Introducción

Angular 2 ha cambiado tanto que hasta el nombre es distinto. Lo conocíamos como "AngularJS" y ahora es sólo "Angular". No deja de ser una anécdota que hayan eliminado el "JS" hasta del nombre del dominio, pero es representativo. No porque ahora Angular no sea Javascript, sino porque es evolución radical. Angular 2 es otro framework, no simplemente una nueva versión. A los que no conocían Angular 1 ésto les será indiferente, pero los que ya dominaban este framework sí deben entender que el conocimiento que necesitan adquirir es poco menos que si comenzasen desde cero. Obviamente, cuanta más experiencia en el desarrollo se

tenga, más sencillo será lanzarse a usar Angular 2 porque muchas cosas sonarán de antes.

Problemas en Angular 1

Javascript: Para comenzar encontramos problemas en la creación de aplicaciones debido al propio Javascript. Es un lenguaje con carácter dinámico, asíncrono y de complicada depuración. Al ser tan particular resulta difícil adaptarse a él, sobre todo para personas que están acostumbradas a manejar lenguajes más tradicionales como Java o C#, porque muchas cosas que serían básicas en esos lenguajes no funcionan igualmente en Javascript.

Problemas en Angular 1

Desarrollo del lado del cliente: Ya sabemos que con Angular te llevas al navegador mucha programación que antes estaba del lado del servidor, comenzando por el renderizado de las vistas. Esto hace que surjan nuevos problemas y desafíos. Uno de ellos es la sobrecarga en el navegador, haciendo que algunas aplicaciones sean lentas usando Angular 1 como motor. Por otra parte tenemos un impacto negativo en la primera visita, ya que se tiene que descargar todo el código de la aplicación (todas las páginas, todas las vistas, todas las rutas, componentes, etc), que puede llegar a tener un peso de megas.

Soluciones implementadas en

Angular 2

TypeScript / Javascript: La sugerencia de usar TypeScript para desarrollar en Angular es casi una imposición porque la documentación y los generadores de código están pensados en TypeScript. Se supone que en futuro también estarán disponibles para Javascript, pero de momento no es así. De todos modos, para la tranquilidad de muchos, TypeScript no agrega más necesidad de procesamiento a las aplicaciones con Angular 2, ya que este lenguaje solamente lo utilizas en la etapa de desarrollo y todo el código que se ejecuta en el navegador es al final Javascript, ya que existe una transpiración previa.

Soluciones implementadas en Angular 2

Lazy SPA: Ahora el inyector de dependencias de Angular no necesita que estén en memoria todas las clases o código de todos los elementos que conforman una aplicación. En resumen, ahora con Lazy SPA el framework puede funcionar sin conocer todo el código de la aplicación, ofreciendo la posibilidad de cargar más adelante aquellas piezas que no necesitan todavía.

Soluciones implementadas en Angular 2

Renderizado Universal: Angular nació

para hacer web y renderizar en HTML en el navegador, pero ahora el renderizado universal nos permite que no solo se pueda renderizar una vista a HTML. Gracias a ésto, alguien podría programar una aplicación y que el renderizado se haga, por ejemplo, en otro lenguaje nativo para un dispositivo dado. Otra cosa que permite el renderizado universal es que se use el motor de renderizado de Angular del lado del servidor. Es una de las novedades más interesantes, ya que ahora podrás usar el framework para renderizar vistas del lado del servidor, esto permite mejorar tu SEO

Soluciones implementadas en Angular 2

Data Binding Flow: Uno de los motivos del

éxito de Angular 1 fue el data binding, pero éste tenía un coste en tiempo de procesamiento en el navegador, que si bien no penalizaba el rendimiento en todas las aplicaciones sí era un problema en aquellas más complejas. El flujo de datos ahora está mucho más controlado y el desarrollador puede direccionarlo fácilmente, permitiendo optimizar las aplicaciones. El resultado es que en Angular 2 las aplicaciones pueden llegar a ser hasta 5 veces más rápidas.

