1. Hola Angular

En esta primera lección del primer tema vamos a empezar por el postre, por lo más sabroso, que va a ser crear una aplicación siguiendo una receta moderna, útil y eficiente. Luego aprenderemos los comandos básicos para cocinar y servir la aplicación a nuestros usuarios de manera que estén siempre satisfechos.

Por último, y para prevenir que nada salga mal, ajustaremos nuestras herramientas para ayudarnos mientras codificamos y mientras preparamos aplicaciones **Angular** modernas.

1.1 Instalación del Angular CLI y generación de una aplicación.

Bienvenido a un viaje que te llevará desde un **servidor** hasta un **navegador** hasta un **browser**, enviándole por el camino **Javascript** con datos en formato **JSON** que harán las delicias de tus usuarios porque irá rápido y ágil.

Para realizar este viaje, necesitaremos algunos elementos y herramientas. Como no, en todo el ecosistema relacionado con **Javascript**, aparecerá **Node**. Y aunque he dicho **Javascript**, realmente programarás en **TypeScript**. No te asustes, esto es **Javascript** pero con tipos. Si vienes del **Java** o del **C#**, estarás como en casa. Familiarízate con estas iniciales: **Command Line Interface** (CLI), porque sí, vamos a trabajar bastante desde la **línea de comandos**.

Instalaremos el programa **ng** que nos permitirá generar, compilar y distribuir el código de **Angular**. Con el programa **ng** instalado en tu **línea de comandos**, en tu **CLI**, puedes empezar a invocarlo pasándole argumentos, como por ejemplo **new**, para que cree una nueva aplicación y dándole a esa aplicación un nombre.

La aplicación que usaremos durante el curso será para la gestión de reservas en actividades deportivas. Así que se llamará **Activity Bookings**. Además, podrías seguir dándole argumentos para controlar cómo se va a crear esa aplicación.

El resultado, sea como sea, va a generar unas carpetas de ficheros y uno de ellos va a ser el conocido **package.json**, que viene con todas las dependencias que necesitas para descargar **Javascript** que genere nuevas aplicaciones, que compile estas aplicaciones y que te las distribuya.

Todo lo puedes ver en tu editor de código favorito. Yo utilizaré **Visual Studio Code**. Y eso es lo que vamos a ver ahora: **Código**.

1.1.1 Crear un proyecto nuevo: Activity Bookings

Este curso es de **Angular** Moderno, y como toda cosa moderna, depende de las versiones de las herramientas que tengas instaladas y eso será lo primero que vamos a comprobar. Es esencialmente asegurarnos de tener instalado todo lo necesario, que no es más que tener una versión moderna de **Nodo**. Así que voy a escribir node -v para comprobar que tengo una versión suficiente, por ejemplo la versión 20, cualquier versión superior a esta que sea par, como la 22 o 24, serán aceptables.

Lo siguiente sería tener instalado el propio **Angular**. Para instalar **Angular**, se recomienda hacerlo de manera global mediante el comando npm de **Node Package Manager**, install, de manera global, ¿el qué? Pues del fabricante **Angular**, que esto es como se debería leer este @angular, me instalas el producto

CLI y en su última versión. De esta manera, nos aseguraremos de tener la versión de **Angular** más reciente disponible en cada momento.

Una vez terminada la instalación deberíamos comprobar qué es lo que se ha instalado y qué consecuencias tiene. Para ello voy a escribir el comando ng, que es como nos referiremos a **Angular** a partir de ahora. Este programa **ng** tiene argumentos y uno de ellos, muy sencillo, es simplemente v, que lo que hará es darnos la versión actual que tenemos instalada. Pero esto es lo mínimo, tener la versión de **Angular** actualizada con el **Node** adecuado.

Hay tablas que te dejaré en la documentación donde especifican qué versión va con cuál y cómo instalar la versión de **Node** más adecuada. Bien, pues vamos a empezar utilizando nuestra nueva herramienta **ng**, creando una nueva aplicación, new, llamándole pues algo tan genérico como my-application. Sin más, me va a empezar a preguntar cosas como con qué herramienta de **CSS** voy a trabajar, dejaremos el estándar de **CSS**, cosas que ahora mismo no nos preocupan demasiado así que le doy a la opción por defecto.

Pero lo interesante es que quiero que veas que se ha generado todo este código. Estos son carpetas, ficheros, y aquí estará el código fuente de tu aplicación más otros ficheros de configuración que después ya veremos. Además se genera un fichero **package.json** que trae o que incluye un montón de dependencias que es lo que ahora estará instalando. Estas dependencias nos servirán tanto para compilar la aplicación como para que después el resultado se ejecute en el **browser** del navegador.

