**Requisitos**

**Conocimientos**

* Conocimiento de JavaScript básico es necesario.
* Conocimiento básico de la estructura de páginas HTML

**Software**

* **NodeJS**
  + https://nodejs.org/es/
* **Google Chrome (Recomendado**
  + https://www.google.es/chrome/browser/desktop/
* **TypeScript**
  + http://www.typescriptlang.org/
  + npm install -g typescript
  + Verificar que versión sea superior a la 1.0
* **Angular CLI**
  + npm install -g @angular/cli
  + https://cli.angular.io/
  + https://github.com/angular/angular-cli
* **Editores recomendados**
  + Atom
    - https://atom.io/
  + Visual Studio Code
    - https://code.visualstudio.com/
  + StackBlitz (editor Online)
    - https://stackblitz.com/angular
* **Plugins**
* **Visual Studio Code**
  + Angular 2 TypeScript Emmet
  + Angular X Snippets – TypeScript, Html, Angular Material...
  + Angular Language Service
  + Angular Snippets (VX)
  + Angular2-inline
  + Bootstrap 4 & Font Awesome snippets
  + HTML CSS Support
  + JavaScript (ES6) code snippets
  + JS-CSS-HTML Formatter
  + JSHint
  + Material Icon Theme
  + Prettier – Code Formatter
  + Terminal
  + TSLint
  + TypeScript Hero
  + TypeScript Importer
* **ATOM**
  + Angular 2 Type Script Snippets
  + Atom Bootstrap3
  + Atom Typescript
  + File Icons
  + Platformio Ide Terminal
  + V Bootstrap4

**Herramientas**

* https://quicktype.io/

**Conceptos**

**ECMAScript**

ECMAScript específicamente es el estándar que a partir del año 2015 a la actualidad se encarga de regir como debe ser interpretado y funcionar el lenguaje JavaScript, siendo este (JS – JavaScript) interpretado y procesado por multitud de plataformas, entre las que se encuentran los navegadores web, NodeJS u otros ambientes como el desarrollo de aplicaciones para los distintos sistemas operativos que actualmente existen en el mercado. Los responsables de dichos navegadores y JavaScript deben encargarse de interpretar el lenguaje tal como lo fija ECMAScript.

**TypeScript**

En resumen, TypeScript se define como una especie de superset de JavaScript, cuyo resultado final es un código de JavaScript.

**Compilar TS:** tsc [nombre del archivo .ts]

**Crear un componente básico con función anónima autoinvocada**

(function() { })();

**Crear archivo de configuración TS (tsconfig.json):** tsc --init

**TS en modo listener (cambios detectados automáticamente):** tsc -w

**Definir variables**

* **let [nombre variable]: [tipo de dato]:** Genera variables diferentes por ámbito.
* **var [nombre variable]: [tipo de dato]:** No genera variables diferentes por ámbito.

**Definir constantes**

* **const [nobre variable]:** Ocupa menos memoria.

**Excluir directorios de compilación:** Agregar "exclude": [[Nombre directorio]]

**Templates literales:** `${[nombre variable o función]}`

**Funciones de flecha**

1. const miFuncion6 = (a: number, b: number) => a + b;
2. No modifican el scope o alcande de las variables en caso de ser una función interna a un objeto.

**Promesas**

Las promesas representan un resultado eventual de una operación asincrónica, la primera manera de interactuar con una promesa o promise es a través del método then el cual registra el callback que recibirá la respuesta o la razón por la cual la promesa no ha podido ser cumplida.

**Decoradores en TypeScript**

Los decoradores (decorators en inglés) permiten añadir anotaciones y metadatos o cambiar el comportamiento de clases, propiedades, métodos, parámetros y accesors. Con TypeScript podemos usarlos activando la propiedad experimentalDecorators del tsconfig.json de nuestro proyecto o si decidimos compilar mediante el comando tsc, colocar siempre la opción de --experimentalDecorators ---target ES5.

**Types e interfaces**

Sirven para definir un contrato que puede extenderse.

**Importación**

Incluir clases desde orígenes externos. Las clases deben tener la palabra reservada **export**.

