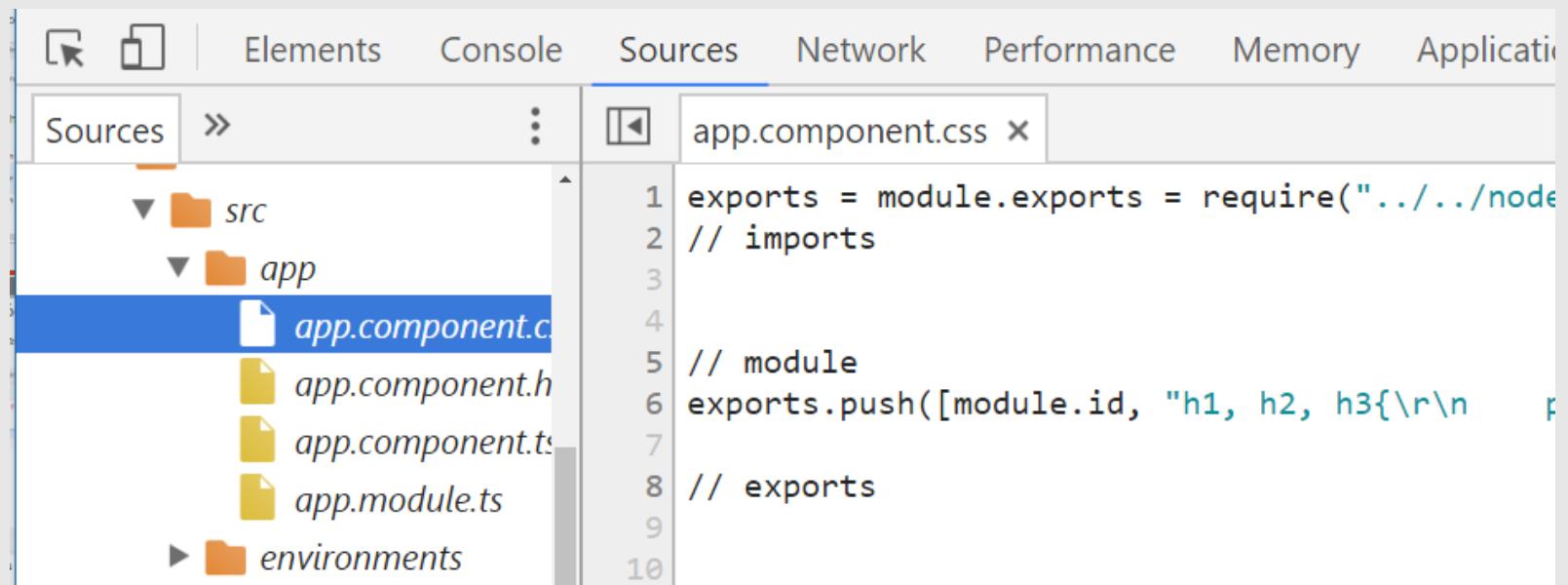
- Y en la parte del DOM, vemos en qué se ha convertido nuestro **template**:



A screenshot of the Chrome DevTools 'Elements' tab. It shows the rendered DOM structure of the application. The root element is `<app-root _ngcontent-c0 ng-version="4.1.1">`. Inside it, there is an `<h1 _ngcontent-c0>` element containing the text '¡Primera Aplicación Angular 4!'. Below that is an `</h1>` tag. Underneath the `</h1>` tag is another `<h3 _ngcontent-c0>` element containing the text 'Información extra...'. Finally, the entire structure is closed by a `</app-root>` tag. The entire code block is highlighted with a light blue background.

Puesta en marcha: Depuración

- De igual forma, podemos ver la solapa "Sources", con todos los detalles:



The screenshot shows the Chrome DevTools interface with the "Sources" tab selected. On the left, the file tree shows the project structure: src/app. The file app.component.css is selected and shown in the main pane. The code content is as follows:

```
1 exports = module.exports = require("../node_modules/raw-loader/dist/index.js");
2 // imports
3
4
5 // module
6 exports.push([module.id, "h1, h2, h3{\r\n    font-size: 1em;\r\n    font-weight: bold;\r\n    color: #000;\r\n}\r\nh1 {\r\n    font-size: 1.5em;\r\n}\r\nh2 {\r\n    font-size: 1.2em;\r\n}\r\nh3 {\r\n    font-size: 1.1em;\r\n}\r\n\r\nh1, h2, h3 {\r\n    margin: 0;\r\n    padding: 0;\r\n}"]);
7
8 // exports
9
10
```

- Incluso podemos poner puntos de ruptura en el código TypeScript.

Puesta en marcha: Depuración

- También es posible analizar lo que angular y/o el entorno de desarrollo añade a nuestra página para el soporte extendido
- Como **BrowserSync** o las asignaciones a eventos globales:

The screenshot shows a code editor with several tabs: app.component.js, main.ts, (index) (highlighted), styles.css, and systemjs.config.js. The (index) tab contains the following HTML code:

```
18     </script>
19 </head>
20 <!-- 3. Muestra la aplicación -->
21 <body><script id="__bs_script__">//<![CDATA[
22   document.write("<script async src='/browser-sync/browser-sync-client
23 //]></script>
24
25   <primera-aplicacion>Cargando...</primera-aplicacion>
26 </body>
27 </html>
```

A red box highlights the browser-sync client script tag. A blue arrow points from this highlighted area to the developer tools panel on the right. The developer tools panel shows the "Event Listeners" section with the "beforeunload" event listener for the "Window" object. The event details show:

- handler: function ()
- useCapture: false
- passive: false

zone.js:301

- Así que disponemos de un entorno completo de desarrollo multi-plataforma, con todas las capacidades habituales.

Otra demo inicial

- Existe una demo inicial (lo más simple que les ha sido posible) instalada y accesible en Plunker
- <https://embed.plnkr.co/?show=preview&show=app%2Fapp.component.ts>

The screenshot shows a code editor interface for an Angular project named "Angular Example - QuickStart". The left sidebar displays the project structure:

- Project
- app
 - app.component.ts
 - app.module.ts
- index.html
- main.ts
- systemjs-angular-loader.js
- systemjs.config.js

The right side of the interface is divided into several panes:

- Preview**: Shows the rendered output "Hello Angular".
- Search**: Search bar.
- Code Editor**: The file `app/app.component.ts` is open, containing the following code:

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1>`
})
export class AppComponent { name = 'Angular'; }

/*
Copyright 2017 Google Inc. All Rights Reserved.
Use of this source code is governed by an MIT
license that can be found in the LICENSE file at http://ar
*/
```

Referencias

➤ Documentación y vídeos

- Introducción a Visual Studio Code (Videos):
 - <https://code.visualstudio.com/docs/introvideos/basics>

- Angular 5 QuickStart Tutorial:

- <https://angular.io/docs/ts/latest/quickstart.html>

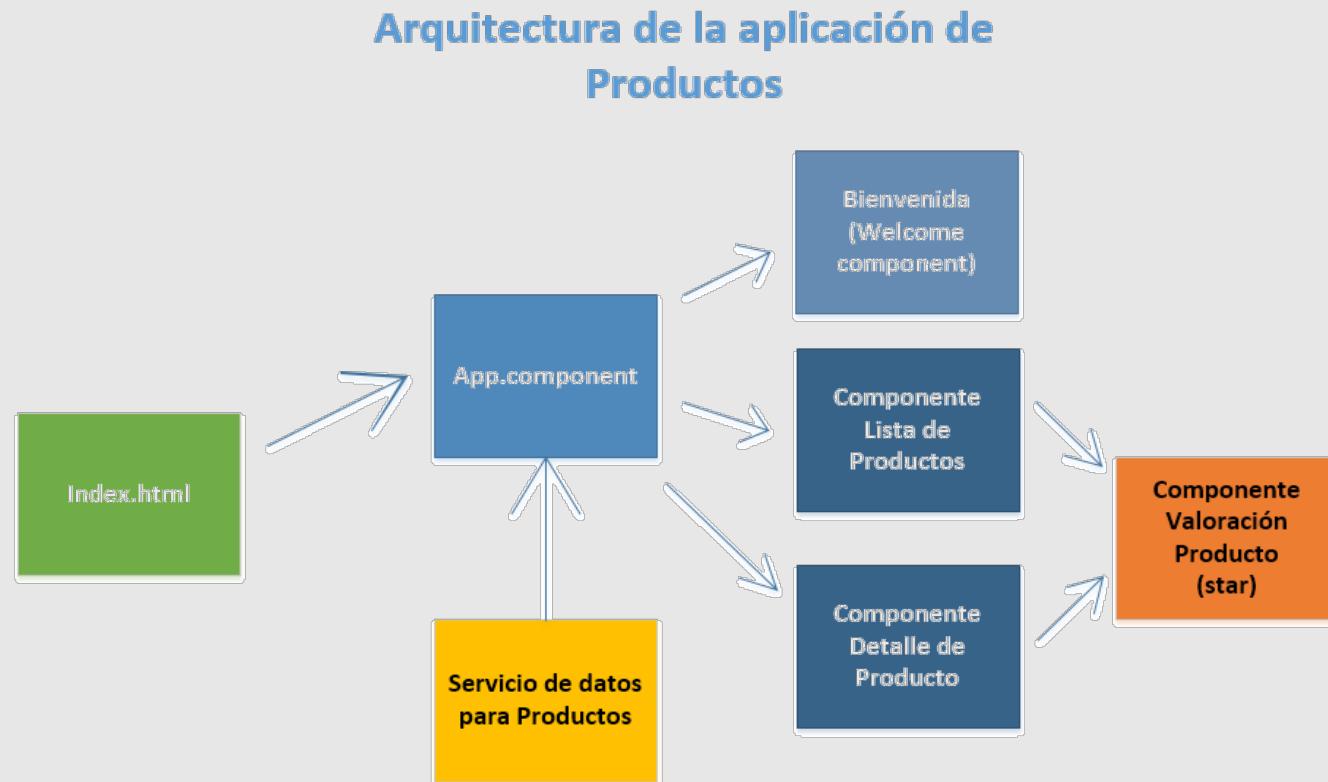
- Módulos en Angular 5:

- <https://angular.io/docs/ts/latest/guide/ngmodule.html>

Componentes

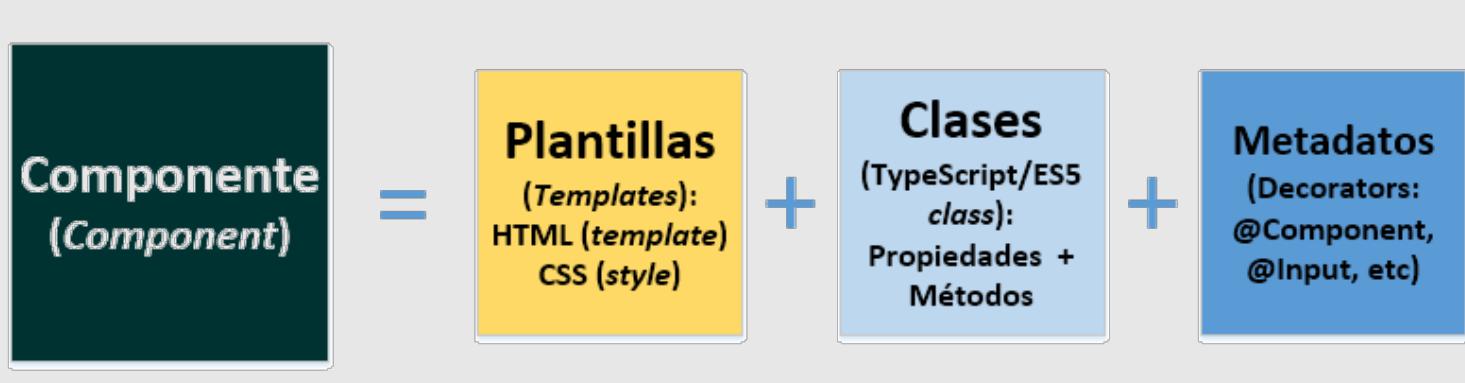
Construcción de componentes

- Vamos a plantear una aplicación simple, de gestión de productos, y vamos a ir viendo a través de su construcción los distintos aspectos de Angular 5



Estructura de un componente

- Recordemos el esquema de componentes:



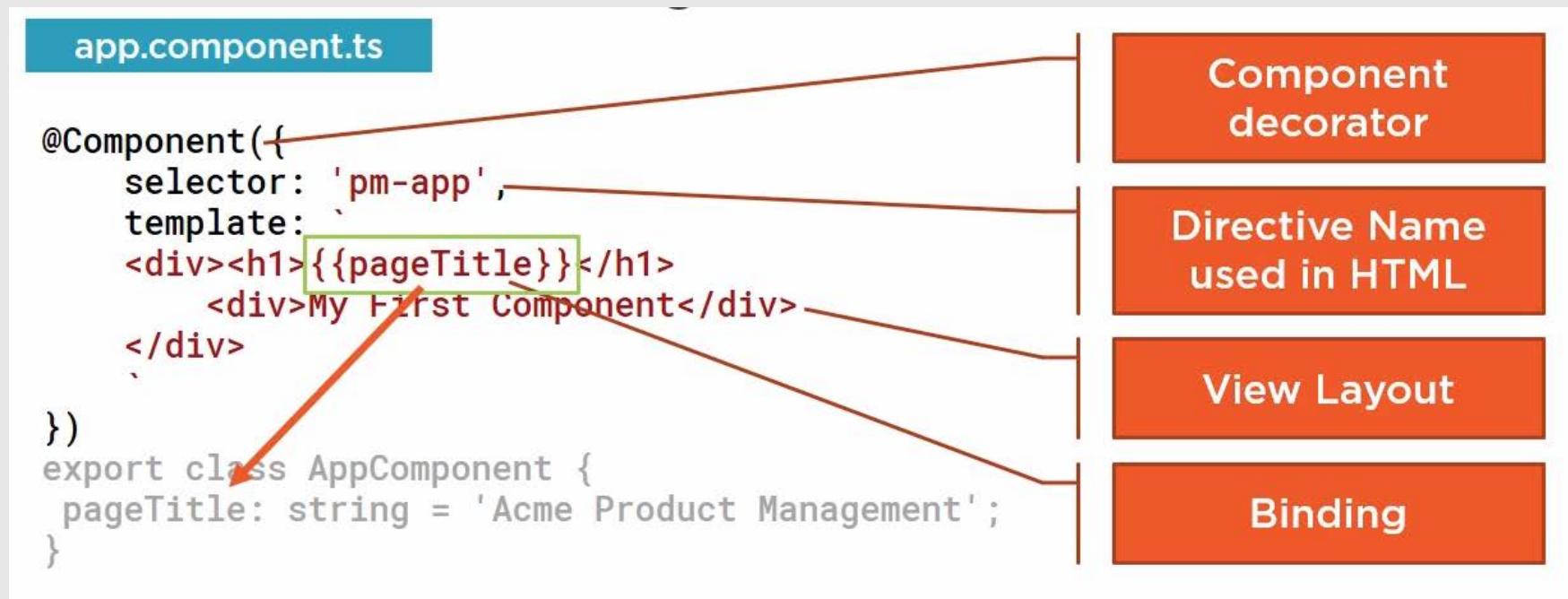
- La plantilla pueden contener "*bindings*" para enlazar con datos almacenados en la clase (usando sintaxis *moustache* `{{dato}}`) así como otras directivas definidas por Angular.
- Una clase, que aporta los datos que debe enlazar la IU y la parte funcional (dispone de propiedades y métodos).
- Metadatos (*decorators*), que configuran el comportamiento de ambos y su relación entre sí, y los registra en el "motor" de Angular.

Decorators

- Un decorador no es más que una función que añade metadatos a una clase, un método o una propiedad.
- Los decoradores, no están disponibles todavía en JavaScript (ninguna versión), aunque están propuestas para la siguiente (ES7).
- Siempre se programan con el prefijo @
- Algunos, están incluidos en Angular de manera predeterminada:
 - **@Component**
 - **@NgModule**
 - **@Router**
 - **@Injectable**
 - **@Inject**
 - **@View**
 - **@Input**
 - **Etc...**

Decorators

- Los elementos que componen un *decorator* podemos analizarlos en el siguiente esquema de código:



- Finalmente, para que **@Component** sea reconocido, utilizamos en la aplicación inicial la sentencia:

```
import { Component } from '@angular/core'
```

El componente inicial de la primera aplicación

- Podemos basarnos en la aplicación **quickstart**, copiar su contenido en otro directorio, y cambiar algunas cosas para adaptarlas a nuestro proyecto.
- Además, así repasamos la cadena de control, y probamos alguna funcionalidad extra.
- La demo **app_productos_1** es el punto de partida con unos elementos básicos cambiados y adaptados a la nueva propuesta.
- Hay una carpeta "**recursos**" que sirve para almacenar los datos externos que maneje la aplicación.
 - Iconos, archivos de datos .json, gráficos, fuentes, etc.
- Hemos añadido el **favicon** de la página (aunque en Edge podría no verse desde **localhost** sino solo en producción (dependiendo de la versión)).
- En IE10+ y en resto de navegadores, se ve perfectamente también desde el modo desarrollo.

El componente inicial de la primera aplicación

- Podemos basarnos en la aplicación **quickstart**, copiar su contenido en otro directorio, y cambiar algunas cosas para adaptarla a nuestro proyecto.
- También hemos añadido un elemento **styles** a la plantilla del componente para comprobar la sintaxis y la funcionalidad.
- Igual que sucede con la plantilla, si se incrusta directamente el código CSS se coloca entre comilla invertida (tilde francesa o "*back-tick*": `).
- Eso permite escribir mas de una línea de definición. (también podríamos haber incluido una etiqueta <**style**> dentro de la propia plantilla)

El componente inicial de la primera aplicación

- Estos estilos solo se aplican al componente donde se definen.
- Esto evita colisiones de nombres con otras definiciones (propias o ajenas)
- Además, los cambios de estilo en otras zonas de la página **no afectan** a lo definido por estos estilos.
- Por supuesto, la forma alternativa consiste en indicar un archivo separado para la parte HTML mediante **templateUrl** y lo mismo para los archivos CSS, mediante la definición **styleUrls**.
 - **Estos enlaces serían relativos a la raíz del sitio, no al path del componente que los define.**
 - También podemos usar la directiva @import 'fichero.css';

Componentes

Demo componentes con estilo y plantillas

Referencias

- Arquitectura y documentación oficial de Angular
 - Sitio oficial de Angular:
 - <https://angularjs.org/>
 - Documentación oficial:
 - <https://drive.google.com/drive/folders/0BxgtL8yFJbacQmpCc1NMV3d5dnM>
 - Última documentación sobre Angular
 - <https://angular.io/docs/ts/latest/>

Navegación y rutas

Sistema de navegación

- El sistema de navegación ha sido modificado varias veces hasta quedar definido en su versión final (>= **RC5**).
 - El sistema de la v.2, fue el causante del salto de versión a la 4
- La estructura final se apoya igualmente en el modelo SPA, pero el elemento que contiene las partes variables se denomina (por convención):
<router-outlet></router-outlet>
- Existen un objeto que controla el componente a visualizar dentro de ese elemento: **app.routing.ts** (gestor de URLs)
- En el caso de la navegación interna se utilizan *bindings* especiales designados con
[routerLink] = ["ruta a navegar"].
- Finalmente, también podemos navegar desde el código utilizando el objeto **Router**.

Modelo de rutas

- Para configurar el sistema de rutas debemos de:
 - Trasladar la funcionalidad actual de **app.component** a otro componente, dejando solamente el sistema de rutas.
 - Crear el nuevo componente separado
 - Definir el elemento base en **index.html**
 - Registrar los proveedores de rutas.
 - Enlazar el proveedor de rutas con la aplicación.
- Por tanto la navegación se define en el sentido *Path => Componente* (navegación lógica) en lugar de utilizar el modelo *Path => Recurso.html* (Navegación física).
 - Este modelo es el típico de las aplicaciones MVC/SPA

Cambios en la página maestra

- El primer paso lo tenemos que realizar en *index.html*:
 - Se debe de declarar qué ruta entendemos por ruta base, mediante el elemento `<base href="/">` en la cabecera del documento.
 - A continuación, modificamos ligeramente el elemento `<gestion-productos>`, para que esté empotrado en otro elemento (como un `<section> </section>`), que pueda servir de contenedor para el formato visual con clases Bootstrap
 - No obstante, ten presente que esto es solo por la parte visual, lo único realmente imprescindible aquí (en lo funcional) es el elemento `<base>`
- Al final tendremos algo como:

```
<!-- 3. Muestra la aplicación -->
<body class="container-fluid">
  <section class="row">
    <gestion-productos class="col-sm-12 col-md-12 col-lg-12">
      Cargando...
    </gestion-productos>
  <section>
</body>
```

Creación del menú y del “placeholder”

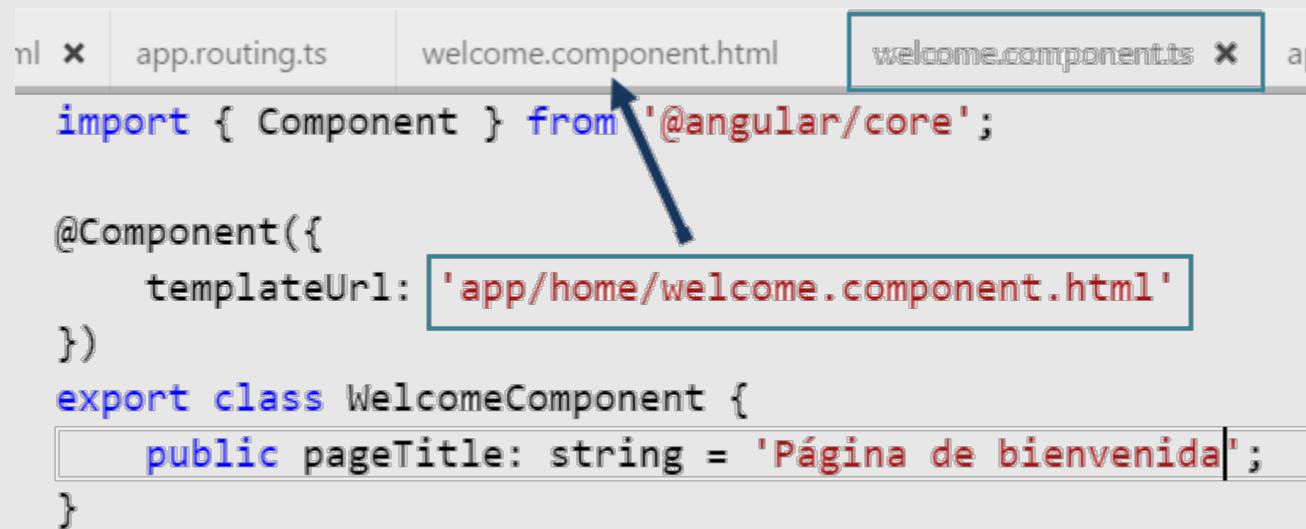
- El segundo paso consiste en modificar la plantilla de *app.component*, para que genere un menú con un par de opciones e incluya el elemento que va a ser el destino de las rutas: **<router-outlet>**

```
@Component({  
  selector: 'gestion-productos',  
  template: `  
    <h1>{{tituloPagina}}</h1>  
    <a class="btn btn-info" [routerLink] = ["'"]>Inicio</a>  
    <a class="btn btn-info" [routerLink] = ["['listado']"]>Lista Productos</a>  
    <br/><br/>  
    <router-outlet></router-outlet>  
  `,  
  providers: [ ServicioProductos ]  
})
```

- En este momento, hemos sustituido el elemento **<listado-productos>** anterior por el nuevo *placeholder* de rutas definido con ese nombre por convención dentro de Angular 2.
 - Más adelante entraremos en el significado de los atributos [routerLink]

El componente de rutas

- El tercer paso, es crear otro componente (componente de bienvenida), de forma que podamos tener un par de ubicaciones distintas para navegar (ahora solo tenemos una).
- Así que, en otra carpeta (home) creamos un *Welcome.component* con sus dos partes: TypeScript y HTML.
- La parte que interesa, TypeScript puede ser como cualquier otro componente:



The screenshot shows a code editor with several tabs at the top: 'nl x', 'app.routing.ts', 'welcome.component.html', and 'welcome.component.ts x'. The 'welcome.component.ts' tab is active, highlighted with a blue border. A red arrow points from the text 'templateUrl:' in the code below to the tab bar, specifically pointing at the 'welcome.component.ts' tab.

```
import { Component } from '@angular/core';

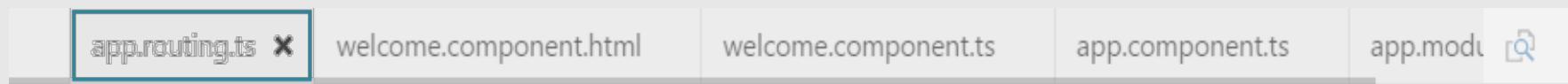
@Component({
  templateUrl: 'app/home/welcome.component.html'
})
export class WelcomeComponent {
  public pageTitle: string = 'Página de bienvenida';
}
```

El componente de rutas

- El cuarto paso, es crear el componente que se va a hacer cargo de las rutas (de todas las rutas en principio).
 - Como es global a la aplicación, lo declaramos en el mismo nivel que **app.module** y **app.component**.
 - El componente, debe de importar (para hacer referencia a ellos), todos los elementos que vayan a ser destino de una de sus rutas.
 - Además utiliza dos componentes separados del "core" de Angular (**Router** y **RouterModule**).
- El objetivo es crear una constante de tipo **Routes** (formato de array), que defina en sus elementos tantas parejas { **path / component** } como rutas queramos para nuestra aplicación.
 - Esta constante puede tener el nombre que queramos
- Por último esa constante debe ser expuesta al exterior para que el componente principal (**app.module**) pueda registrarla.

El componente de rutas

- El código resultante no es muy complicado, solo hay que entender el propósito de cada elemento:



```
app.routing.ts x welcome.component.html welcome.component.ts app.component.ts app.modu 🔎  
import { Routes, RouterModule } from '@angular/router';  
import { ListaProductosComponent } from './productos/lista-productos.component';  
import { WelcomeComponent } from './home/welcome.component';  
  
const APP_ROUTES: Routes = [  
  { path: "", component: WelcomeComponent},  
  { path: "listado", component: ListaProductosComponent}  
]  
export const Routing = RouterModule.forRoot(APP_ROUTES);
```

- El método ***forRoot*** de ***RouterModules*** asocia las rutas definidas con la constante de tipo **Routes** que ahora podremos usar para el registro

Registro de rutas

- Así que en el módulo principal añadiremos las referencias necesarias a este elemento:
- En la parte de declaraciones, ahora deben aparecer las referencias a **Router** y al nuevo componente **WelcomeComponent**.

The screenshot shows the `app.module.ts` file in an IDE. The file contains the following code:

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/http';
import { Routing } from './app.routing';
import 'rxjs/Rx';

import { AppComponent }  from './app.component';
import { WelcomeComponent } from './home/welcome.component';
```

A blue double-headed arrow points from the word `Routing` in the first import statement to the `app.routing.ts` tab at the top. Another blue double-headed arrow points from the `WelcomeComponent` import statement to the `app.component.ts` tab at the top.

Registro de rutas

- En el decorador @NgModule, incluimos el nuevo componente en el apartado ***declarations***, y la referencia al **Router** dentro de los **imports** (componentes importados).

```
@NgModule({  
  imports:      [ BrowserModule, RouterModule ],  
  declarations: [ AppComponent, LoginComponent, WelcomeComponent, FiltroProductosPipe ],  
  bootstrap:    [ AppComponent ],  
})  
export class AppModule { }
```

Observa que Routing es la constante definida y exportada

Este componente se exporta para ser utilizado en las rutas

- De esa forma las rutas definidas en la constante **Routing** quedan disponibles para cualquier elemento manejado por el módulo principal.

Resultados finales

- Además, si nos fijamos en la definición de las rutas, estábamos usando como atributo **path** las cadenas "" y "**listado**" respectivamente.
- Se entiende que "" (cadena vacía) hace referencia a la raíz del sitio, y que tenemos asociada con **WelcomeComponent**.
- Y "**listado**" es un fragmento a resolver que hemos asociado **con un componente** (en este caso, con **ListadoComponent**).
- Por tanto, si nuestro sitio está configurado por defecto para comenzar por la URL **localhost:3000**, deberemos ver esa salida de forma inicial



Enlaces a rutas en la vista

- Pero, para poder ver el listado de productos necesitaríamos teclear manualmente en la URL el fragmento "*listados*".
- Por esa razón usamos los atributos **[routerLink]**, que asocian el valor indicado con las rutas definidas.

```
@Component({  
  selector: 'gestion-prod',  
  template: `<h1>{{tituloP}}</h1>  
  <a class="btn btn-info" [routerLink] = "[ '' ]">Inicio</a>  
  <a class="btn btn-info" [routerLink] = "[ 'listado' ]">Lista Productos</a>  
  <br/><br/>  
  <router-outlet></router-outlet>  
`,  
  providers: [ ServicioProductos ]  
})
```

- Con esta asignación al seleccionar el enlace "listado" navegaremos a nuestro listado básico.

Rutas y navegación

Demo

Uso de bibliotecas externas

- El uso de bibliotecas gráficas externas con Angular 5 no está recomendado.
- **jQuery**: Preferible modificar el DOM utilizando plantillas u otros recursos propios de Angular 5.
- **Bootstrap avanzado** (su parte JavaScript): se basa en jQuery, así que es preferible utilizar *ng2-Bootstrap*, que son componentes Bootstrap, pero están adaptados a NG2.
- **Otras bibliotecas**: Preferible utilizar las que están diseñadas para Angular (cada vez aparecen más)
 - Se evitan problemas de rendimiento y de complejidad en la estructura de la aplicación.

Ecosistema de Angular 5

- **Angular 5 – Electron:** aplicaciones de escritorio multi-plataforma.
 - Ejemplo: Visual Studio Code
- **Ionic:** Aplicaciones híbridas para móviles.
- **NativeScript:** Aplicaciones móviles con UI nativa para Angular 5.
- **Angular 5 – Meteor:** *Framework* JavaScript/TypeScript para desarrollar aplicaciones web interactivas (cliente/servidor con comunicación vía Web Sockets)
- **AngularFire 2:** Versión actualizada de *AngularFire* (servicio REST de datos JSON en la nube, gestionado por Google)
 - Dispone de suscripciones gratuitas de hasta 2Gb (máx. 10 BB.DD.)