Una vez terminado este proceso podría entrar a ver que efectivamente veo que aquí tengo una serie de ficheros y directorios. El generador no sólo nos permite al crear una nueva aplicación darle el nombre sino que le podemos poner argumentos para que no nos pregunte. Si yo ya sé que quiero **CSS** puedo ponerlo así. Si yo creo que voy a utilizar algo llamado **server-side rendering** que ahora aún no vamos a estudiar podría ponerlo así. De esta manera se generará lo mismo sólo que ya no nos pregunta.

Bien, de nuevo puedo entrar a ver qué es lo que se me ha generado y vuelvo a tener ahí una serie de ficheros y directorios pero de verdad aún no nos toca estudiar esto. Bueno, pero vamos a utilizar la herramienta para algo útil. Vamos a crear una aplicación que nos sirva de demostración de ejemplo durante todo el curso y que

voy a llamar **ActivityBookings** porque se va a dedicar a la gestión de reservas para actividades. A partir de aquí le pondré atributos que ya sé que me interesan como por ejemplo que utilice **server-side rendering**, ya veremos para qué vale, que utilice **CSS** como base, otras cosas que se sugieren en las aplicaciones modernas y es que incrustemos los estilos y las plantillas dentro de los componentes, esto tendrá sentido, ya lo veremos, pero por ahora simplemente quiero que veas que la generación de aplicaciones se puede modular y se puede modificar como uno quiera. Se crean recetas para crear ciertos tipos de aplicaciones. Esta es la que yo te propongo.

Y por último, un modificador muy interesante es este que he puesto aquí, **prefix**, prefijo. Normalmente se pone un acrónimo de tres letras, yo he puesto **lab** que suena un poquito a laboratorio y también a las iniciales de **activity bookings**. Bueno, pues este prefijo se va a utilizar para de alguna manera firmar todos los componentes que hagamos. Se distinguirán los componentes creados por mí del **HTML** estándar o de componentes de otros. Es muy habitual que las empresas utilicen sus propias iniciales o las iniciales del proyecto o tú, en tu caso, por ejemplo, tus iniciales como desarrollador.

Una vez terminado podemos, como siempre, entrar a ver qué es lo que tenemos ahí. Y si queremos verlo en todo su esplendor podremos abrirlo directamente con nuestro editor de referencia. Yo voy a utilizar **Visual Studio Code** y al abrirlo sobre esta carpeta, voilà, aquí lo tenemos. Nos dedicaremos ahora a analizar qué es lo que se nos ha generado, pero por ahora recordemos que **Angular** es una herramienta de **línea de comandos** que tiene potencialidad para crear aplicaciones y más adelante para compilarlas y distribuirlas.

1.1.2 Árbol de ficheros y carpetas

Vamos a darnos un paseo sobre el código generado para familiarizarnos con la estructura de una aplicación en **Angular**. Esencialmente, en la carpeta raíz tendremos una serie de ficheros de extensión **json**, donde estarán configuraciones de las herramientas que se utilicen durante el proceso de desarrollo, configuración, etc.

Todo aquello que tenga que ver con **TypeScript** estará en estos ficheros de **tsconfig**. Y en el **package.json** tendremos, como es habitual en cualquier framework **JavaScript**, el listado de las dependencias en modo ejecución. Casi todo tiene que ver con **Angular**, como es natural.

En las **Dependencies**, encontraremos lo que necesitamos durante el proceso de desarrollo, en concreto el **CLI** para generar más código y las herramientas de construcción y compilación. Por supuesto, alguna utilidad más para pruebas y el propio **TypeScript**.

En el **package.json** también aparecen pregrabados ciertos scripts conocidos, como el **npm start**, que lanzan comandos del **CLI**. Recordad que es el famoso **ng**, el programa **ng**.

Más específico de **Angular** es el fichero **angular.json**, el cual tiene un poco de metadata, el recordatorio del prefijo para todos los componentes que hagamos y, sobre todo, sirve como índice del resto de la aplicación. Por ejemplo, indica que la raíz del código fuente estará en una carpeta llamada **src** (de source) y que dentro de eso, encontraremos el **index.html** y el **main.ts** como ficheros principales.