Soluciones implementadas en Angular 2

Componentes: La arquitectura de una aplicación Angular ahora se realiza mediante componentes. En este caso no se

trata de una novedad de la versión 2, ya que en la versión de Angular 1.5 ya se introdujo el desarrollo basado en componentes. Sin embargo, la componetización no es algo opcional como en Angular 1.5, sino es una obligatoriedad. Los componentes son estancos, no se comunican con el padre a no ser que se haga explícitamente por medio de los mecanismos disponibles, etc.

Angular CLI

Angular CLI

El intérprete de línea de comandos de Angular 2 facilitará el inicio de proyectos y la creación del esqueleto, o scaffolding, de la mayoría de los componentes de una aplicación Angular.

Angular CLI no es una herramienta de terceros, sino que nos la ofrece el propio equipo de Angular.

En pocos minutos te ofrece el esqueleto de archivos y carpetas que vas a necesitar, junto con una cantidad de herramientas ya configuradas.

Instalar node JS y NPM

Para poder instalar Angular CLI debemos

instalar previamente node js y el gestor de paquetes NPM.

- Ingresar a: https://nodejs.org/en/
- Descargar la versión recomendada

Instalar node JS y NPM

- Ejecutar el archivo descargado
- Apretar “Siguiente” o “Next”

Instalar node JS y NPM

- Aceptar términos y condiciones y apretar “Next”

Instalar node JS y NPM

- Elegir directorio de instalación

Instalar node JS y NPM

• Elegir configuraciones a instalar (Dejar todas por defecto)

Instalar node JS y NPM

• Finalmente apretar en “Install”

Instalar node JS y NPM

Una vez instalado ejecutar **node –v** (Mediante este comando podemos comprobar que versión de node tenemos instalada en nuestra PC.)

Luego ejecutar **npm –v**

Instalar Angular CLI

Para instalar angular CLI debes ejecutar en la consola lo siguiente:

Durante este proceso se instalara la herramienta Angular CLI con todas las dependencias necesarias.

ngMediante el comando ng podrás lanzar cualquiera de las acciones disponibles en

angular, por ejemplo:

ng –help

Podemos encontrar mas información sobre como utilizar angular CLI en https://cli.angular.io/

Crear nuestra app

New

El comando **ng new nombre_app** nos creara un nuevo proyecto angular 2. Lanzado este comando se creará una carpeta igual que el nombre del proyecto indicado y dentro de ella se generarán una serie de subcarpetas y archivos.

New

Serve

Angular CLI lleva integrado un servidor web, lo que quiere decir que podemos visualizar y usar el proyecto sin necesidad de cualquier otro software. Para servir la aplicación lanzamos el comando "serve".

Para lanzar la aplicación en el navegador debemos ingresar al directorio en el cual se creo la misma. Siguiendo el ejemplo anterior debemos hacer (desde línea de comandos) **cd angular-ejemplo.** Una vez dentro del directorio ejecutamos **ng serve**

Serve

La aplicación por defecto se lanzara en **http://localhost:4200/**, en caso de querer modificar el puesto debemos ejecutar **ng serve –port 4201** (o el numero de puerto que queramos configurar). Abrimos el navegador y colocamos la URL mencionada anteriormente, debemos ver algo como esto:

Estructura de Directorio

¿Qué editor utilizar?

Podemos utilizar cualquier editor, es recomendable descargar algun plugin para el mismo que nos brinde ciertas herramientas basicas para typescript. Como recomendación se sugiere usar Visual Studio Code. porque ya viene

configurado con una serie de herramientas útiles y clave para desarrollar con Angular 2 como es el “intellisense" de TypeScript.

Archivos y carpetas del proyecto

Archivos del directorio raíz: Seguro que muchos de los lectores reconocen muchos de los archivos que hay dentro, como package.json (descriptor de dependencias npm) o .gitignore (archivos y carpetas que git debería ignorar de este proyecto cuando se añada al repositorio). En resumen, todo lo que encontraremos en esta raíz no son más que archivos que definen nuestro proyecto y configuran el entorno para diversas herramientas.