Referencias

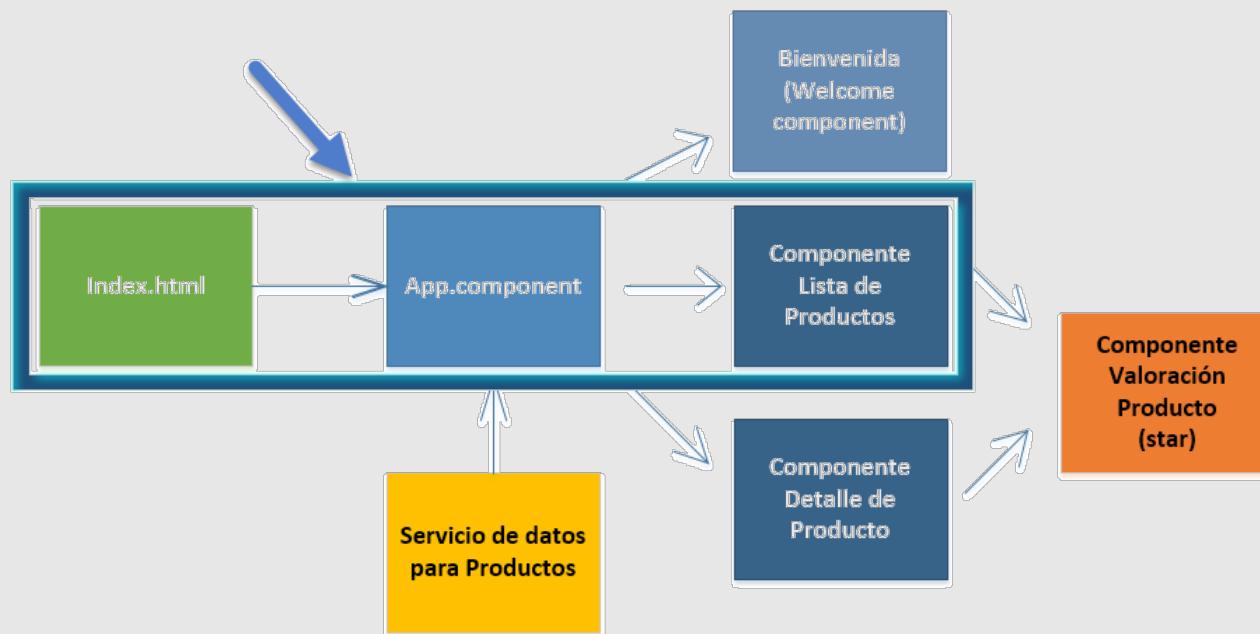
- Arquitectura y documentación oficial de Angular 2
 - Sitio oficial de Angular:
 - <https://angularjs.org/>
 - Documentación oficial:
 - <https://drive.google.com/drive/folders/0BxgtL8yFJbacQmpCc1NMV3d5dnM>
 - Última documentación sobre Angular:
 - <https://angular.io/docs/ts/latest/>

Data Binding y Pipes

Interactividad entre el DOM y los componentes

- Ya tenemos una lista de productos. Pero no existe ningún tipo de interactividad entre el DOM y los componentes
- Vamos a añadir esos factores para incorporar eventos y otros aspectos.

Arquitectura de la aplicación de
Productos



Interactividad entre el DOM y los componentes

- Ya tenemos una lista de productos. Pero no existe ningún tipo de interactividad entre el DOM y los componentes
- Vamos a modificar los datos, de forma que incluyan una imagen y podamos ver el funcionamiento de los gráficos y dar cobertura al botón correspondiente.
- La manera en que el *databinding* se ha modificado en esta versión ha sufrido dos modificaciones importantes respecto a versiones 1.x (no así respecto a la 4)
- Por un lado se mantiene la posibilidad de usar la sintaxis *moustache* como antes (mecanismo de interpolación), y por otro, se puede expresar un *databinding* poniendo entre paréntesis la propiedad a enlazar y entre comillas el dato de destino.
- Por ejemplo para enlazar con un gráfico podemos seguir utilizando interpolación:

```

```

Data Binding

- Pero también disponemos de una alternativa propia de Angular 2, en la que la propiedad a enlazar tiene este aspecto:



- De modo que la sintaxis anterior quedaría de esta forma:

```
<img [src]="producto.imageUrl" style="width:50px"/>
```

- De modo que si modificamos los datos del componente para que incluya un par de referencias a imágenes, tendremos que tener la salida modificada incluyendo los gráficos.
- Aunque se prefiere el **databinding**, si tenemos que usar una expresión compuesta, la interpolación puede ser la solución más adecuada:

```
<img src='http://openclipart.org/{{product.imageUrl}}'>
```

Data Binding

- También podemos establecer un mecanismo de **binding** sobre propiedades de una clase.
- En este caso, la anchura y el margen de la propiedad gráfica de cada producto, se prestan bien para ser expresadas de esa forma:

```
export class ListaProductosComponent {  
    imageWidth: number = 50;  
    imageMargin: number = 3;  
    productos: any[] = [  
        {  
            "Producto": "Leaf Rake",  
            "Codigo": "GDN-0011",  
            "Disponible": "March 19, 2016",  
            "Precio": 19.95,  
            "Valoracion": 3.,  
            "imageUrl": "http://openclipart.org/  
    },  
    -
```

Data Binding

- Y el correspondiente código fuente de la plantilla, quedaría así:

```
<img [src]="producto.imageUrl"
      [title]="producto.Producto"
      [style.width.px]="imageWidth"
      [style.margin.px]="imageMargin" >
```

- Observa que para los estilos, no ponemos el nombre del objeto, por que son propiedades de la clase con al que enlazamos.
- El mecanismo de **Databinding** (igual que la interpolación) va en un solo sentido (**one-way**).
- Ahora nuestra demo tiene que aparecer correctamente, aunque el botón de ocultar las imágenes, (obviamente) no funciona, porque tenemos que incluir el manejador de eventos correspondiente.

Data Binding

- En la demo se incluyen las dos versiones, para comprobar que son equivalentes en este caso.

Demo de Gestión de Productos

Lista de productos

Filtrar por:

Filtrado por:

Mostrar imagen



Garden Cart



Binding de [src]

Producto	Codigo	Disponible	Precio	Valoracion
Leaf Rake	GDN-0011	March 19, 2016	19.95	3
Garden Cart	GDN-0023	March 21, 2016	32.99	4.2

Binding de [title]

Data Binding

Data Binding de propiedades

Vinculación con eventos

- En la parte de los eventos la sintaxis es similar, pero el evento correspondiente se encierra entre paréntesis. De modo que modificaremos el código del botón de la siguiente forma:

```
<button class="btn btn-warning"
        (click)="mostrarImagen()"
        Mostrar imagen
    </button>
```

- A continuación, crearemos una propiedad y un método en la clase del componente:

```
imageVisible: boolean = false;  
  
mostrarImagen(): void {  
    this.imageVisible = !this.imageVisible;  
}
```

- Por último estableceremos que la imagen es visible solo condicionalmente:

```
<img *ngIf="imageVisible"  
      [src]="producto.imageUrl"
```

Data Binding

Vinculación con eventos

Expresiones de interpolación

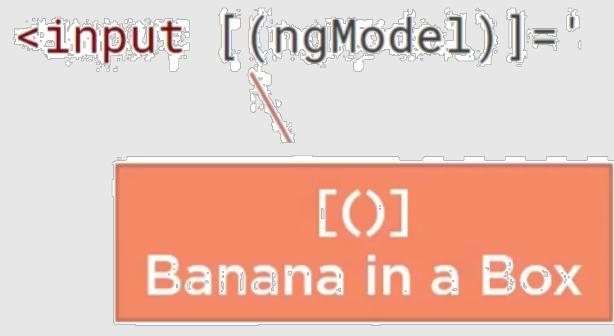
- Nos queda hacer que el texto del botón también se modifique cuando pulsemos sobre él, cambiando el texto de "Mostrar Imagen" a "Ocultar Imagen".
- Esto es sencillo: cambiamos el texto del botón por una interpolación:

```
<button class="btn btn-warning"
        (click)="mostrarImagen()">
    {{imageVisible ? "Mostrar" : "Ocultar"}} imágenes
</button>
```

- Recordemos que la interpolación no solo lee datos, sino que evalúa expresiones.

Binding bidireccional (two-way binding)

- Si queremos que la entrada de la caja de texto muestre el criterio de ordenación, tenemos que hacer que, al introducir una expresión de filtro, ésta aparezca en la etiqueta inferior.
- Esto se realizaba con la directiva **ng-model** en la versión anterior, y en ésta es muy similar pero se usa la sintaxis "*banana in a box*".
- El nombre es debido a la sintaxis que se utiliza:



- Los corchetes externos indican enlace a propiedad (la que se indique después del signo igual), mientras los paréntesis internos indican evento (cualquier entrada que cambie los datos a los que apunta)

Binding bidireccional (two-way binding)

- Ahora bien, si queremos este tipo de funcionalidad en elementos de entrada, tenemos que modificar nuestro módulo, para importar esta operativa desde el bloque "*FormsModule*"
- Este bloque contiene esa y otras funcionalidades propias de los mecanismos de entrada y su paso al modelo de datos.
- Por tanto, deberemos volver a la función de nuestro módulo, y modificarlo de la siguiente forma:

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';|<----->
import { AppComponent } from './app.component';
import { ListaProductosComponent } from './productos/lista

@NgModule({
  imports:      [ BrowserModule, FormsModule ],
  declarations: [ AppComponent, ListaProductosComponent ],
  bootstrap:    [ AppComponent ]
})
```



Binding bidireccional (two-way binding)

- Ahora, podremos cambiar nuestro mecanismo de filtro, que contendrá en la plantilla esta sintaxis (**ngModel** se define en **FormsModule**):

```
<div class="row">
    <div class="col-md-2">Filtrar por:</div>
    <div class="col-md-4">
        <input type="text"
            [(ngModel)]='filtro' >
    </div>
</div>
<div class="row">
    <div class="col-md-6">
        <h3>Filtrado por: {{filtro}}</h3>
    </div>
</div>
```

- Y definimos la propiedad '**filtro**' en nuestra clase:

```
filtro: string = "cart";
```

Binding bidireccional (two-way binding)

- Con lo que –nuevamente- tendremos una pantalla inicial que muestra un filtro predeterminado en la caja de texto, y el "eco" de esa propiedad en el elemento **<h3>** ("Filtrado por:") de la plantilla.
- Si cambias el valor del **input** de entrada, verás como se hace eco automáticamente en la propiedad:

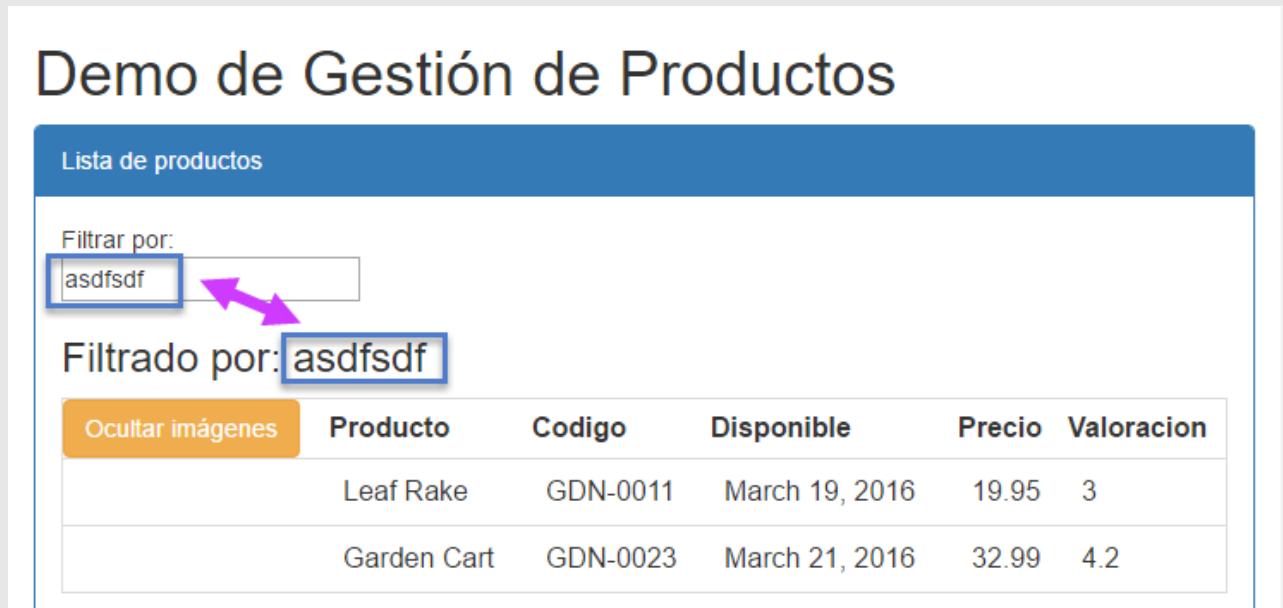
Demo de Gestión de Productos

Lista de productos

Filtrar por: asdfsdf

Filtrado por: asdfsdf

Ocultar imágenes	Producto	Código	Disponible	Precio	Valoración
	Leaf Rake	GDN-0011	March 19, 2016	19.95	3
	Garden Cart	GDN-0023	March 21, 2016	32.99	4.2



Pipes (filtros de salida)

- Otro aspecto similar al que existía en Angular 2 son los filtros de salida, antes llamados *filters*, y ahora *pipes* por el símbolo utilizado en su sintaxis.
- Se escriben a continuación de una expresión y modifican la manera es que ésta se interpreta en el DOM, sin modificar en ningún momento el dato de entrada.

```
 {{ product.productCode | lowercase }}  
  
 <img [src] ='product.imageUrl'  
       [title] ='product.productName | uppercase'>  
  
 {{ product.price | currency | lowercase }}  
  
 {{ product.price | currency:'USD':true:'1.2-2' }}
```



Pipes (filtros de salida)

- Existen un buen número de filtros predeterminados (*Built-in pipes*), que son aplicables a algunos tipos de datos usados más comúnmente.
 - *DatePipe, UppercasePipe, LowercasePipe, CurrencyPipe, PercentPipe, DecimalPipe, CurrencyPipe, JsonPipe, etc.*
- En ésta versión no se dispone de filtros **OrderBy** o **Filter**, (típicos en AngularJS) por razones de rendimiento y comportamiento en procesos de "*bundling & minifying*".
- A su vez, las "pipes" pueden encadenarse, en forma similar a como lo hace el "*middleware*" de un servidor.
 - Cada componente (**pipe**) recibe una entrada, produce una transformación y elabora con ello un dato de salida.

Pipes (filtros de salida)

- Un tipo especial de pipe es la JsonPipe, que se usa habitualmente en desarrollo, para presentar con facilidad el contenido de un bloque JSON obtenido de una llamada o lectura de datos.
- Angular distingue dos tipos de "**pipes**": puras e impuras.
 - Simplificando, una **pipe** impura se ejecuta con cada modificación (incluso parcial) de un elemento de entrada
 - Un pipe pura se ejecuta solo cuando se produce un "cambio puro". Y se entiende por tal un cambio en un valor de entrada primitivo (**String**, **Number**, **Boolean**, **Symbol**) o una referencia de objeto cambiada (**Date**, **Array**, **Function**, **Object**).
- Además su construcción no es complicada (lo veremos en un próximo módulo), con lo cual podemos realizar transformaciones de salida de cualquier clase, sin modificación de los datos de entrada.

Pipes (filtros de salida)

- Podemos probar algunos aspectos de *pipes* en nuestro código, cambiando la salida del código de los productos para que aparezca en minúsculas (*lowercase*), y formateando el precio para que adopte el formato de moneda (*)