En el **index.html** encontraremos una carta vacía en la que destaca lo que falta y lo que sobra. Lo que sobra, por ejemplo, es la etiqueta **lab-root**, que no es estándar pero tiene el prefijo **lab**, que se repetirá en todos los componentes que hagamos. Y lo que falta es todo, desde cualquier elemento **div**, **h1**, etc., o cualquier script que pudiésemos necesitar. Pero todo esto, durante el proceso de compilación, se generará para que lo pueda ver el usuario.

Ese proceso de compilación parte del fichero **src/main.ts**. Al ser un fichero **ts**, quiere decir **TypeScript**, que es **JavaScript** con los esteroides de los tipos. También, como cualquier **JavaScript** moderno, tiene la capacidad de importar otros ficheros. Estos ficheros pueden estar en una posición relativa (todo aquello que empiece por punto, barra, punto, punto, etc.) o absoluta, que lo irá a buscar a lugares bien conocidos, como la carpeta **NodeModules**.

Los que sean propios de la aplicación, hayan sido generados o programados por mí, estarán de esta manera relativamente cercana. Puedo entrar en cualquiera de ellos y veré que casi todo el código, por ahora, está en una única carpeta llamada **app**. No es el momento de estudiar el contenido de esta carpeta, sino centrarnos en que esto es nuestro hogar a partir de ahora. La carpeta **app**, dentro de **src**, tendrá el código con el que nosotros tendremos que trabajar.

1.2 Comandos básicos para depurar y ejecutar aplicaciones.

Bueno, pues poco a poco empezarás a crear tu propio código, que estará en la carpeta **src**.

Y algo, alguna herramienta habrá que lo compile a ese proceso que hace **Angular**. Le vamos a llamar **build** (construcción), y lo dejará en una carpeta de distribución.

A partir de ahí, esta carpeta de distribución, a través de cualquier mecanismo **HTTP**, llegará al usuario, al **browser** del usuario.

Muy bien, esto será el proceso de puesta en producción normal.

Pero mientras estemos desarrollando, nos va a interesar disponer de un **browser** para nosotros, los programadores, y lo que nos interesa es que esto sea muy rápido y que cambie automáticamente.

Este proceso también nos lo hace **Angular**, de hecho junta las dos interacciones anteriores en algo que llamará **serve** (servir).

Los ficheros interesantes en todo este proceso empezarán por ser el **index.html**, que de alguna manera se relaciona con el **main.ts**. Desde aquí habrá enlaces a otros ficheros **ts**.

Y un producto, que tradicionalmente era **webpack**, ahora se puede hacer con **bit**, con **bang**, etc. Una serie de herramientas los procesan y generan un **html** distinto, que ya lleva incrustadas referencias a ficheros de **javascript**.

Estos ficheros de **javascript** tendrán el código que se ejecutará en producción.

Todo estará orquestado desde un fichero de configuración llamado **angular.json**. Y tú podrás crear y lanzar los comandos de una manera automatizada desde el **package.json**.

En definitiva, el proceso, tanto de depuración mientras estás desarrollando como de puesta en marcha, está automatizado y sigue un flujo desde la fuente hacia el destino, el **browser** de tu usuario.

Veámoslo en código.

1.2.1 Compilar y servir en modo producción

Bien, pues vamos a empezar a construir algo, algo que ya viene generado, pero que lo vamos a aprovechar. En el **package.json** tenemos uno de los scripts, el script **build**, que llama al programa **ng** con el comando **build**, el cual está asociado, aquí en el **angular.json**, a estas acciones que están dentro de la sección **architect**.

En estas secciones tenemos qué podemos hacer con el código fuente: construirlo (compilarlo), servirlo (una compilación dinámica que nos la sirve), testear y hacer más cosas. Para ejecutar estos scripts, normalmente utilizo la consola integrada del **Visual Studio** y simplemente lanzo el comando adecuado. En este caso, **npm run build** lanzará al **ng build** y esto lo que va a hacer es servirse de estas

indicaciones, ejecutando un **builder**, un programa que va a mirar a estas opciones de configuración, el famoso **index.html**, el **main.ts**, etcétera.

Al ejecutar, se preguntará si queremos tener analíticas o no. Por ahora le voy a decir que no. Lo que va a hacer es recorrer todo este **main** y todas sus importaciones, y una vez que tenga todo esto empaquetado, lo soltará en una carpeta que también está configurada, que es la carpeta de distribución.

Esa carpeta de distribución creará otra carpeta con el nombre de la aplicación, en este caso **activitybookings**, y dentro de eso nos quedaremos con la sección del **browser** donde el **index.html** será muy similar al que vimos en el código fuente. Este es el fuente, este es el destino. La diferencia es que en el destino ya hay algo incrustado dentro del **body**, eso es lo que nos tocará estudiar cuando empecemos a ver **Angular** de verdad, y además hay unos scripts.

