

# Herramientas de desarrollo

- [Git for Windows](#)
- Servidor JavaScript [Node.js](#)
  - Gestor de paquetes npm, incluido en Node.js
  - [Angular CLI](#): generador de aplicaciones plantilla para Angular 2
  - Instalador de paquetes [yarn](#)
- Editores
  - [Atom editor](#)
  - [Visual Studio Code](#)
- Entornos visuales de desarrollo
  - [Angular IDE](#)
  - [Plugin Angular 2 para Eclipse Neon](#)
  - [Visual Studio Community Edition](#)

# Entorno de desarrollo. Pasos a seguir

- Instalar un cliente para Git
  - Configurar el cliente si estamos usando un proxy
- Instalar Node.js
  - Configurar el gestor de paquetes npm si estamos usando un proxy
- Instalar el gestor de paquetes yarn (opcional)
  - Configurar el gestor de paquetes yarn si estamos usando un proxy
- Instalar typescript: `npm install -g typescript`
- Instalar angular-cli: `npm install -g @angular/cli`
- Instrucciones para la [configuración](#) mínima de las herramientas

# Introducción a TypeScript

- TypeScript (a partir de ahora ts) es un superconjunto de JavaScript, elegido oficialmente por Angular 2
- La herramienta tsc (TypeScript Compiler) traduce a JavaScript estándar (es3, es5, es6) las extensiones del lenguaje
- La forma más simple de trabajar con TypeScript es instalar Node.js, y mediante el gestor de paquetes npm:
  - npm install -g typescript
- Podemos experimentar desde el playground sin instalar nada en absoluto
- Los ejemplos oficiales de typescript muestran numerosos casos de uso
- En un proyecto real, lo normal es que se configure el tsc mediante el archivo **tsconfig.json**. Para compilar un archivo ts: **tsc archivo.ts**

# Introducción a TypeScript (características)

- Tipos de datos
- Disponibilidad de var, let y const. Ver esta discusión
- Interfaces
- Clases
- Funciones
- Genéricos
- Tipos Enumerados

# Introducción a TypeScript (características)

- Inferencia de tipos
- Compatibilidad estructural de tipos
- Símbolos
- Tipos intersección y unión
- Iteradores
- Módulos
- Espacios de nombres
- Módulos y espacios de nombres

## Introducción a Angular 2

- Es un framework soportado por Google para crear aplicaciones HTML basadas en JavaScript o lenguajes como TypeScript que compilan a JS
- Se ocupa de ofrecer una solución completa para el lado cliente
- Proporciona mecanismos sencillos para comunicarse con el lado servidor, que puede estar implementado en cualquier tecnología
- A menudo se utiliza para crear apps de una sólo página (SPA, [Single Page Application](#))
- Actualmente, una de las más utilizadas
- Sus elementos fundamentales son: módulos, componentes, servicios, routers, directivas y tubos

# Introducción a Angular 2

- Ver la [arquitectura](#)
- Para crear una app en Angular 2
  - creamos plantillas HTML decoradas con directivas y tubos
  - escribimos componentes para gestionarlal
  - encapsulamos la lógica de negocio y el acceso al servidor en servicios
  - organizamos todo lo anterior empleando módulos
- Para ejecutar la app, cargamos su módulo raíz y la desplegamos en un servidor
- Este proceso está automatizado siempre y cuando usemos las herramientas de desarrollo previstas por Angular
- En el repo, ver: **angular\ejemplos\ejemplo0001**

# Ejercicio 0001

- Crear un proyecto nuevo usando angular-cli
  - Primero, crear sólo la estructura de carpetas: `ng new nombre --skip-install`
  - Después instalar las dependencias con npm o yarn
  - Segundo, crear el proyecto de la manera estándar: `ng new nombre`
- Sugerencia:
  - Examinar la [Guía](#) de la herramienta Angular-cli



# Módulos I

- Una app Angular está conformada por módulos
- Debe existir un módulo principal para que la herramienta sepa cómo arrancar la app
- Un módulo Angular se decora con @NgModule
- Como veremos más adelante, una app puede tener tantos módulos como queramos
- En el repo, ver: [angular\ejemplos\ejemplo0001](#)

# Componentes

- Las apps Angular se basan en [componentes](#), los cuales residen dentro de módulos
- Los elementos cruciales de un componente son:
  - Su decorador: `@Component`
  - Selector: nombre arbitrario que se emplea para invocar al componente
  - Su plantilla ([template](#)): básicamente el HTML del componente
  - Los estilos CSS que emplea el componente
  - Una clase TypeScript que encapsula el [estado](#) y [comportamiento](#) del componente
- La clase TypeScript debería existir siempre. Sin embargo, la plantilla y el CSS puede colocarse en el mismo archivo que la clase o en archivos externos
- En el repo, ver: [angular\ejemplos\ejemplo0002](#)