Archivos y carpetas del proyecto

Archivos y carpetas del proyecto

dist: Es la versión de tu aplicación que subirás al servidor web para hacer público el proyecto. En dist aparecerán todos los archivos que el navegador va a necesitar y nunca código fuente en lenguajes no interpretables por él. (Observa que no hay archivos .ts dentro de dist). Ojo, pues muy probablemente tengas que iniciar el servidor web integrado en Angular CLI para que aparezca la carpeta "dist" en el directorio de tu proyecto.

Archivos y carpetas del proyecto

Public: Es donde colocas los archivos estáticos del proyecto, imágenes y cosas

similares que se conocen habitualmente como "assets". Estos archivos también se moverán a "dist" para que estén disponibles en la aplicación una vez subida al servidor web desde donde se va a acceder.

e2e: Es para el desarrollo de las pruebas. Viene de "end to end" testing.

tmp: Es una carpeta que no tocaremos, con archivos temporales que generará Angular CLI cuando esté haciendo cosas.

Archivos y carpetas del proyecto

node_modules: Son los archivos de las dependencias que mantenemos vía npm. Por tanto, todas las librerías que se declaren como dependencias en el archivo

package.json deben estar descargados en esta carpeta node_modules. Esta carpeta podría haber estado dentro de src, pero está colgando de la raíz porque vale tanto para las pruebas, como para la aplicación cuando la estás desarrollando.

Package.json

Es un archivo en el que se declaran las dependencias de una aplicación, gestionadas vía npm. Su código es un JSON en el que se declaran varias cosas. Inicialmente hay que ver la propiedad "dependencies". En ella encontrarás la librería Angular separada en varios módulos. Esta es una de las características de la nueva plataforma de desarrollo promovida por Angular 2, la modularización

del código. Encontrarás que una aplicación Angular 2 necesita ya de entrada diversos módulos como son "common", "compiler", "core", etc.

Package.json

Estructura de Directorio **SRC**

Index.html

De lo que es Angular 2 lo único interesante es el componente **<app-root>Loading...</app-root>.** Todo, incluida la aplicación principal, debe ser definido y declarado como un componente. De hecho definiremos las aplicaciones Angular2 como árboles de componentes. Y todo árbol debe tener una raíz. Mientras Angular no entre en fucionamiento, el usuario verá el mensaje de Loading... después la magia de Angular2 lo sustituirá por el contenido del componente app-root predefinido por el generador.

Index.html

Main.ts

Centrándonos en el código que habremos de mantener fíjate en la línea **import { AppModule } from './app/';**. Le indica a WebPack que importe el contenido de la carpeta **./app/.**

Modulo raíz

Las aplicaciones Angular2 están pensadas para crecer. Para ello es fundamental cierto grado de modularidad. Dentro del archivo **app/app.module.ts** te encontraras con algo como lo siguiente:

Modulo raíz

Un módulo no es más que una clase

contenedora. Cada módulo puede incluir múltiples componentes y servicios. Normalmente un módulo dependerá de otros. El módulo raíz declara un componente especial para el arranque de la aplicación: El componente raíz

Componente Raiz

Los componentes son los bloques de construcción de Angular 2 que representan regiones de la pantalla. Las aplicaciones se definen como árboles de componentes. Nuestra aplicación es un árbol que tiene una raíz, habitualmente llamado app y que es común a cualquier desarrollo. Cada componente a su vez está formado por tres partes: 1. **La vista**: es el código que se renderizará para los

usuarios. Esta plantilla estará en un fichero de extensión .html. 2. **La clase controladora**: En ES6 usaremos clases para declarar los controladores que exponen datos y funcionalidad a la vista. 3. **Metadata**: Se declara como un decorador, una función especial de TypeScript, que recibe un objeto de configuración. Esto acompaña al controlador en un fichero de extensión .ts