```
<td>{{producto.Producto}}</td>
<td>{{producto.Codigo | lowercase }}</td>
<td>{{producto.Disponible}}</td>
<td>{{producto.Precio | currency:'EUR':true:'1.2-2' }}</td>
<td>{{producto.Valoracion}}</td>
```

- En el caso del formato de moneda, el segundo argumento, (a diferencia de lo que sucedía en la versión anterior), es el código ISO 4217:
 - https://en.wikipedia.org/wiki/ISO_4217
- El tercer argumento (*true*) indica que queremos mostrar decimales, y el cuarto significa que queremos al menos una cifra entera y al menos 2 decimales con un máximo de 2 decimales.

Pipes (filtros de salida)

- Al igual que sucedía con los filtros, el usuario puede crear sus propios filtros, que no son sino funciones que se registran como *pipes*.
- Otro aspecto a tener en cuenta es que la salida predeterminada de Angular 2+ (y la anterior) es en inglés de EE.UU.
- Para cambiar el comportamiento predeterminado, deberemos incluir el "*locale*" que corresponda al país que se desee (numerónimos *i18n*).
- La salida reflejará los cambios, como es de esperar:

Demo de Gestión de Productos

Lista de productos

Filtrar por:
cart

Filtrado por: cart

Ocultar imágenes	Producto	Código	Disponible	Precio	Valoración
	Leaf Rake	gdn-0011	March 19, 2016	€19.95	3
	Garden Cart	gdn-0023	March 21, 2016	€32.99	4.2

Pipes

Demo

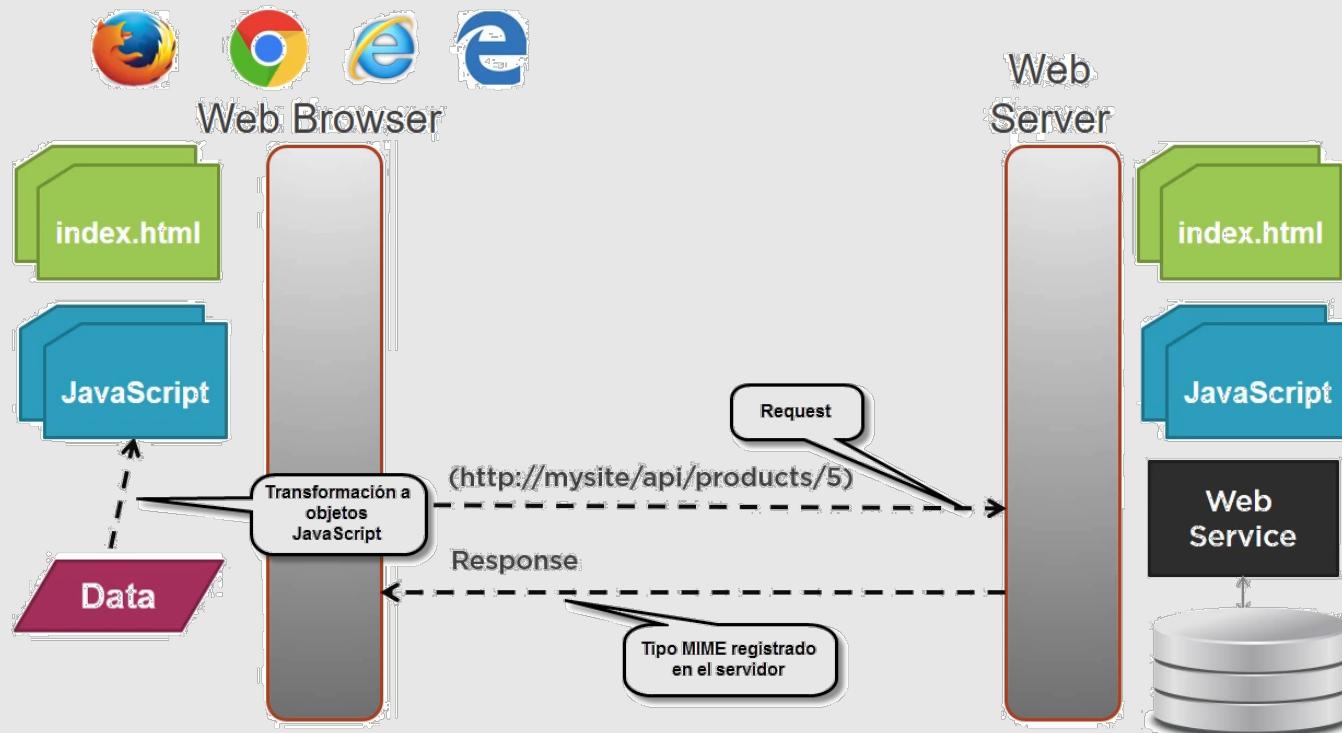
Referencias

- Arquitectura y documentación oficial de Angular
 - Sitio oficial de Angular:
 - <https://angularjs.org/>
 - Documentación oficial:
 - <https://drive.google.com/drive/folders/0BxgtL8yFJbacQmpCc1NMV3d5dnM>
 - Última documentación sobre Angular:
 - <https://angular.io/docs/ts/latest/>

HTTP

Objetivos del módulo

- Este módulo trata sobre los mecanismos de acceso a datos en Angular 5
- Especialmente, los relativos al uso del protocolo HTTP y las colecciones *observable*.
- Recordemos el diagrama básico de comunicación cliente/servidor



Observables

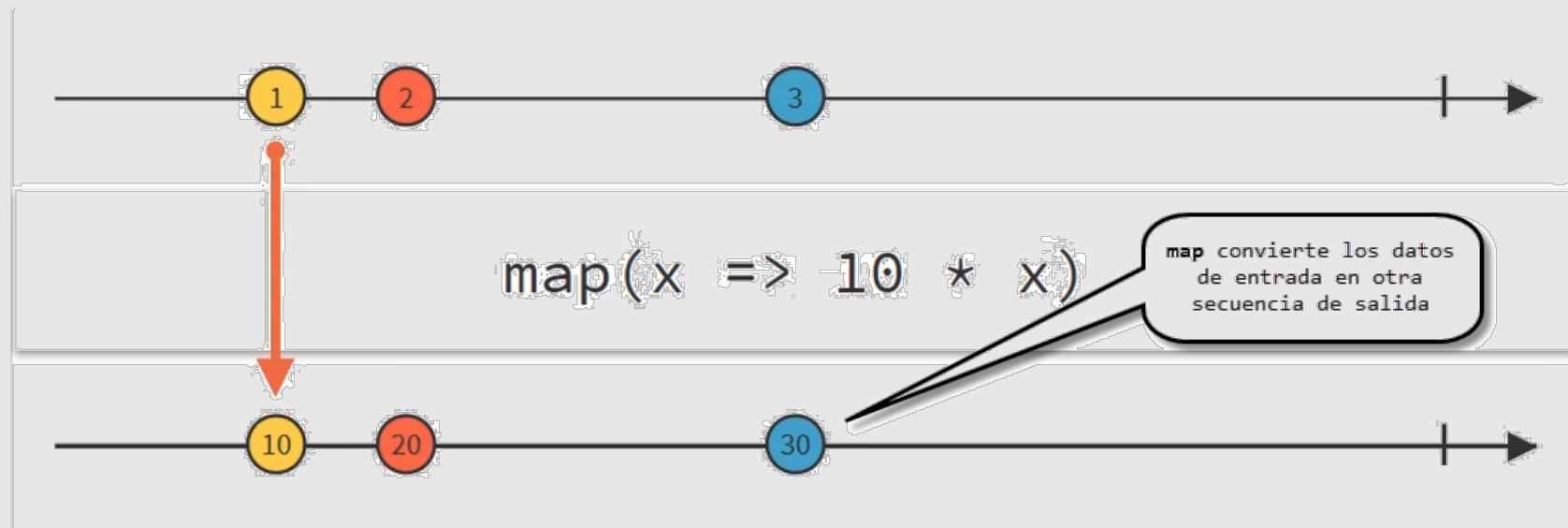
- Entendemos por *observable* un array que llega a un destino en bloques (*marbles*) de forma asíncrona.



- Son fundamentales en peticiones de servicios para unir de forma coherente datos que llegan de manera muy dispar.
- Son una característica propuesta para ES2016, pero para poderlo utilizar ahora, Angular usa la librería *RxJS* (*Reactive Extensions*).
- Angular 2+ también los utiliza en sus sistema de eventos.

Observables

- El funcionamiento de los observables queda explicado en el siguiente diagrama, donde vemos el uso de la sentencia **map(x => x * 10)**, para convertir una secuencia observable en otra distinta.



- Existe una cierta similitud con las *Promises*

Observables: diferencias respecto a las Promises

➤ Hay diferencias notables entre ambos:

➤ Promises

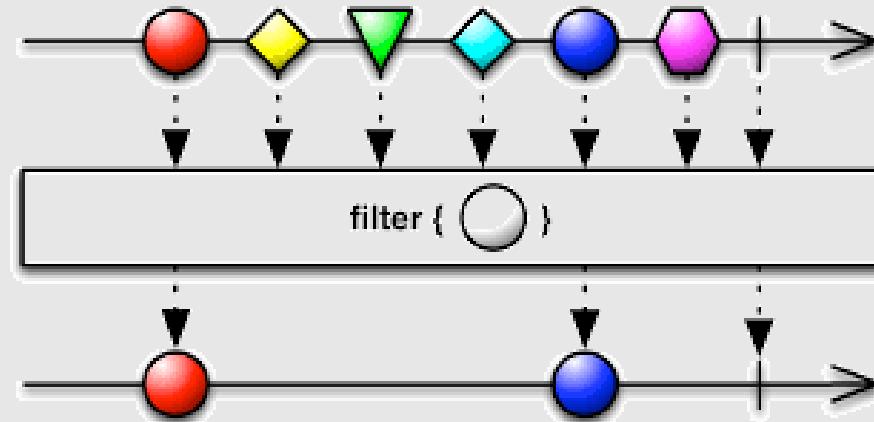
- Devuelven un único valor
- No son cancelables
- También se pueden usar en Angular 5 (existe algún ejemplo de uso en la documentación oficial).

➤ Observables

- Devuelven valores múltiples en el tiempo.
- Pueden cancelarse
- Soportan operadores modernos como **map**, **reduce**, **filter**, **retry**, etc.
- Utilizan una arquitectura parecida a los Web Sockets.

Observables: Configuración

- Para utilizar observables con HTTP se requieren algunos pasos previos:
 - Incluir el script de Http para Angular 2+
 - Registrar **HTTP_PROVIDERS**
 - Importar Reactive Extensiones (**RxJS**)
- Esto nos permite utilizar otro conjunto de recursos como los filtros:



Observables: Configuración

- Las secuencias de datos pueden tomar diversas formas:
 - **Streams** provenientes de un servidor de back-end o servicio web.
 - Un conjunto de notificaciones de sistema
 - Series de eventos, tales como las entradas del usuario
- Nuestro código se puede suscribir para recibir esas notificaciones asíncronas, según van llegando los datos.
- De esa forma puede "reaccionar" cuando llegan los datos, o cuando se le notifica un error.
- Las "*Reactive extensions*" proporcionan, además, otras funcionalidades.

Observables: Configuración

- La forma en que se expone la funcionalidad es a través de una constante, que aglutina unos cuantos servicios relacionados con HTTP.
- Para disponer de esa funcionalidad en diversas partes de la aplicación, importamos el módulo **HttpModule** en el componente principal, y también todo lo incluido en las "**reactive extensions**".
 - ¡Ojo! En versiones anteriores a la final, se utilizaba una **constante** **HTTP_PROVIDERS**, que ya no funciona en la versión final).
- En nuestro caso, deberemos de importar esos elementos en **app.module.ts**, y declarar **HttpModule** en la lista de **imports**:

```
import { HttpModule } from '@angular/http';
import 'rxjs/Rx';

@NgModule({
  imports:      [ BrowserModule, FormsModule, HttpModule ],
  declarations: [ AppComponent, ListaProductosComponent,
                 FiltroProductosPipe, StarComponent ],
  bootstrap:   [ AppComponent ],
})
```

Observables: Programación

- De esta forma se expone su funcionalidad en todos los módulos dependientes.
- Por otro lado, nuestro componente no sufre modificaciones, solamente cambiaremos el servicio para que lea los datos y le añadimos una rutina de tratamiento de errores para ayuda en la depuración.
- Así pues, el servicio queda de la siguiente forma:
 - En la parte de declaraciones, ahora declaramos los componentes **Http** y **Response** del módulo http y **Observable** de las RX:

```
import { Injectable } from '@angular/core';
import { IProducto } from './producto';
import { Http, Response } from '@angular/http';
import { Observable } from 'rxjs/Observable' ;
```

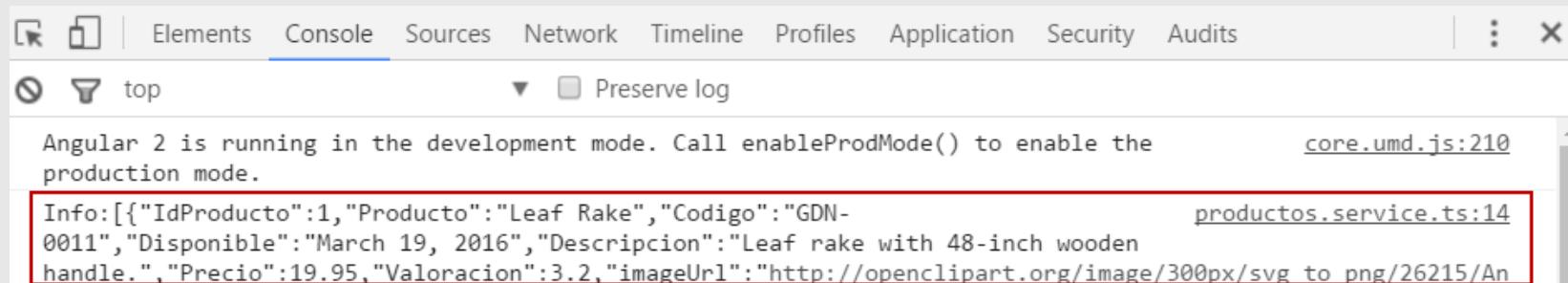
Observables: Programación

- La clase es la que más modificaciones tendrá.
- Por un lado declaramos una variable local para la Url de los productos.
- Además, tenemos que declarar (por inyección de dependencia) el servicio Http, y para la lectura eliminamos los datos y los sustituimos por la llamada correspondiente:

```
@Injectable()  
export class ServicioProductos {  
    private _productosUrl = 'app/productos/Productos.json';  
    constructor(private _http: Http){}  
  
    leerProductos(): Observable<IProducto[]> {  
        return this._http.get(this._productosUrl).  
            map( (response: Response) => <IProducto[]>response.json() ).  
            do(datos => console.log('Info: ' + JSON.stringify(datos))).  
            catch(this.manejarError);  
    }  
}
```

Observables: Programación

- Observa que la llamada devuelve un **Observable** genérico, y – cuando los datos se reciben- se recogen el en objeto **response** mediante **map**, después de haber convertido el "stream" a **json**.
- Además utilizamos el método **do()** para realizar otra labor adicional, presentar los datos por consola, de forma que los podamos ver allí directamente, además de en la IU:



The screenshot shows the Chrome DevTools console interface. The tabs at the top are Elements, Console (which is selected), Sources, Network, Timeline, Profiles, Application, Security, and Audits. Below the tabs, there are two dropdown menus: one for 'Level' set to 'top' and another for 'Preserve log' which is checked. The main area displays the following log message:
Angular 2 is running in the development mode. Call enableProdMode() to enable the production mode. core.umd.js:210
Info:[{"IdProducto":1,"Producto":"Leaf Rake","Codigo":"GDN-0011","Disponible":"March 19, 2016","Descripcion":"Leaf rake with 48-inch wooden handle.", "Precio":19.95,"Valoracion":3.2,"imageUrl":"http://openclipart.org/image/300px/svg_to_png/26215/An"}productos.service.ts:14

- Finalmente, creamos una rutina de tratamiento de error (**manejarError**), que presenta los errores potenciales por consola y también lanza una excepción.

Observables: Programación

- La rutina de error es sencilla:

```
private manejarError(error: Response) {  
    console.error(error);  
    return Observable.throw(error.json().error || "Error del servidor");  
}
```

- Si todo ha ido bien, deberemos de ver finalmente los cinco productos del fichero json en el listado final.

Producto	Código	Disponible	Precio	Valoración
Leaf Rake	gdn-0011	March 19, 2016	€19.95	★★★
Garden Cart	gdn-0023	March 18, 2016	€32.99	★★★★
Hammer	tbx-0048	May 21, 2016	€8.90	★★★★★
Saw	tbx-0022	May 15, 2016	€11.55	★★★★★
Video Game Controller	gmg-0042	October 15, 2015	€35.95	★★★★★

HTTP

Demo HTTP y Observables

Referencias

- Arquitectura y documentación oficial de Angular
 - Sitio oficial de Angular:
 - <https://angularjs.org/>
 - Documentación oficial:
 - <https://drive.google.com/drive/folders/0BxgtL8yFJbacQmpCc1NMV3d5dnM>
 - Última documentación sobre Angular
 - <https://angular.io/docs/ts/latest/>

We Know IT

Barcelona

C. dels Almogàvers 123
08018 Barcelona
Tel. +34 933 041 720
Fax. +34 933 041 722

Madrid

Plaza Carlos Trías Bertrán,
7 Edificio Sollube, 1^a Planta
28020 Madrid
Tel. +34 914 427 703
Fax +34 914 427 707

Bilbao

C/ San Vicente, n° 8
Edificio Albia I,
6^a planta - dpto. 8
48001 – Bilbao
Tel. +34 944 354 982

www.netmind.es

Desarrollo Web con Angular 5 (front) y API REST con NodeJS (back)

Índice

1. Herramientas
2. Angular 5 – Front
- 3. Angular Material**
4. NodeJS – Back
5. Testing



Angular Material

Características

Componentes

Instalación y uso

Angular Material

- Componentes visuales para Angular
- Características:
 - Componentes modernos y comprensibles
 - Válidos para Web, móvil y escritorio
 - Rápidos y consistentes
 - Versátiles mediante temas
 - Optimizados para su uso con Angular

Componentes

Form Controls

Navigation

Layout

Buttons & Indicators

Popups & Modals

Data table

Form Controls

Autocomplete



Checkbox



Datepicker



Form field

Input



Radio button



Select



Slider



Slide toggle



Navigation Controls

Menu



Sidenav



Toolbar



Layout Controls

Card



Divider

Expansion Panel

Grid list



List



Stepper

Tabs



Tree

Buttons & Indicators Controls

Button



Button toggle



Badge

Chips



Icon



Progress spinner



Progress bar



Popups & Modals Controls

Bottom Sheet

Dialog



Snackbar



Tooltip



Data table Controls

Paginator

Sort header

Table

Instalación y uso

➤ Pasos para instalar Angular Material

➤ Instalación independiente

NPM

```
npm install --save @angular/material @angular/cdk @angular/animations
```

Yarn

```
yarn add @angular/material @angular/cdk @angular/animations
```

➤ Añadir Angular Material a proyecto Angular

```
ng add @angular/material
```

Instalación y uso

➤ Configurar animaciones

```
import {BrowserAnimationsModule} from '@angular/platform-browser/animations';

@NgModule({
  ...
  imports: [BrowserAnimationsModule],
  ...
})
export class PizzaPartyAppModule { }
```

Instalación y uso

➤ Importar componentes

```
import {MatButtonModule, MatCheckboxModule} from '@angular/material';

@NgModule({
  ...
  imports: [MatButtonModule, MatCheckboxModule],
  ...
})
export class PizzaPartyAppModule { }
```

Instalación y uso

➤ Incluir un tema

```
@import "~@angular/material/prebuilt-themes/indigo-pink.css";
```

➤ Instalar el soporte de gestos con HammerJS

NPM

```
npm install --save hammerjs
```

Yarn

```
yarn add hammerjs
```

```
import 'hammerjs';
```

Instalación y uso

- Añadir iconos Material (opcional)

```
<link href="https://fonts.googleapis.com/icon?family=Material+Icons"  
      rel="stylesheet">
```

- Ejemplos disponibles en:
 - <https://material.angular.io/>

We Know IT

Barcelona

C. dels Almogàvers 123
08018 Barcelona
Tel. +34 933 041 720
Fax. +34 933 041 722

Madrid

Plaza Carlos Trías Bertrán,
7 Edificio Sollube, 1^a Planta
28020 Madrid
Tel. +34 914 427 703
Fax +34 914 427 707

Bilbao

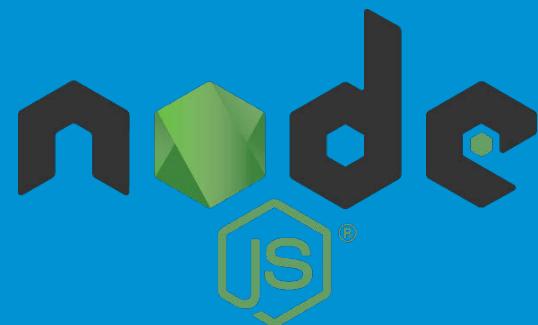
C/ San Vicente, n^o 8
Edificio Albia I,
6^a planta - dpto. 8
48001 – Bilbao
Tel. +34 944 354 982

www.netmind.es

Desarrollo Web con Angular 5 (front) y API REST con NodeJS (back)

Índice

1. Herramientas
2. Angular 5 – Front
3. Angular Material
- 4. NodeJS – Back**
5. Testing



NodeJS - Back

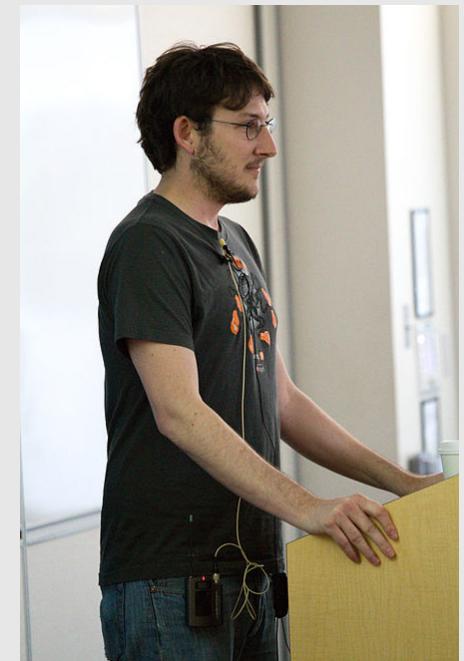
Primeros pasos con NodeJS

Instalación y uso de Express

Servicios REST y protocolo HTTP

NodeJS

- Entorno de ejecución JavaScript
- Multi-plataforma
- Open Source
- Ejecuta el código JavaScript fuera de un navegador
- Creado por **Ryan Dahl** en **2009**



Historial de versiones

Release	Nombre	Fecha	LTS	Fin soporte
v0.10.x		11/03/2013	End-of-life	31/10/2016
v0.12.x		06/02/2015	End-of-life	31/12/2016
4.x	Argon	08/09/2015	End-of-life	30/04/2018
5.x		29/10/2015	No LTS	30/06/2016
6.x	Boron	26/04/2016	Maintenance	Abril 2019
7.x		25/10/2016	No LTS	30/06/2017
8.x	Carbon	30/05/2017	Active	Diciembre 2019
9.x		01/10/2017	No LTS	Junio 2018
10.x	Dubnium	24/04/2018	Pending	Abril 2021
11.x		23/10/2018	No LTS	Junio 2019

Instalación

- Instalación en:
 - Windows
 - Linux
 - Mac
- Disponible en:
 - <https://nodejs.org/en/download/current/>
- Dos tipos de versiones:
 - LTS (Long Time Support) ➔ Recomendado
 - Current ➔ Últimas características disponibles

Instalación – comprobación

- Versión de **NodeJS** (node -v)

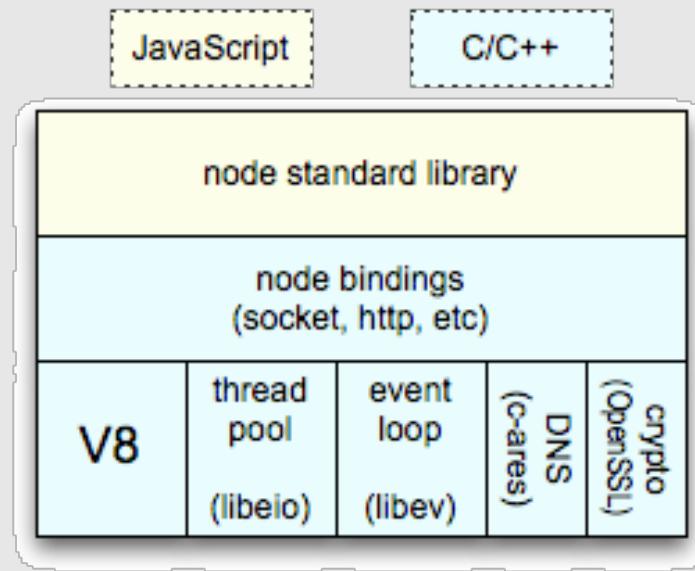
```
C:\Users\Angel>node -v  
v8.9.4
```

- Versión de Node Package Manager (**NPM**) (npm -v)

```
C:\Users\Angel>npm -v  
5.6.0
```

NodeJS - Arquitectura

- ¿Cuál es la gran diferencia en este comportamiento?
 - Sobre todo que Apache utiliza una hebra de ejecución por conexión.
 - Para el caso de IIS, existen técnicas de limitación de hebras, pero la arquitectura es similar: [https://msdn.microsoft.com/en-us/library/ee377050\(v=bts.70\).aspx](https://msdn.microsoft.com/en-us/library/ee377050(v=bts.70).aspx)
- La solución propuesta por **Dahl**, ofrece un tiempo de respuesta mejorado notablemente (y mucho más estable en concurrencia), al tiempo que permite una configuración total del comportamiento.

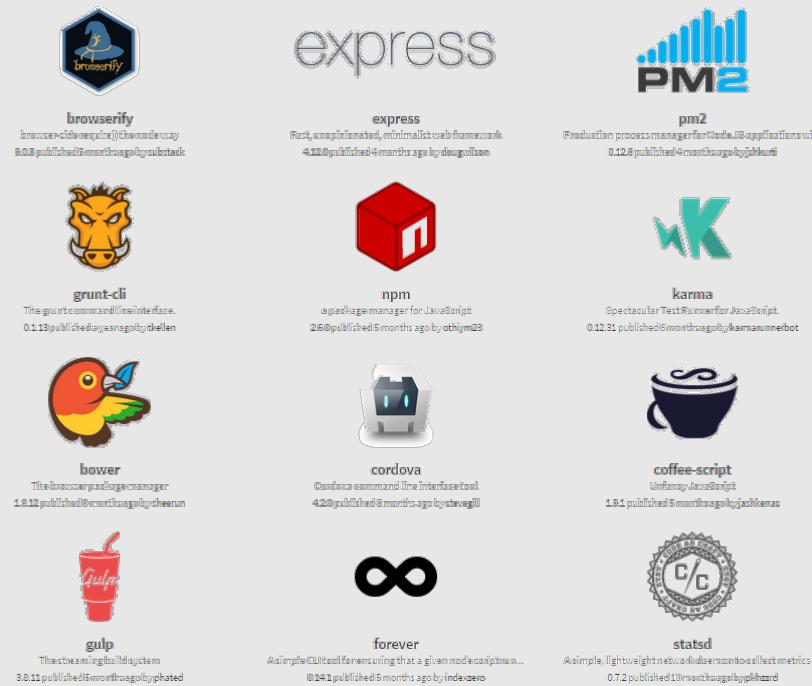


Node Package Manager

- NPM tiene una triple funcionalidad:
 - Repositorio de paquetes de terceros
 - Gestión de los paquetes instalados en el equipo
 - Estándar para definir dependencias de otros paquetes.
- NPM ofrece un servicio público de registro que contiene todos los paquetes que los programadores publican en el sitio Web asociado con NPM.
 - Disponible en <https://www.npmjs.com/>
 - La lista de paquetes disponibles está en crecimiento continuo...)
- También proporciona una herramienta de línea de comandos para descargar, instalar y manejar estos paquetes.
- Y podemos utilizar el formato de descriptor de paquete estándar para especificar qué módulos de terceros dependen de la aplicación.
 - Para esto se utilizan usualmente ficheros de configuración .JSON

Node Package Manager

- En el sitio web podemos igualmente consultar la lista de paquetes disponibles, aunque los más utilizados nos aparecen en un listado inicial
- El total de paquetes disponibles supera los 150.000.



Instalación y comprobación

Instalación de Node.js para Windows

Comprobación de funcionamiento de NPM

Alta y verificación del usuario activo (NPM)

Comprobación de las versiones en uso (NPM)

Acceso a la consola de Node

Algunas pruebas básicas

Express

➤ Características

- Framework web para trabajar con NodeJS
- Rápido
- Minimalista

➤ Instalación:

- `npm install express` ➔ instala en la carpeta del proyecto
- `npm install -g express` ➔ instala a nivel global del equipo
- `npm install express --save` ➔ instala y guarda en **package.json**
- `npm install express --save-dev` ➔ instala y guarda como dependencia para desarrollo (debug) en **package.json**

Express - Capacidades

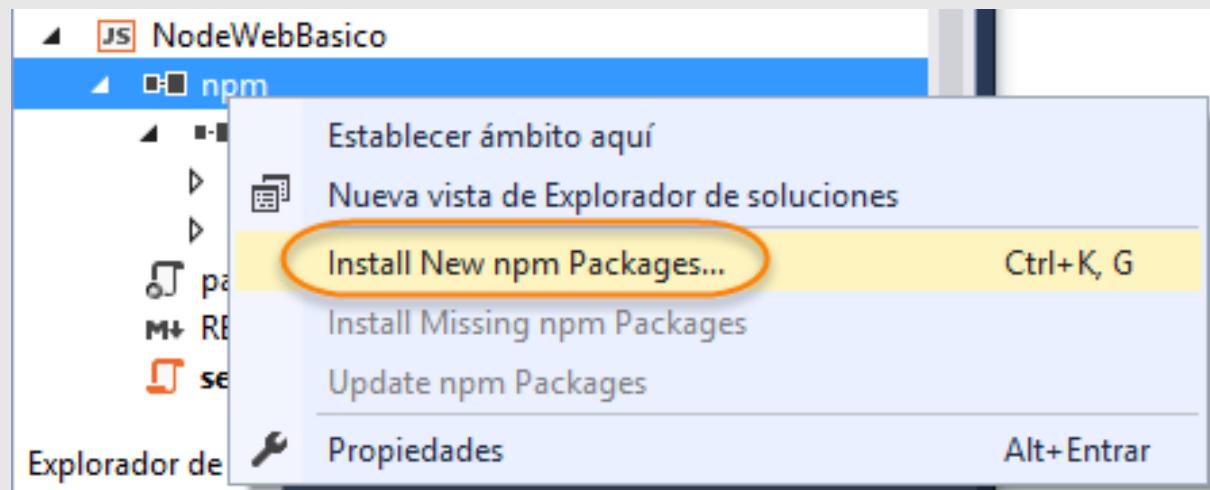
- Cuando indicamos que se trata de una API de REST, queremos decir que las operaciones típicas (CRUD, manejo de rutas, etc.) estarán disponibles
- Express facilita estas operaciones respecto a lo que sería su equivalente JavaScript.
- Además, se integra perfectamente con otros módulos existentes, como *path*, *location*, *url*, etc., para facilitar las configuraciones habituales de un entorno de desarrollo Web MVC.
- Muchas veces, el valor de retorno de una petición se "traducirá" en un fichero *json*, que manejaremos en el cliente.
- Funciona **estrictamente** en el "*Back-End*"

Express - Instalación

- Si instalamos Express (y/o Jade) con NPM desde la línea de comandos, solo tenemos que indicar:

```
npm install express
```

- Si lo hacemos con Visual Studio, seleccionamos los paquetes NPM, y con botón derecho nos da la opción de "Install NPM Packages"



Express - Instalación

- Una vez hecho esto, se añade una dependencia al fichero *package.json*.



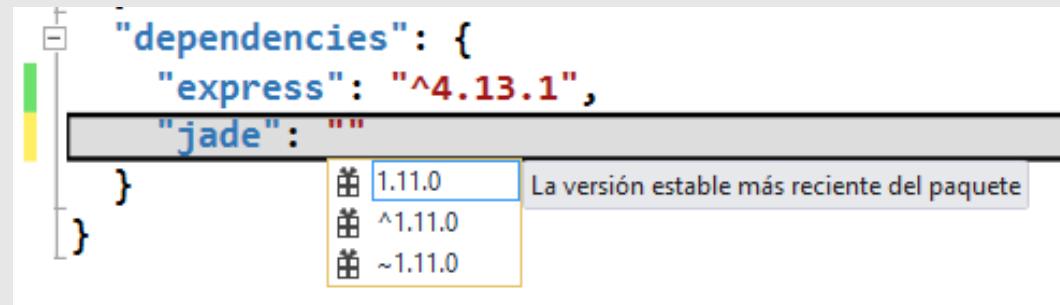
```
package.json*  X server.js  server.js  package.json
http://json.schemastore.org/package

{
  "name": "NodeWebBasico",
  "version": "0.0.0",
  "description": "NodeWebBasico",
  "main": "server.js",
  "author": {
    "name": "Marino",
    "email": "mpm@elavefenix.net"
  },
  "dependencies": {
    "express": "^4.13.1"
  }
}
```

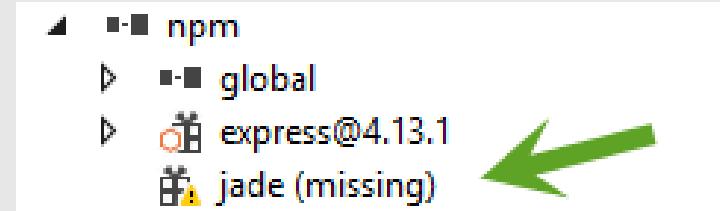
- Donde el acento circunflejo que aparecen delante de la versión indica que es válida cualquier versión hasta la 4.13.1.
- Si apareciera el símbolo tilde (~) indica límite de versión a dos niveles.
- Y el asterisco (*) indica que la versión es indiferente.

Express - Instalación

- La otra opción (cuando ya conocemos el paquete a instalar), consiste en indicárselo directamente en el editor de *package.json* y aprovechar el "Intellisense", ya que el propio editor nos va sugerir las versiones disponibles:

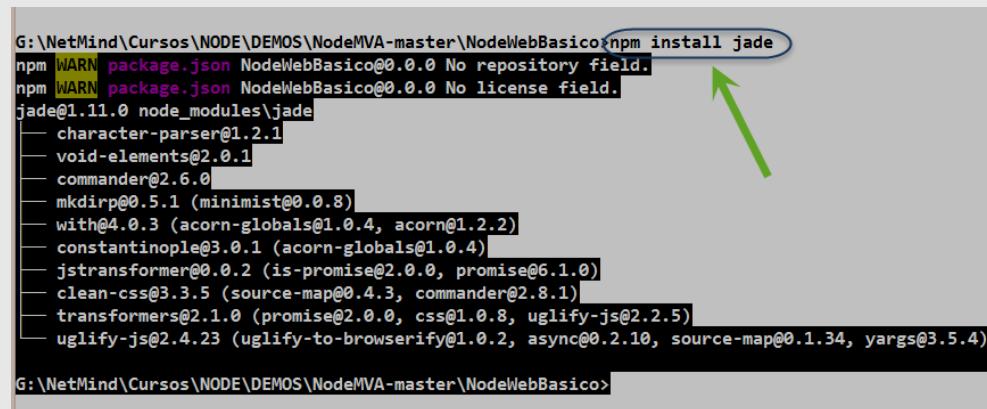


- Para acelerar el tiempo de respuesta y permitir configurar por esta vía sin necesidad de instalar, el paquete aparecerá en la vista de soluciones, como "*jade (missing)*", y podremos instalarlo después a voluntad. (Esto depende de la configuración de Visual Studio.)



Express - Instalación

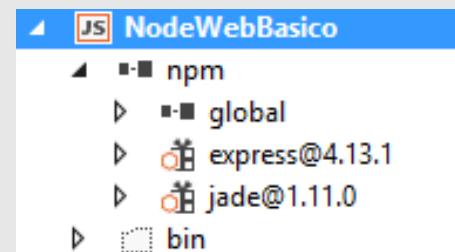
- Otra posibilidad, (o si estamos en otro entorno u otro IDE), consiste en instalarlo directamente desde la línea de comandos.
- (Para instalar **Jade** desde la línea de comandos: CMD - posiblemente con permisos de administrador)
- Y la sintaxis comentada anteriormente



```
G:\NetMind\Cursos\NODE\DEMOS\NodeMVA-master\NodeWebBasico>npm install jade
npm WARN package.json NodeWebBasico@0.0.0 No repository field.
npm WARN package.json NodeWebBasico@0.0.0 No license field.
jade@1.11.0 node_modules\jade
├── character-parser@1.2.1
├── void-elements@2.0.1
├── commander@2.6.0
├── mkdirp@0.5.1 (minimist@0.0.8)
├── with@4.0.3 (acorn-globals@1.0.4, acorn@1.2.2)
├── constantinople@3.0.1 (acorn-globals@1.0.4)
├── jstransformer@0.0.2 (is-promise@2.0.0, promise@6.1.0)
├── clean-css@3.3.5 (source-map@0.4.3, commander@2.8.1)
├── transformers@2.1.0 (promise@2.0.0, css@1.0.8, uglify-js@2.2.5)
└── uglify-js@2.4.23 (uglify-to-browserify@1.0.2, async@0.2.10, source-map@0.1.34, yargs@3.5.4)

G:\NetMind\Cursos\NODE\DEMOS\NodeMVA-master\NodeWebBasico>
```

- Podemos comprobar que está instalado en la carpeta de **npm** en el proyecto



Express

Instalación

Opciones de configuración

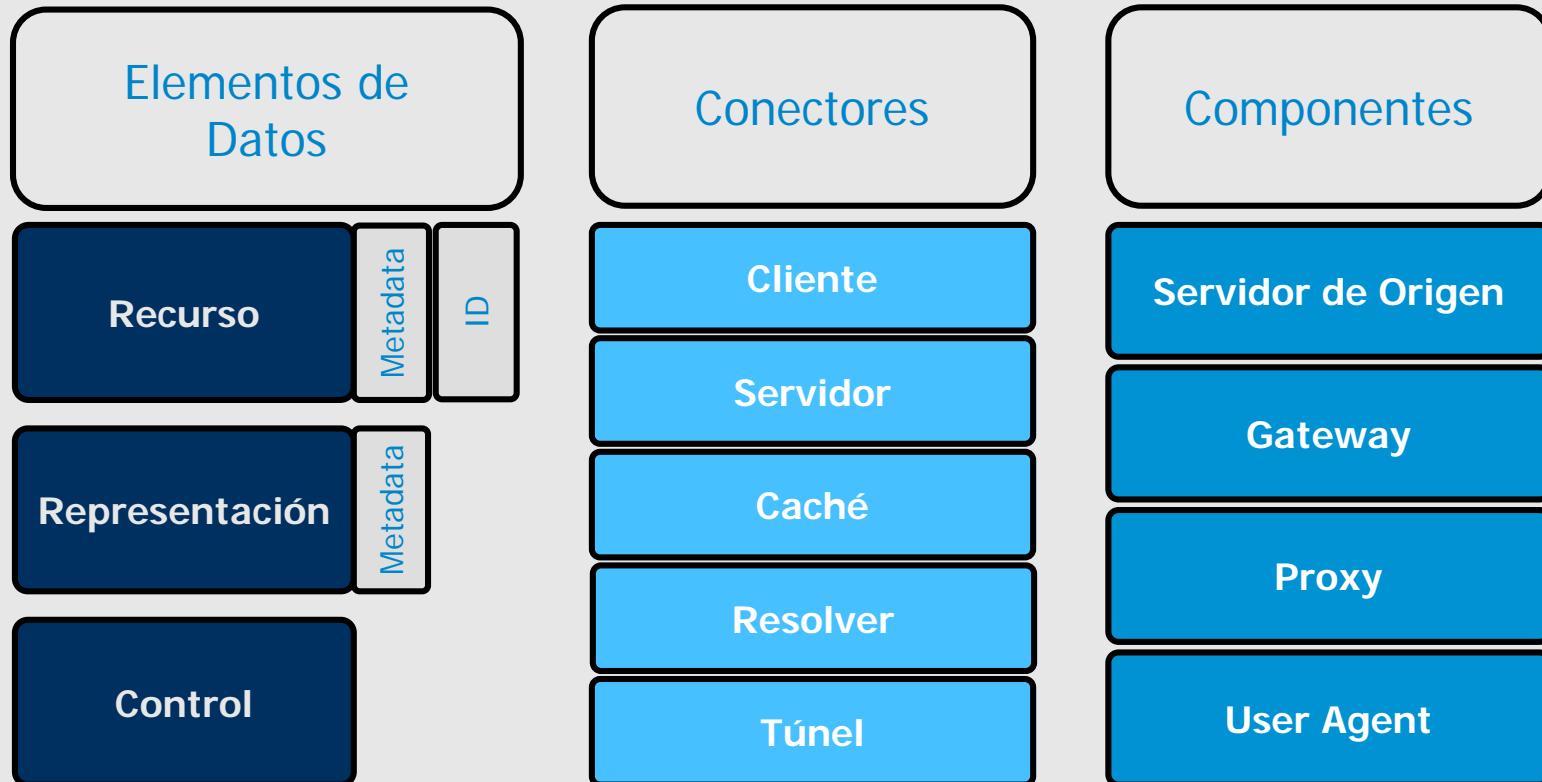
Manejo del archivo “package.json”

Servicios REST

➤ **Representational State Transfer**

- “REST está dirigido a obtener una imagen de cómo de bien diseñada está una aplicación Web: una red de páginas web (una máquina de estados virtual) donde el usuario navega a lo largo de una aplicación seleccionando enlaces (transiciones de estado) apareciendo en la siguiente página (siguiente estado) transfiriendo al usuario y mostrando el contenido para su uso.” – Roy Fielding
- Arquitectura simple sin estado

REST - Componentes



REST

<https://www.googleapis.com/language/translate/v2/languages>

Representación

```
{  
  "data": {  
    "languages": [  
      {  
        "language": "af"  
        ...},  
      {  
        "language": "am"  
        ...},  
      {  
        "language": "ar"  
        ...},  
      {  
        "language": "az"  
        ...},  
      {  
        "language": "be"  
        ...},  
      {  
        "language": "bg"  
        ...},  
      {  
        "language": "ca"  
        ...},  
      {  
        "language": "cs"  
        ...},  
      {  
        "language": "da"  
        ...},  
      {  
        "language": "de"  
        ...},  
      {  
        "language": "el"  
        ...},  
      {  
        "language": "en"  
        ...},  
      {  
        "language": "es"  
        ...},  
      {  
        "language": "et"  
        ...},  
      {  
        "language": "fi"  
        ...},  
      {  
        "language": "fr"  
        ...},  
      {  
        "language": "hr"  
        ...},  
      {  
        "language": "it"  
        ...},  
      {  
        "language": "lt"  
        ...},  
      {  
        "language": "lv"  
        ...},  
      {  
        "language": "nl"  
        ...},  
      {  
        "language": "no"  
        ...},  
      {  
        "language": "pl"  
        ...},  
      {  
        "language": "pt"  
        ...},  
      {  
        "language": "ro"  
        ...},  
      {  
        "language": "ru"  
        ...},  
      {  
        "language": "sl"  
        ...},  
      {  
        "language": "sv"  
        ...},  
      {  
        "language": "th"  
        ...},  
      {  
        "language": "tr"  
        ...},  
      {  
        "language": "uk"  
        ...},  
      {  
        "language": "vi"  
        ...},  
      {  
        "language": "zh"  
        ...}  
    ]  
  }  
}
```



Recurso

Listado Idiomas
Google Translate



Principios de diseño REST

➤ Principios de diseño REST

1. Identificar todas las entidades que queremos exponer
2. Crear una URL para cada recurso
3. Categorizar los recursos con las opciones disponibles (leer, modificar, añadir, eliminar)
4. Por defecto todos los recursos se pueden leer, salvo restricciones por permisos
5. Establecer vínculos a los recursos
6. Diseñar los servicios de manera gradual
7. Especificar el formato de respuesta (JSON, XML, PDF...)
8. Documentar los servicios para que los clientes sepan cómo consumirlos

Operaciones REST

➤ RESTful

- Servicios que implementan la arquitectura REST
- Típicamente ofrecen todas las operaciones CRUD (Create, Reference, Update, Delete)
- Operaciones alineadas con métodos HTTP

Recurso / Estado	GET	POST	PUT	DELETE
/pedidos	Devuelve la lista de pedidos	Crea un pedido	Modifica pedidos	Elimina los pedidos
/pedidos/432	Devuelve los detalles del pedido 432	ERROR	Modifica el pedido 432	Borra el pedido 432

Uso de REST

➤ **Conexión entre plataformas**

1. Exponer datos de una fuente
2. Consumirlo desde cualquier plataforma

➤ Ejemplo: Lectura / escritura de datos de una base de datos Oracle desde una app Android

➤ **Clientes ligeros interactuando con API REST públicas o privadas**

1. Casi el 70% de API públicas tienen una interfaz tipo REST

➤ Ejemplos: Amazon, Google, Netflix, Twitter, Facebook, ...

➤ **Rich Internet Applications (RIA)**

1. Envío de peticiones AJAX a la interfaz REST utilizando JavaScript
2. Respuestas en formato XML o JSON principalmente

REST - Características

Creadores y
Proveedores
desacoplados

Sistema de
capas

Interfaz
uniforme

Interfaz uniforme

➤ HTTP – RFC 2616

- **Seguro** ➔ Sin efectos laterales == no modifica el recurso
- **Idempotente** ➔ Efectos laterales similares con una o n peticiones

Método	Operación en el recurso	Seguro	Idempotente
POST	Crear un nuevo recurso	No	No
GET	Obtiene una representación de recurso	Sí	Sí
PUT	Actualiza un recurso o crea un nuevo recurso	No	Sí
DELETE	Elimina un recurso	No	Sí

RESTful API

- **Versionar nuestra API**
- Manejar errores con **códigos de estado HTTP**

Código	Mensaje	Resultado
200	OK	Todo funciona bien
201	Created	Recurso creado
202	Accepted	Petición aceptada pero sin procesar
204	OK	Recurso eliminado
304	Not Modified	Se puede usar datos de caché
400	Bad Request	Petición no válida
401	Unauthorized	Requiere autenticación de usuario
403	Forbidden	Servidor entendió pero rechazó petición
404	Not Found	No hay recurso en la URI
405	Method Not Allowed	Método HTTP no permitido
422	Unprocessable Entity	No se puede procesar la entidad
429	Too Many Requests	Possible ataque DoS detectado
500	Internal Server Error	No se registra la petición

Creación de una API básica: Enrutamiento

- El sistema de "*routing*" funciona de manera similar a otros entornos MVC.
- Permite definir patrones de rutas que serán interpretados en tiempo de ejecución para determinar cuál es el contenido solicitado.
- Por tanto, "mapean" una petición URL a una función de "*callback*".
- Las peticiones, se pueden emitir en cualquiera de los verbos válidos de HTTP para operaciones REST: GET, POST, PUT, DELETE, etc.
- La combinación del sistema de "*routing*" y el uso de verbos HTTP posibilita la creación de aplicaciones estilo "MVC".

Construcción de una API básica: Enrutamiento

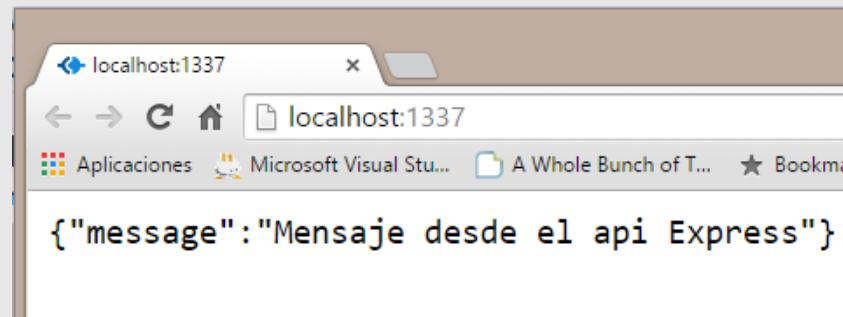
- Permite re-direccionar *urls* del tipo a <http://localhost:8888/index> a una vista concreta
- El modelo de programación adopta esta forma:

```
app.get('/index', function (req, res) {});
```

- En el ejemplo, cuando el usuario solicita "**/index**" dentro del sitio, una función de *callback* gestionará la petición y la respuesta.
 - También se pueden usar expresiones regulares para indicar la ruta.
 - Opcionalmente, se puede devolver *json* directamente.
 - Y también otros formatos MIME reconocibles.

Construcción de una API básica: Enrutamiento

- El código siguiente cumple la función de crear un sistema básico de respuesta
- Implícitamente, está creando un servidor que escucha en el puerto de configuración, o –alternativamente- en el 8080



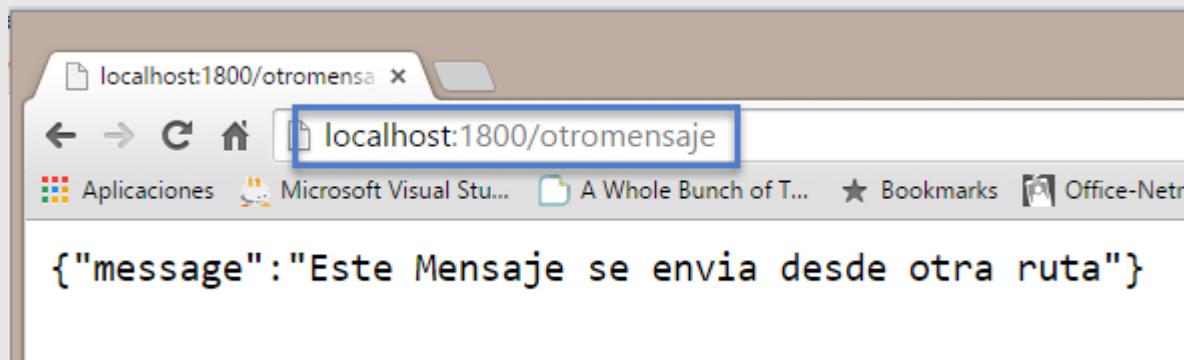
- También podemos lanzar la aplicación desde la línea de comandos
- En ese caso podemos cambiar el puerto de escucha si le añadimos el modificador **-p {puerto}**

Construcción de una API básica

- Si añadimos una ruta alternativa, podemos modificar el contenido de la respuesta:

```
app.get('/otromensaje', function (req, res) {  
    res.json({ message: 'Mensaje desde otra ruta' });  
});
```

- De forma que solicitamos la nueva url:
<http://localhost:9999/otromensaje>
- Y la respuesta debe ser distinta



Express

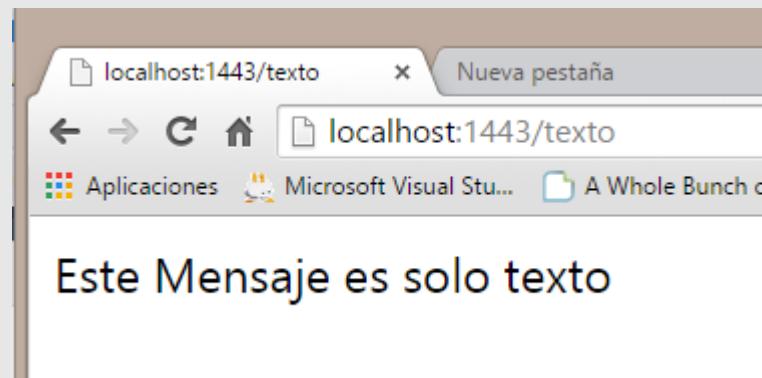
Enrutamiento

Valores de retorno

- El objeto *res* dispone de métodos diversos de retorno en función del tipo de contenido.
- Esto incluye la posibilidad de enviar texto plano, de forma que podríamos utilizar el método *send('txt')*

```
app.get('/texto', function (req, res) {  
    res.send('Este Mensaje es solo texto');  
});
```

- Con el mismo resultado:



Devolver archivos

- En aplicaciones reales, podremos utilizar dos enfoques distintos para la construcción de un sitio:
 - Enfoque clásico: usar archivos HTML/CSS
 - Enfoque modular: usar motores de vistas que generen ambos tipos
- Si queremos devolver un fichero HTML, resulta sencillo de hacer mediante el método *sendfile("archivo")*:

```
app.get('/AcercaDe', function (req, res) {  
    res.sendFile('./acercade.html');  
});
```

- Y si queremos una estructura básica podemos enlazar la página inicial con vínculos a las otras páginas utilizando la sintaxis clásica de Node en un fichero HTML

Devolver archivos

- Este sería el cuerpo de la página principal:

```
<h1>Esta es Página Principal</h1>
  
  <!--ul>li>a[href=./enlace]{enlace}*3-->
  <ul>
    <li>
      <a href="./OtrosDatos">Otros Datos</a>
      <a href=".//Acercade">Acerca de...</a>
    </li>
  </ul>
```

- Donde la petición a "**Acercade**", observamos que no lleva extensión (igual que la otra), ya que es manejada por Node correctamente, mediante el sistema de enrutamiento.

Express

Página básica con Node y Express

Express como "middleware"

- La otra funcionalidad principal de Express es la de sus servicios de "*middleware*".
- Entendemos por tal una función con acceso al objeto *request* (*req*) y al objeto *response* (*res*).
- Eso implica igualmente el acceso a todo el ciclo de solicitud-respuesta de la aplicación.
- Este ciclo se referencia en Express comúnmente con un identificador llamado *next* que reenvía el procesamiento al "siguiente de las lista de procesos".

Capacidades del "middleware"

➤ *Middleware* es capaz de:

- Ejecutar cualquier código
- Realizar cambios en la petición y en la respuesta.
- Finalizar el ciclo petición-respuesta
- Llamar al siguiente "middleware" en la pila
- Si el *middleware* actual no termina el ciclo de solicitud-respuesta, debe llamar a `next()` para pasar el control a la siguiente *middleware*, de lo contrario la solicitud se quedará "colgada".

Express como middleware

- En general, se pueden considerar el "middleware" como un recurso aplicable a 5 contextos de desarrollo distintos:
 - **Aplicación (*Application-level*)**
 - **Rutas (*Router-level*)**
 - **Gestión de errores (*Error-handling*)**
 - **Incorporado (*Built-in*)**
 - **De terceras partes**
- De esta forma, cuando construimos un sitio web más complejo, el "andamiaje" que necesitemos para conseguirlo se compondrá de:
 - **servidor + middleware + cliente**
- Lo correcto es crear una estructura de directorios adecuada pensando en los dominios de la aplicación, de manera que resulte fácil localizar cualquier recurso programable.

Express como "middleware"

- Express puede servir contenido estático (como hemos visto), pero también dinámico.
- En el modo dinámico, además, puede operar de otras formas para servir el contenido a los clientes:
 - Integrando motores HTML como "*Handlebars*" (HBS) o EJS
 - Utilizando otros motores de optimización especializados (pero que no soportan la mayoría de editores todavía), como *Jade* y *Stylus*
 - Jade funciona como motor de vistas HTML, (ya comentado), y tiene una sintaxis curiosa, basada en eliminar todos los símbolos "<" y ">" del código HTML, respetando la indentación para indicar las estructuras "contenedor/contenido"
 - Por su parte, *stylus* hace lo mismo con el código CSS, solo que esta vez eliminando las llaves "{}" y utilizando indentación para indicar pertenencia.
- Las referencias a "middleware" se utilizan casi siempre con la sentencia:
`app.use("middeware");`

Express como “middleware”

- Para ver un ejemplo de esto, consideremos el siguiente código fuente:

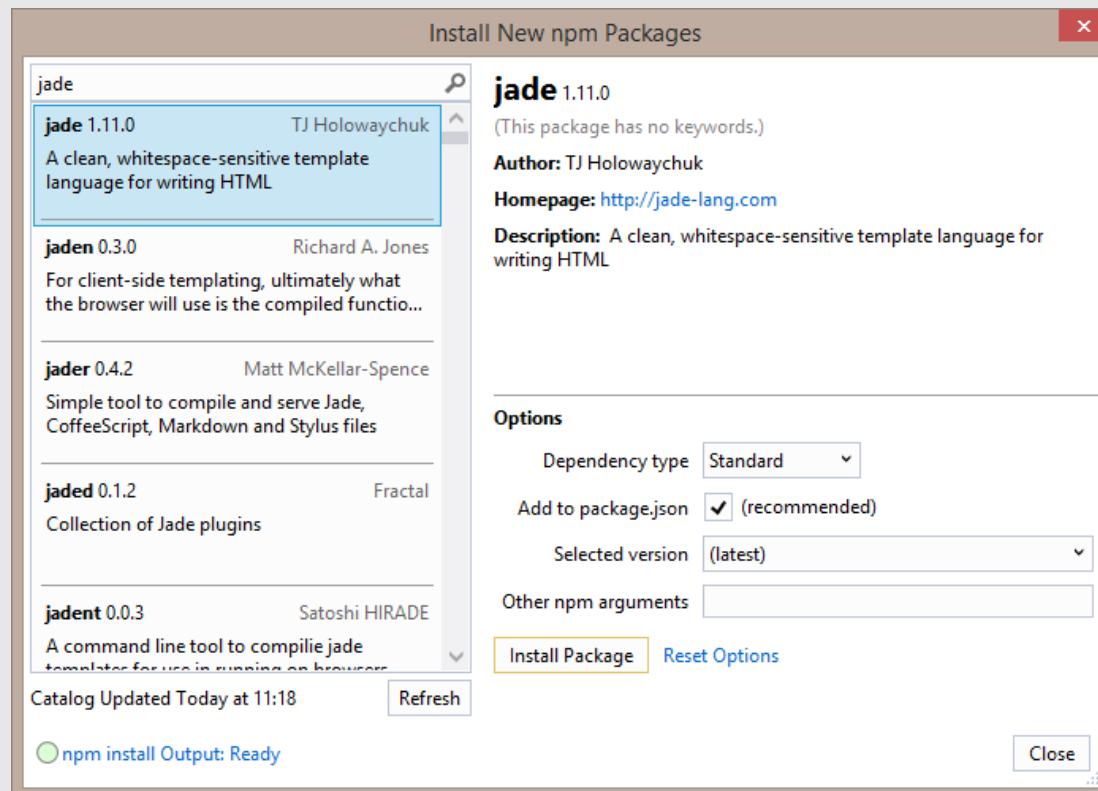
```
// Este middleware impide que el request continúe
app.use(function (req, res, next) {
    res.send('Hola Mundo');
})
```

```
// los requests nunca alcanzan esta ruta
app.get('/', function (req, res) {
    res.send('Bienvenidos');
})
```

- La sentencia *app.get* es inútil, ya que el “middleware” ha omitido la llamada a *next()*;

Usando "Jade" para un sitio Web

- El sitio web oficial de *jade*: <http://jade-lang.com/>
- Una vez creado el sitio Web, tendremos que instalar "jade" (ahora en la versión 1.11.0) además de "Express":



Usando “Jade” para un sitio Web

- Para probar inicialmente el funcionamiento de los motores de renderización, podemos utilizar una de las plantillas predeterminadas de aplicación Express (la versión 3 o la 4), dentro de Visual Studio
- Estas plantillas usan *jade* y *stylus*, para montar una aplicación básica que ya contiene una estructura de directorios y ubica los elementos en distintas carpetas que son manejadas como módulos separados.
- El fichero principal de la aplicación (*app.js*, habitualmente) es el encargado de "aglutinar" todos los contenidos de forma que podamos disponer de una separación adecuada de los componentes.
 - Recordemos que –aunque no exista el concepto de componente en JavaScript, Node puede tratar como módulos separados cada una de las referencias a *require* que hacemos en el principal.
- La plantilla, además nos mostrará otras buenas prácticas, como la división de las vistas y su ubicación en una carpeta "*views*" que Node asume como ubicación predeterminada.

Análisis de la primera aplicación con Express

- Aparte de la estructura de directorios y contenidos que nos ofrece la aplicación, el análisis del contenido de los ficheros *js* y *jade* es bastante revelador.
- En la aplicación principal, vemos que (y quedémonos ya con eso como buena práctica), lo primero que se hace es referenciar en variables todos los módulos necesarios para el funcionamiento, incluyendo los que definen las rutas.

```
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var routes = require('./routes/index');
var users = require('./routes/users');
```

- Hecho esto, la llamada

```
var app = express();
```

- Establece el contexto principal de ejecución.

Análisis de la primera aplicación con Express

- A continuación, se configuran los motores de vistas:

```
// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');
```

- Lo siguiente, es configurar y definir el "middleware" (referencias a gestión de "cookies", al contexto de ejecución, al motor de estilos *stylus*, etc.

```
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(require('stylus').middleware(path.join(__dirname, 'public')));
app.use(express.static(path.join(__dirname, 'public')));
```

- Hay vemos que las rutas (con esta configuración separada) se inicializan como "middleware" igualmente.

```
app.use('/', routes);
app.use('/users', users);
```

Análisis de la primera aplicación con Express

- La operativa siguiente, consiste en definir funciones para el tratamiento de errores (dos distintas, según estemos en depuración o en producción, para evitar incluir en los errores la pila de llamadas).
- Por último, se exporta la propia aplicación para que el código de otros módulos pueda acceder a sus contenidos:

```
module.exports = app;
```

- El fichero de rutas se define separadamente en el directorio "*routes*".
 - Notar que la última instrucción "añade" al mecanismo general lo definido en este punto

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function (req, res) {
  res.render('index', { title: 'Express con Jade' });
});

module.exports = router;
```

Análisis de la primera aplicación con Express

- Por último, las vistas de *jade* tienen ese aspecto de HTML "pelado":

```
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    block content
```

- Donde vemos que las etiquetas carecen de símbolos "<>", los atributos de los elementos se expresan con sintaxis de función, y existen algunas palabras reservadas que parecen definidas en alguna otra parte.
- La solución la vemos en la línea de rutas anterior, que definía la vista a presentar cuando se solicite la raíz del sitio **y los argumentos de datos a pasar a esa vista, expresados como un objeto json.**
- Lo que sucede es que el mecanismo de compilación de *jade* se encarga de buscar y resolver esas referencias.