## Ejercicio 0002

- Sobre el proyecto generado en el ejercicio 0001, añadirle un componente. Probar de las dos maneras, todo en la clase y en archivos distintos.
- Para crearlo, en el directorio del proyecto, `ng g component nombre-del-componente`
- Sugerencia:
  - En el repo, ver: `angular\ejemplos\ejemplo0002`

# Servicios

- En una app real (compleja) es costumbre encapsular el acceso a datos y todo tipo de utilidades empleando servicios
- Un servicio típicamente define métodos y se decora empleando @Injectable
- Si el servicio se asocia al componente que lo usa a través de su propiedad “providers”, cada vez que se instancia el componente se instancia también su servicio
- Para que un servicio sea un singleton real para un módulo, es mejor declararlo en la propiedad “providers” de app.module.ts
- En el repo, ver: **angular\ejemplos\ejemplo0003 y 04**

## Ejercicio 0003

- Sobre el proyecto generado en el ejercicio 0002, añadirle un servicio y asociarlo a un componente. Crear un segundo servicio y asociarlo al módulo
- Sugerencia:
  - En el repo, ver: [angular\ejemplos\ejemplo0003 y 04](#)

# Asincronía I

- Una operación de acceso a datos en el lado servidor podría tardar en completarse un tiempo apreciable
- Probablemente no nos interesa que la página esté bloqueada mientras tanto
- La forma tradicional de resolver esto (perfectamente válida) es emplear AJAX
- Sin embargo, en Angular 2, se aconseja el uso de promesas y/o observables

# Asincronía II

- Una [promesa](#) representa una computación en curso, que en algún momento finalizará con éxito (entregando un valor) o fracaso
- En el repo, ver: [angular\ejemplos\ejemplo0005](#)
- Un [observable](#) representa una [secuencia de datos](#) que se genera de forma asíncrona
- La idea es subscribirse a un observable para ser automáticamente notificado cuando un dato nuevo ha llegado
- En el repo, ver: [angular\ejemplos\ejemplo0006](#)

## Acceso al servidor

- Angular 2 ofrece un cliente HTTP para acceder al lado servidor
- Se nos ofrece un api simulada en memoria para hacer pruebas
- Es posible usar promesas y/o observables, aunque Angular aconseja emplear observables
- En el repo, ver: `angular\ejemplos\ejemplo0007` para el uso de promesas
- En el repo, ver: `angular\ejemplos\ejemplo0008` para el uso de observables
- Naturalmente, hemos de acceder a servidores reales.
- En el repo, ver: `angular\ejemplos\ejemplo0009 y 10`



# Ejercicio 0005

1. Vamos a emplear un [Api para pruebas](#) disponible en la red
2. Acceder a esta url desde el navegador:  
<https://jsonplaceholder.typicode.com/posts/1> y ver el resultado
3. Crear un nuevo proyecto: `ng new nombre-del-proyecto`
4. Crear un servicio: `ng g service Usuario`
5. Empleando el objeto “http” predefinido por Angular, realizar una petición “get” a la url del punto 2 y mostrar los datos recibidos por consola y en la página del componente principal de la aplicación (app.component.html)
6. Sugerencia: en el repo, ver: `angular\ejemplos\ejemplo0009 y 10`

# Interactividad I

- La interactividad tiene lugar entre componentes (javascript) y [plantillas](#) (HTML, CSS, [eventos](#))
- Existen diferentes maneras y sintaxis de hacer esto
- Uso de estilos, enlace de propiedades y eventos
  - En el repo, ver: [angular\ejemplos\ejemplo0011 y 12](#)
- En una aplicación real puede ser importante no sólo crear componentes, sino hacerlos reutilizables, normalmente empleando propiedades de tipo input
  - En el repo, ver: [angular\ejemplos\ejemplo0013](#)

## Interactividad II

- Para controlar los datos que se ven y actualizan dinámicamente en una vista, existe la posibilidad de emplear [directivas](#) estructurales y [basadas en atributos](#) estándar de Angular y/o programar directivas personalizadas: ver [angular\ejemplos\ejemplo0014](#)
- A veces es necesario filtrar (transformar) los datos que van a mostrarse en un template.
- Para ello, Angular recomienda el uso de [tubos](#) (pipes), predefinidos y/o personalizados: ver [angular\ejemplos\ejemplo0015](#)
- También es posible emplear elementos externos en componentes mediante ng-content: ver [angular\ejemplos\ejemplo0016](#)

## Ejercicio 0004

- Crear un nuevo proyecto y añadir un nuevo tubo personalizado
  - ng g pipe concatenar
  - El tubo opera sobre un string y recibe como parámetro una nueva string que concatena a la primera
- Sugerencia:
  - En el repo, ver: [angular\ejemplos\ejemplo0015](#)