App.component.ts

App.component.ts

La función decoradora **(@component**) se encarga declara de diversas cuesiones. Una de ellas es el "**selector**" de este componente, o el nombre de la etiqueta

que se usará cuando se desee representar. Luego está asociada la **vista** (propiedad "templateUrl") al archivo .html del componente y su **estilo** (propiedad "styleUrls") a un array de todas las hojas de estilo que deseemos. En la clase del componente, que se debe colocar con un **export** para que se conozca fuera de este módulo, es la parte que representa el controlador en una arquitectura MVC. En ella colocaremos todas las propiedades y métodos que se deseen usar desde la vista

App.component.ts

App.component.html

Estas son cosas que te resultarán muy familiares como la interpolación **{{ title }}**

que permite mostrar el famoso app works!, nueva versión del hola mundo. Ya está, el resto ya es sólo usar este componente en el index.html

Vistas

Declaraciones de una vista

• **Propiedad**: Cualquier valor que podemos asignar por medio de un atributo del HTML. Ese elemento puede ser simplemente un atributo del HTML estándar, un atributo

implementado mediante el propio Angular 2 o un atributo personalizado, creado para un componente en específico.

• **Expresión**: Es un volcado de cualquier información en el texto de la página, como contenido a cualquier etiqueta. La expresión es una declaración que Angular procesará y sustituirá por su valor, pudiendo realizar pequeñas operaciones.

Declaraciones de una vista

• **Binding**: Es un enlace entre el modelo y la vista. Mediante un binding si un dato cambia en el modelo, ese cambio se representa en la vista. Pero además en Angular se introduce el "doble binding", por el cual si un valor se modifica en la vista, también viaja hacia el modelo.

• **Evento**: es un suceso que ocurre y para el cual se pueden definir manejadores, que son funciones que se ejecutarán como respuesta a ese suceso.

Flujo de información entre el modelo y la vista

El programador ahora será capaz de expresar cuándo una información debe ir del modelo hacia la vista y cuándo debe ir desde la vista al modelo. Para ello usamos las anteriores "piezas" o "herramientas" en el HTML, las cuales tienen definida de antemano un sentido para el flujo de los datos.

Flujo de información entre el

modelo y la vista

1. Las **propiedades** tienen un flujo desde el modelo a la vista. Una información disponible en el modelo se puede asignar como valor en un elemento del HTML mediante una propiedad, usando la notación corchetes. Por ej: **[propiedad]** 2. Las **expresiones** también viajan desde el modelo a la vista. La diferencia de las propiedades es que en este caso las usamos como contenido de un elemento y además que se expresan con dobles llaves. Por ej: **{{expresión}}**

Flujo de información entre el modelo y la vista

3. El **binding** (a dos sentidos, o doble

binding) lo expresamos entre corchetes y paréntesis. En este caso la información fluye en ambos sentidos, desde el modelo a la vista y desde la vista al modelo. **Por ej: [(ngBind)]** 4. Los **eventos** no es que necesariamente hagan fluir un dato, pero sí se considera un flujo de aplicación, en este caso de la vista al modelo, ya que se originan en la vista y generalmente sirven para ejecutar métodos que acabarán modificando cosas del modelo. **Por ej: (evento)**

Ejemplos

App.component.ts

App.component.html

Ejemplos

En este caso se agregar una clase css con el nombre que tenga la variable clase declarado en el modelo, en el ejemplo es css-class.

Ver ejemplo 1

Taller

Ejercicio

Con los conocimientos adquiridos hasta ahora desarrolla una página que diga **hola nombre_persona**.

Debajo del texto mencionado anteriormente se debe agregar un input, al ingresar un texto en el mismo se debe reemplazar **nombre_persona** por el texto introducido.

Debes utilizar doble bindig de manera que al completar el texto en el input el mismo se muestre automáticamente por pantalla.