**Backticks (``)**

Permiten trabajar con textos largos y con saltos de línea.

**Referencia a objetos**

1. Con doble llave {{}} para el valor.
2. Con corchete en atributo [].

**Operador Spread (…)**

Sirve para “extender” los elementos de un arreglo u objeto.

**Angular**

Angular es un framework de front-end impulsado por Google. Creado para desarrollar aplicaciones web, aplicaciones móviles o realizar procesos del lado del servidor utilizando NodeJS.

**Características**

* Separar frontend y backend de la aplicación
* Simplifica el código
* Sigue el patrón MVC
* Está basado en componentes
* Es de código abierto
* Desarrollo rápido de aplicaciones
* Modularización
* Fácil mantenimiento
* Multiplataforma

**Directivas estructurales**

Las directivas estructurales corresponden a elementos en el HTML que permiten añadir, manipular o eliminar elementos del DOM.

**Tipo componente**

* Angular
  + @Component
  + @Module
  + Personalizadas (Custom)
    - Creadas específicamente
* Tipo estructura
  + ngIf-else-then
  + ngFor
  + ngSwitch
  + ngPlural
  + ngTemplate
  + ngComponentOutlet
* Tipo atributo
  + ngClass
  + ngStyle
  + ngModel

**Componente**

Un componente en Angular es un bloque de código re-utilizable, que consta básicamente de 3 archivos: un CSS, un HTML (también conocido como plantilla o en inglés, template) y un TypeScript. La carpeta app con la que viene Angular por defecto es un componente, aunque un tanto especial.

**Ciclo de Vida (Lifecycle)**

* **ngOnChanges:** se llama cuando cambia el valor de la propiedad de entrada de enlace de datos
* **ngOnInit:** llamado después del primer ngOnChanges
* **ngDoCheck:** método personalizado para detectar y procesar cambios de valor
* **ngAfterContentInit:** se llama después de que se inicializa el contenido del componente
* **ngAfterContentChecked:** llamado cada vez que el componente verifica el contenido
* **ngAfterViewInit:** se llama después de que se inicializa la vista correspondiente del componente
* **ngAfterViewChecked:** llamado cada vez que el componente verifica la vista
* **ngOnDestroy:** llamado antes de que se destruya la instrucción

**Interpolación de texto**

Permite incorporar valores dinámicos de texto en la aplicación.

**Angular CLI**

Angular CLI es la forma más cómoda para empezar a desarrollar aplicaciones web, móvil con Angular 2+, es una herramienta de línea de comandos que facilita la creación, generación, ejecución, testing, deploy.

**Comandos más utilizados**

**Crear aplicación angular:** ng new <nombre aplicación>

**Instalar paquetes:** npm install

**Ejecutar aplicación:**

* ng serve -o
  + -o Para abrir automáticamente el navegador
* npm start
  + Cuando hay diferencia con la versión de angular CLI actual

**Crear componente**

Generar un componente por defecto y lo agrega al app.module.ts, en caso de haber otro módulo en el mismo nivel del app.module.ts se utiliza --module=<módulo>

* ng g c components/<nombre>
  + g: generate, c: component
  + ng g c components/<nombre> -it -is --flat --skip-tests --module=<módulo>
  + it ó t: inline template (código HTML dentro del ts)
  + is ó s: inline style (estilo CSS dentro del ts)
  + flat: sin crear carpeta
  + skip-tests: omitir archivos spec de pruebas
  + --module=<módulo>: agregar componente a un módulo determinado

**Crear pipe**

Generar un pipe por defecto y lo agrega al app.module.ts

* ng g p pipes/<nombre>

**Crear directiva**

Generar una directiva por defecto y lo agrega al app.module.ts.

* ng g d directives/<nombre>

**Crear módulos**

Generar un módulo por defecto y lo agrega al app.module.ts, se los agrega en los imports del app.module. Contiene ngModule (declaraciones y configuración) y CommonModule (operaciones básicas, ngIf, ngFor, etc.). Los módulos sirven para agrupar recursos.

* ng g m <nombre>

**Crear servicio**

Generar un servicio por defecto y lo agrega al app.module.ts dentro de providers. Usualmente globales y singleton.