# Formularios I

- En Angular hay dos formas de crear formularios:
  - Basados en [plantillas](#) (Template Forms)
  - Basados en [modelos](#) (Reactive Forms)
- Los primeros son más cómodos, los segundos más potentes, especialmente en lo relacionado con las validaciones.
- En los que se basan en plantillas, es Angular el que crea los elementos visuales automáticamente (a partir de la plantilla)
- En los que se basan en modelos, como nosotros los que creamos programáticamente desde JavaScript los elementos visuales

## Formularios II

- Formularios basados en plantillas
- Enlace entre propiedades en ambos sentidos (Two-way binding).
- En el repo, ver: [angular\ejemplos\ejemplo0017](#)
- Validaciones y envío condicional de formularios
- En el repo, ver: [angular\ejemplos\ejemplo0018](#)

# Ejercicio 0006

- En este ejercicio, no usaremos CSS. Crear un nuevo proyecto: `ng new ...`
- Añadir un nuevo componente: `ng g nombre-form`
  - En la plantilla del componente, crear un formulario con una “label”, un “input type='text'...” y un botón de enviar (“input type='submit'...”). El formulario debe usar un `ngSubmit`
  - Al pulsar el botón de enviar, mostrar un mensaje por consola con el contenido del campo de texto. Sugerencia: emplear `ngModel`
  - Fuera del formulario, mostrar el valor del campo de texto. Sugerencia: emplear un enlace unidireccional, del componente a la página
- Sugerencia: en el repo, ver: `angular\ejemplos\ejemplo0017`

# Formularios III

- Formularios basados en modelos
  - Creación de controles para el formulario.
  - En el repo, ver: [angular\ejemplos\ejemplo0019](#)
  - Validaciones personalizadas y uso de Form Builder
  - En el repo, ver: [angular\ejemplos\ejemplo0020](#)
  - Encapsulación del acceso a datos con servicios, validación “global” al enviar el formulario
  - En el repo, ver: [angular\ejemplos\ejemplo0021](#)



# Enrutamiento y Navegación I

- Angular está muy preparado para permitirnos crear aplicaciones de una sólo página (**S**ingle **P**age **A**pplication, SPA)
- La idea es muy sencilla: a medida que el usuario interacciona con la página, los contenidos de la misma cambian
- Para poder hacer esto, es necesario emplear el Router de Angular2
- En el repo, ver: [angular\ejemplos\ejemplo0022](#)

## Enrutamiento y Navegación II

- Cuando tenemos una SPA con N componentes, es esencial saber cómo pasar información (parámetros) entre ellos
- En el repo, ver: [angular\ejemplos\ejemplo0023](#)
- A veces existen condiciones para pasar de una vista a otra. Angular ofrece vigilantes (guards) para asegurarnos de que dichas condiciones se cumplan
- En el repo, ver: [angular\ejemplos\ejemplo0024](#)

# Ejercicio 0007

- Crear un nuevo proyecto preparado para usar el router: `ng new nombre-del-proyecto --routing`
  - Añadir tres componentes nuevos (`ng g component nombre`), llamados primero, segundo y no-encontrado
  - Configurar el archivo `app-routing.module.ts` con las rutas asociadas a cada componente
  - Colocar en `app.component.html` los enlaces para las rutas (`routerLink`) y definir la zona donde Angular debe mostrar bajo demanda los componentes (`router-outlet`)
- Sugerencia: en el repo, ver: `angular\ejemplos\ejemplo0022`

# Aplicaciones CRUD

- Es muy común que una aplicación interactúe con una base de datos alojada en servidor, y lleve a cabo con ella operaciones de tipo **Create**, **Read**, **Update** y **Delete**
- Para ejemplificarlo, hemos elegido emplear [Firebase](#), de Google y una biblioteca para facilitar la conexión: [Angularfire2](#)
- Creación de la base de datos, configuración del proyecto e importación de algunos datos de prueba en formato JSON
- En el repo, ver: [angular\ejemplos\ejemplo0025 y 26](#)

# Aplicaciones Multimódulo

- La aplicación Angular que genera [angular-cli](#) tiene un sólo módulo, el cual es precargado cuando nuestra aplicación se ejecuta
- De la misma forma que un módulo puede incluir N componentes, una aplicación puede constar de N módulos
- En el repo, ver: [angular\ejemplos\ejemplo0028 y 29](#)
- Los módulos de nuestra app puede precargarse todos en modo eager, como en los dos ejemplos anteriores, o bajo demanda en modo lazy
- En el repo, ver: [angular\ejemplos\ejemplo0030](#)