Análisis de la primera aplicación con Express

- Los mismo sucede con las "hojas de estilo" *Stylus*. Definen una sintaxis simplificada para expresar CSS:

```
body
  padding: 50px
  font: 14px "Segoe UI", Helvetica, Arial, sans-serif
a
  color: #00B7FF
```

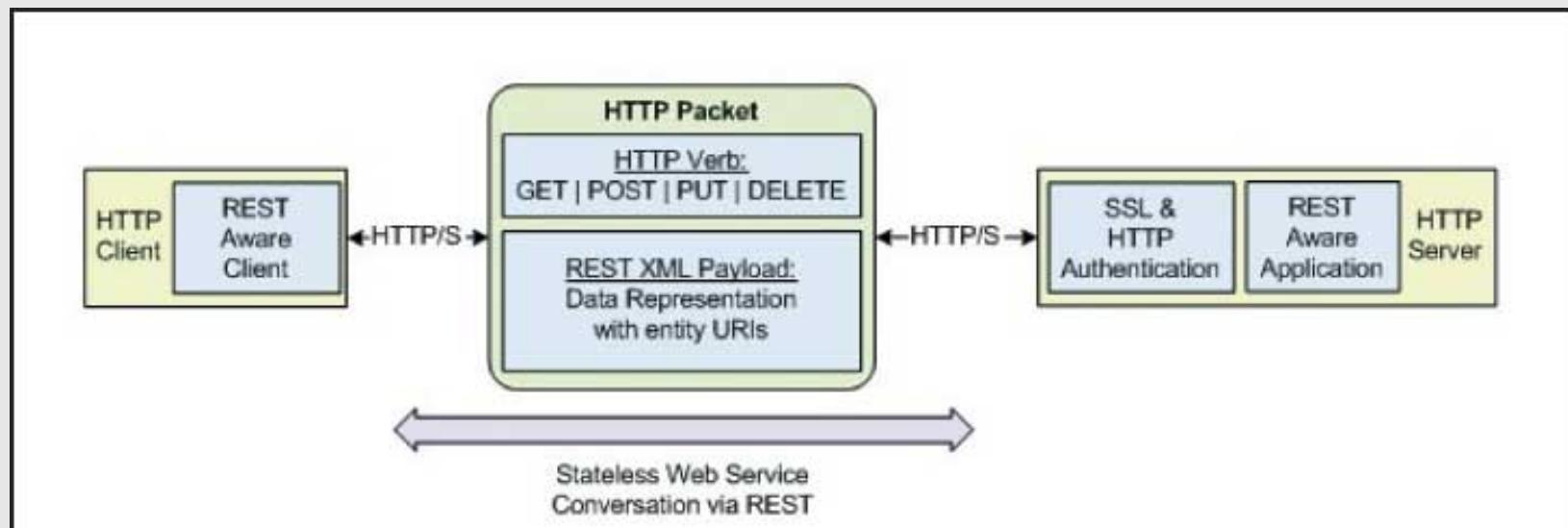
- Lo que es interesante respecto a la economía de pulsaciones, pero tiene el inconveniente de que perdemos las ayudas contextuales y el "Intellisense", la auto-compleción de código, etc., que nos pueden ofrecer los editores, ya que no hay de esto en los editores habituales.
- Además, no resulta imprescindible, porque, como veremos en la siguiente demo, podemos hacer referencia a los CSS sin que suponga ningún problema.

Referencias

- Documentación y sitio relacionados
 - Documentación oficial de Express:
 - <http://expressjs.com>
 - API del "middleware" de Express:
 - <http://expressjs.com/guide/using-middleware.html>
 - Best practices for Express app structure:
 - <http://www.terlici.com/2014/08/25/best-practices-express-structure.html>

REST y NodeJS

- Desde el punto de vista de la programación con Node, podemos configurar cualquier servidor para que se comporte como servidor REST teniendo en cuenta todo lo visto hasta ahora.
- Para ello, deberemos determinar el tipo de petición recibida, según el "verbo" HTTP utilizado en ella.
- Eso supone discriminar la petición, a partir del objeto *request*.
(recordar utilizar protocolos HTTP versión 1.2 o superior)



REST y Node

- En el código fuente, la clave es montar un sistema de discriminación basado en los dos criterios:
 - Gestión de la URL, según el modelo MVC de fragmentación de *path*.
 - Determinación de la acción a realizar, dependiendo del *verbo HTTP* que se solicite en la petición

```
function manejarPeticion(req, res) {  
    switch (req.method) {  
        case "GET": peticionGET(res); break;  
        case "POST": peticionPOST(res); break;  
        case "PUT": peticionPUT(res); break;  
        case "DELETE": peticionDELETE(res); break;  
        default: peticionERROR(res); break;  
    }  
}
```

REST y Node

- Más adelante, cada método implementa la funcionalidad necesaria

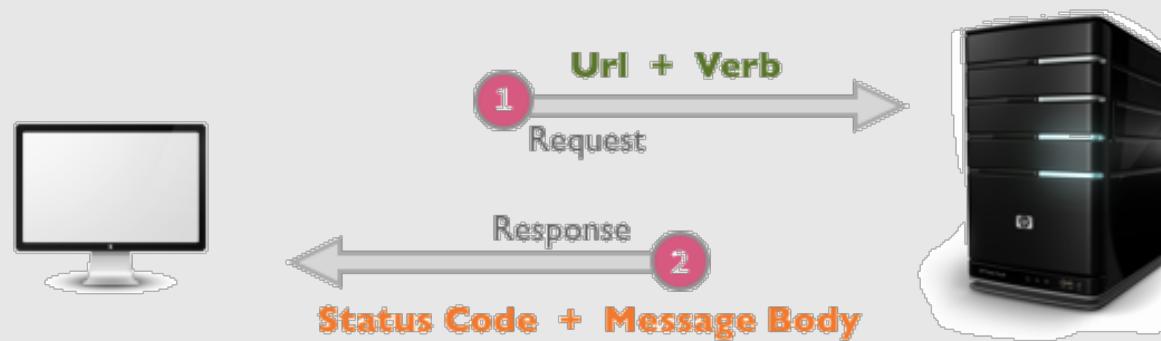
```
function peticionGET(res) {  
    res.writeHead(200, { 'Content-Type': 'text/plain' });  
    res.end('Recibida Petición GET\n');  
}
```

```
function peticionPOST(res) {  
    res.writeHead(200, { 'Content-Type': 'text/plain' });  
    res.end('Recibida Petición POST\n');  
}
```

- Para probarlo podemos usar *Fiddler*, u otras herramientas especializadas como *Curl*

REST y NodeJS

- Este objeto dispone de un conjunto de métodos y propiedades que almacenan la información recibida a bajo nivel, de forma que pueda ser consultada por un API de programación.
- La propiedad *method* del objeto *request* permite determinar qué método se ha usado y actuar en consecuencia.
- Además, en el caso de haber recibido argumentos, tenemos otras propiedades que nos permiten acceder a ellos y manipular los datos correspondientes.



REST

Construcción de un servidor REST con NodeJS

We Know IT

Barcelona

C. dels Almogàvers 123
08018 Barcelona
Tel. +34 933 041 720
Fax. +34 933 041 722

Madrid

Plaza Carlos Trías Bertrán,
7 Edificio Sollube, 1^a Planta
28020 Madrid
Tel. +34 914 427 703
Fax +34 914 427 707

Bilbao

C/ San Vicente, n° 8
Edificio Albia I,
6^a planta - dpto. 8
48001 – Bilbao
Tel. +34 944 354 982

www.netmind.es

Desarrollo Web con Angular 5 (front) y API REST con NodeJS (back)

Índice

1. Herramientas
2. Angular 5 – Front
3. Angular Material
4. NodeJS – Back
- 5. Testing**



Testing

Pruebas JavaScript con Protractor (Angular)

Pruebas JavaScript con Chai y Mocha (NodeJS)

Pruebas JavaScript con Protractor

➤ Protractor

- Framework de tests E2E
- Para aplicaciones Angular y AngularJS
- Disponible en:
 - <https://www.protractortest.org>
- Versión actual:
 - 5.4.0

Historial de versiones

Release	Fecha
0.2.0	12/06/2013
1.0.0	22/07/2014
2.0.0	18/03/2015
3.0.0	18/11/2015
4.0.0	12/07/2016
5.0.0	10/01/2017
5.4.0	18/07/2018

Pruebas JavaScript con Protractor

➤ Instalación

```
npm install -g protractor
```

- Disponibles dos aplicaciones al instalar:
 - **protractor** ➔ motor de ejecución de pruebas
 - **webdriver-manager** ➔ helper para integración con Selenium

Escribir tests

➤ Sintaxis

- **describe** ➔ crea una suite de pruebas
- **it** ➔ crea una prueba
- **browser** ➔ objeto global disponible
- **expect** ➔ valida operaciones

➤ Primer test

➤ Verifico si el título de la página es igual a "Super Calculator"

```
// spec.js
describe('Protractor Demo App', function() {
  it('should have a title', function() {
    browser.get('http://juliemr.github.io/protractor-demo/');

    expect(browser.getTitle()).toEqual('Super Calculator');
  });
});
```

Escribir tests

➤ Primer test

- Verifica si el título de la página es igual a “Super Calculator”

```
// spec.js
describe('Protractor Demo App', function() {
  it('should have a title', function() {
    browser.get('http://juliemr.github.io/protractor-demo/');

    expect(browser.getTitle()).toEqual('Super Calculator');
  });
});
```

➤ Configuración

```
// conf.js
exports.config = {
  framework: 'jasmine',
  seleniumAddress: 'http://localhost:4444/wd/hub',
  specs: ['spec.js']
}
```

Escribir tests

➤ Interactuar con elementos

- Busca en la página los valores "first" y "second"
- Invoca el click en el botón "gobutton"
- Valida que el valor de "latest" es igual a 5

```
// spec.js
describe('Protractor Demo App', function() {
  it('should add one and two', function() {
    browser.get('http://juliemr.github.io/protractor-demo/');
    element(by.model('first')).sendKeys(1);
    element(by.model('second')).sendKeys(2);

    element(by.id('gobutton')).click();

    expect(element(by.binding('latest')).getText()).
     toEqual('5'); // This is wrong!
  });
});
```

Múltiples tests en una suite

```
// spec.js
describe('Protractor Demo App', function() {
  var firstNumber = element(by.model('first'));
  var secondNumber = element(by.model('second'));
  var goButton = element(by.id('gobutton'));
  var latestResult = element(by.binding('latest'));

  beforeEach(function() {
    browser.get('http://juliemr.github.io/protractor-demo/');
  });

  it('should have a title', function() {
    expect(browser.getTitle()).toEqual('Super Calculator');
  });

  it('should add one and two', function() {
    firstNumber.sendKeys(1);
    secondNumber.sendKeys(2);

    goButton.click();

    expect(latestResult.getText()).toEqual('3');
  });

  it('should add four and six', function() {
    // Fill this in.
    expect(latestResult.getText()).toEqual('10');
  });

  it('should read the value from an input', function() {
    firstNumber.sendKeys(1);
    expect(firstNumber.getAttribute('value')).toEqual('1');
  });
});
```

Mocha y Chai

➤ Mocha

- Framework de tests JavaScript
- Se ejecuta sobre Node.JS y en el navegador
- Realiza tests asíncronos

Release	Fecha
0.0.1	22/11/2011
1.0.0	2012
2.0.0	2014
3.0.0	2016
4.0.0	2017
5.0.0	22/01/2018
5.2.0	22/05/2018

Mocha y Chai

➤ Chai

- Biblioteca de creación de tests
- NodeJS y JavaScript en navegador

Release	Fecha
0.0.1	2011
1.0.0	2012
1.9.0	29/01/2014
2.0.0	09/02/2015
3.0.0	04/06/2015
4.0.0	26/05/2017
4.1.2	31/08/2017

Instalación de Mocha y Chai

- Instalación de Mocha
 - **npm install –g mocha**
- Instalación de Chai y Chai-HTTP
 - **npm install –g chai**
 - **npm install –g chai-http**
- Testing ➔ Entorno de dev ➔ configuración siguiente
 - **npm install -g mocha --save-dev**
 - **npm install -g chai --save-dev**
 - **npm install -g chai-http --save-dev**

Primer test

➤ Carga de las bibliotecas

```
var chai = require('chai');
var chaiHttp = require('chai-http');
var server = require('../server/app');
var should = chai.should();

chai.use(chaiHttp);
```

➤ Definición de los tests

```
describe('Blobs', function() {
  it('should list ALL blobs on /blobs GET');
  it('should list a SINGLE blob on /blob/<id> GET');
  it('should add a SINGLE blob on /blobs POST');
  it('should update a SINGLE blob on /blob/<id> PUT');
  it('should delete a SINGLE blob on /blob/<id> DELETE');
});
```

Ejecución de los tests

- Valida que obtiene un HTTP OK (200) al pedir la URL /blobs

```
it('should list ALL blobs on /blobs GET', function(done) {
  chai.request(server)
    .get('/blobs')
    .end(function(err, res){
      res.should.have.status(200);
      done();
    });
});
```

```
$ mocha

Blobs
Connected to Database!
GET /blobs 200 19.621 ms - 2
  ✓ should list ALL blobs on /blobs GET (43ms)
    - should list a SINGLE blob on /blob/<id> GET
    - should add a SINGLE blob on /blobs POST
    - should update a SINGLE blob on /blob/<id> PUT
    - should delete a SINGLE blob on /blob/<id> DELETE

1 passing (72ms)
4 pending
```

Ejecución de los tests

➤ Test para validar una operación POST

```
it('should add a SINGLE blob on /blobs POST', function(done) {
  chai.request(server)
    .post('/blobs')
    .send({name: 'Java', lastName: 'Script'})
    .end(function(err, res){
      res.should.have.status(200);
      res.should.be.json;
      res.body.should.be.a('object');
      res.body.should.have.property('SUCCESS');
      res.body.SUCCESS.should.be.a('object');
      res.body.SUCCESS.should.have.property('name');
      res.body.SUCCESS.should.have.property('lastName');
      res.body.SUCCESS.should.have.property('_id');
      res.body.SUCCESS.name.should.equal('Java');
      res.body.SUCCESS.lastName.should.equal('Script');
      done();
    });
});
```

We Know IT

Barcelona

C. dels Almogàvers 123
08018 Barcelona
Tel. +34 933 041 720
Fax. +34 933 041 722

Madrid

Plaza Carlos Trías Bertrán,
7 Edificio Sollube, 1^a Planta
28020 Madrid
Tel. +34 914 427 703
Fax +34 914 427 707

Bilbao

C/ San Vicente, n^o 8
Edificio Albia I,
6^a planta - dpto. 8
48001 – Bilbao
Tel. +34 944 354 982

www.netmind.es



Tú opinión es importante para nosotros

Por favor, envíanos tus comentarios y sugerencias sobre el contenido ...

Desde Netmind agradecemos tu tiempo y tendremos en consideración tus comentarios para mejorar continuamente nuestra oferta formativa para que siempre os encontréis satisfechos de habernos elegido.

quality@netmind.es

Barcelona Carrer Almogàvers 123 (Edificio Ecourban) / T. +34 933 041 720 / F. +34 933 041 722 / 08018 Barcelona
Madrid Plaza de Carlos Trías Bertrán 7, 1^a Planta (Edificio Sollube) / T. +34 914 427 703 / 28020 Madrid
Bilbao San Vicente 8, 6^a Planta (Edificio Albia I) / T. +34 944 354 982 / 48001 Bilbao
Miami 1101 Brickell Avenue, (South Tower) 8th Floor/ T. +1 (305) 424 1668 / Miami, Florida 33131
Atlanta 296 South Main Street, Suite 300/ T. +1 (404) 678 366 1363 / Alpharetta, Atlanta, GA 30009-1973

info@netmind.es | www.netmind.es | www.netmind.net

Áreas de Formación

Gestión de Proyectos
Business Analysis
Agile Management
Gestión de Servicios TI
Enterprise Architecture
Business Games
Gestión del Cambio
Gobierno TI