* ng g s <nombre>

**Crear guard**

Generar un guard por defecto y lo agrega al app.module.ts

* ng g g <nombre>

**Pipes**

Los pipes son una herramienta de Angular que nos permite transformar visualmente la información, por ejemplo, cambiar un texto a mayúsculas o minúsculas, o darle formato de fecha y hora.

* uppercase y lowercase
* Slice: 1 o 2 parámetros para suprimir caracteres desde o desde - hasta.
* Decimal
* Percent
* Currency
* Json
* Async
* Date

**Localization**

Agregar funciones de localización para traducir nombres, como fechas, al idioma especificado.

**Bindeo de valores (binding)**

El enlace de datos mantiene su página actualizada automáticamente según el estado de su aplicación. Utiliza el enlace de datos para especificar cosas como el origen de una imagen, el estado de un botón o los datos de un usuario en particular.

* [<value>]="<variable>"

**Bindeo de eventos (binding)**

La vinculación de eventos le permite escuchar y responder a las acciones del usuario, como pulsaciones de teclas, movimientos del mouse, clics y toques.

* //Plantilla HTML
  + [<evento>]="<funcion($event)>"
* //TS
  + <Tipo de elemento HTML(<HTMLInputElement>)>event.target.<propiedad>

**Bindeo en dos vías (binding)**

El enlace bidireccional proporciona a los componentes de su aplicación una forma de compartir datos. Utilice el enlace bidireccional para escuchar eventos y actualizar los valores simultáneamente entre los componentes principal y secundario.

* [(ngModel)]="<variable>"
  + [()] se denomina banana in box (sugar syntax)

**Rutas (routes)**

Permite el manejo de la navegación mediante la implementación de rutas virtuales dado que un SPA solo tiene el index.html.

**Hashtag (#)**

Define un nombre a un elemento HTML.

**Formularios**

**Formularios basados ​​en plantillas (Template Forms)**

* FormsModule
* Por convención ubicar primero los tags html y luego angular.
* Se recomienda para formularios cortos y sin bindeo.
* Utilizan el “FormsModule”.
* Son de naturaleza asincrónica.
* La mayor parte de la lógica se basa en la plantilla (HTML).
* Para linkar el form HTML con el componente se utiliza: #<nombre>="ngForm" y se envía como parámetro <nombre> al método que va a procesar el formulario.

**Propiedades principales:** Cada propiedad permite evaluar o recibir resultados o valores además de generar automáticamente clases en el control de acuerdo a los resultados.

* **valid:** Cuando el control es válido.
* **invalid:** Cuando el control es inválido.
* **pending:** Cuando el control está pendiente por una acción asíncrona.
* **pristine:** Cuando el control no ha sido editado de ninguna manera.
* **touched:** Cuando el control ha sido accesado.
* **untouched:** Cuando el control no ha sido accesado.

**Formularios reactivos (Reactive Forms)**

* ReactiveFormsModule
* Se recomienda para formularios extensos y con bindeo.
* Utilizan “ReactiveFormsModule”.
* Son de naturaleza sincrónica en su mayoría.
* La lógica reside principalmente en el componente (TS).

**Decoradores**

* **@Input:** Permite a un componente padre actualizar información en un componente hijo. [Leer más](https://angular.io/guide/inputs-outputs)
* **@Output:** Permite a un componente hijo enviar información a un componente padre.
* **@HostListener:** Permite escuchar eventos del DOM.

**Despliegue (deploy)**

**Despliegue para pre-producción:** Genera el dist (distribuible) con funciones de depuración.

* ng build

**Despliegue para producción:** Genera el dist (distribuible) sin funciones de depuración. Crea un sufijo (hash) en los archivos JS para evitar que se almacene en caché del navegador.

* ng build --prod
* **Consideraciones**
  + En environment.ts cambiar a true la llave production en caso de no tener variables de producción definidas en environment.prod.ts
  + En AppModule.ts agregar useHash: true en las rutas para que el host no crea que son rutas físicas
  + En index.html quitar el / de base href para que utilice las rutas relativas.