Estos scripts tienen el contenido de este **main.ts**, más todas las cosas que ha importado, todo empaquetado ahí. El resultado ya sería directamente ejecutable por un servidor.

Para hacerme la vida más fácil mientras estoy desarrollando y tener algo que me muestre el resultado, no justo durante el desarrollo, sino al terminar alguna acción concreta, utilizo **npx** para ejecutar cosas sin tener una instalación per se, y utilizo **http-server**. Este **http-server** toma como entrada la carpeta en la que está el código, le dices que no utilice caché, es decir, que no recuerde nada, e inmediatamente te abrirá una pantalla, un navegador, en el puerto 8080, con el resultado de lo que acabamos de ver. Es decir, nos muestra el resultado de esta compilación.

1.2.2 Ejecutar y actualizar aplicación en modo desarrollo

Vamos a ver ahora el siguiente comando que viene dentro de la sección **architect**, que es el **serve**. **Serve** realmente llama al **Builder** y después nos lo autosirve, de ahí su nombre, con un programa especializado en detección de cambios y recarga de la aplicación según el programador va cambiando cosas. Esto es lo que nos va a hacer sentir un poquito programadores.

Para ello, lanzo directamente **npm start**, en este caso no hace falta el **run**, que a su vez ejecuta **ng serve**.

Se produce una compilación e inmediatamente se debería abrir, si no hacemos clic aquí, la aplicación mostrándome el estado actual de la ejecución.

A partir de ahora voy a colocar a la vez el código fuente, que está ahora mismo un poquito ampliado de más. Vamos a reducir un poquito el tamaño y aquí veo el

resultado de la ejecución y puedo ver que este contenido de **Welcome to Activity Bookings!** de algún sitio vendrá, del **index.html** no, pero siguiendo el rastro del **main** veo que me lleva a algo llamado **app.component**. Ahora mismo aún no estamos aprendiendo **Angular**, pero sí quiero que veáis este tema.

Aquí tengo una plantilla que dice que cuando se solicita el componente **lab-root** se presente este **h1** con un texto que pone **Welcome to Title** y una admiración. Me parece que una admiración es poco y le voy a poner otras dos.

En cuanto guarde, se producirá una cosa curiosa que es que aquí habrá una recompilación y, al mismo tiempo, aquí una recarga de la página, con lo cual en vivo y en directo podré ver que esto está funcionando. Así que yo puedo ponerle aquí un **Angular works!**, guardamos y esto funciona.

Este comportamiento automático yo lo mejoro cambiando un poquito la secuencia de entrada, poniendo un comando especial que lo que hace es abrirme inmediatamente el **Chrome** en modo incógnito. De esta manera no tiene un recuerdo de otras veces, el **autoopen** con las herramientas, con las **DevTools** automáticas también es una mejora.

Todo esto lo meto a un script llamado **Chrome** y lo lanzo desde el **start**. Cambio un poquitín el **start**. Bueno, estas son pequeñas mejoras.

Además, por ejemplo, fuerzo a que el compilador sea **esbuild** en lugar del **Webpack**, lo cual también acelera un poquito el desarrollo y la recarga automática.

Veréis que ahora cuando lance el **npm start** se me debería abrir un **Chrome**, ahí está, en modo incógnito, con las herramientas ya desplegadas, las herramientas de desarrollador ya desplegadas.

Ahí lo tenéis. A partir de este momento yo puedo trabajar con la comodidad y la tranquilidad de estar siempre en una ventana nueva que no recuerda nada de lo anterior y con el **Chrome** siempre abierto ya con las **DevTools**.

1.3 Configuración del espacio de trabajo, editor y herramientas recomendadas.

Pues es hora de afinar un poquito las herramientas. Como yo os decía, yo uso **Visual Studio Code**, que es extensible. Es decir, que tú le puedes agregar extensiones, de las cuales hay una esencial que te recomiendo que uses, que se llama **Angular Essentials**.

Con ese nombre va a ser fácil recordarla, porque ya te instala una serie de extensiones necesarias para que tu aplicación sea cómoda de tratar en el **Visual Studio**. Concretamente, hay tres:

1. **Angular Language Service**, que se encarga de validar el código de **Angular** según lo escribes.
2. **ESLint**, que se encarga de verificar que el código tiene una buena pinta.
3. **Prettier**, que se encargará de que luzca bien dentro del **Visual Studio**.

Las extensiones te ayudan, pero obviamente necesitas también hacer algún tipo de instalación por debajo. Todo esto lo tienes en la documentación y lo verás durante la demostración. **Prettier** y **ESLint** funcionan bastante bien ya out of the box. **ESLint** verifica que tu código tiene buena pinta y **Prettier** lo pone bonito, pero a veces entran en conflicto sobre quién se encarga de qué cosas.

Así que yo te recomiendo que instales unas herramientas extra más unas configuraciones. Los ejemplos te quedarán en los ficheros de **extension.config** para que veas que funciona. Además de eso, puedes buscarme en **GitHub** y en **Medium**, donde tengo unas configuraciones llamadas **.files** y unos artículos sobre **Visual Studio Code** donde explico configuraciones que a mí me gustan.

Por supuesto, además hay temas sobre temas de colores, claro que sí. Vamos a ver el código.

1.3.1 *Angular Essentials y DotFiles*

Para finalizar, vamos a hablar un poco de la herramienta. Yo utilizo **Visual Studio Code**. Podéis utilizar cualquier otro, por supuesto.

Dentro de **Visual Studio**, a veces me preguntan por qué extensiones se recomiendan. Bueno, pues aquí es fácil. Hay una extensión llamada **Angular Essentials**, que va por su versión 16, aunque es perfectamente compatible con esta versión 17, que ya tiene todo lo necesario o lo más utilizado por los programadores profesionales de **Angular**.

Así que, bien, esto es lo que os recomiendo, que os instaléis el **extension pack**, que son 8 extensiones esenciales para programadores de **Angular**.

Más allá de eso, y en cuanto a configuración del **Visual Studio** y demás, os dejaré por la documentación el enlace a este repositorio que tengo en **GitHub**. Se llama **https://github.com/AlbertoBasalo/dotfiles**, porque se refiere a los ficheros **.dot** que están en la raíz de cualquier proyecto. Aquí podréis ver cuáles son los settings que le pongo a mi **Visual Studio**. Os pueden servir de inspiración.

También la lista completa de las extensiones que suelo utilizar y para qué valen. También, de nuevo, una inspiración.

Y, si queréis profundizar algo más, aquí hay enlaces a artículos que he escrito sobre cómo configurar el **Visual Studio**, qué cosas hacer con él...

En fin, cómo configurar, por ejemplo, **Prettier** y **ESLint**, que son dos herramientas que merecen que le dediquemos unos minutitos aparte.

1.3.2 EsLint y Prettier

Vamos a centrarnos ahora en dos herramientas que, si bien sus extensiones ya están instaladas, que son **ESLint** y **Prettier**. Una está dedicada a que el código quede bonito y la otra más a la calidad en sí del código, pero se pueden configurar específicamente para **Angular**.

Para ello, yo os recomiendo que agreguéis, mediante un comando de **ng**, que tiene también la posibilidad de instalar y configurar vuestra aplicación, las herramientas para **ESLint**. De esta forma, no sólo nos aseguraremos de tener **ESLint** instalado en este proyecto, sino que nos creará ya un fichero de configuración.

Al mismo tiempo, también nos permitirá lanzar scripts que prueben la calidad de nuestro código de esa manera estática que hace **ESLint**. Y esto es lo que habrá ocurrido: ha aparecido y se ha autoconfigurado como un nuevo script y un nuevo comando dentro de **angular.json**. Aparte del **build**, **serve**, etc., también tendremos el **lint**, que herramientas usa, contra qué ficheros, etc.

Tenemos aquí también la preconfiguración hecha para **ESLint**. ¿Esto es suficiente? Sí, pero se puede mejorar si agregamos una serie de paquetes que están en **npm** y que me ayudan con **Prettier**. **Prettier** a veces entra en cierto conflicto con **ESLint** y estos paquetes hacen que funcionen los dos en buena sintonía.

Aquí os dejo el fichero **Prettier** con la configuración que suelo poner en todos mis proyectos, aunque en este screencast seguramente con el ancho de 100 sea suficiente. 120 es lo que suelo poner en condiciones normales.

También voy a sobreescribir con algún cambio menor que incluye la configuración de **ESLint** con **Prettier** en el fichero **eslintrc.json**. Os queda el ejemplo puesto con ambos ficheros de configuración y la instalación tanto de las extensiones de **Visual Studio** como de los paquetes que se necesitan. **ESLint** para que el código sea de mejor calidad y **Prettier** para que quede bonito.