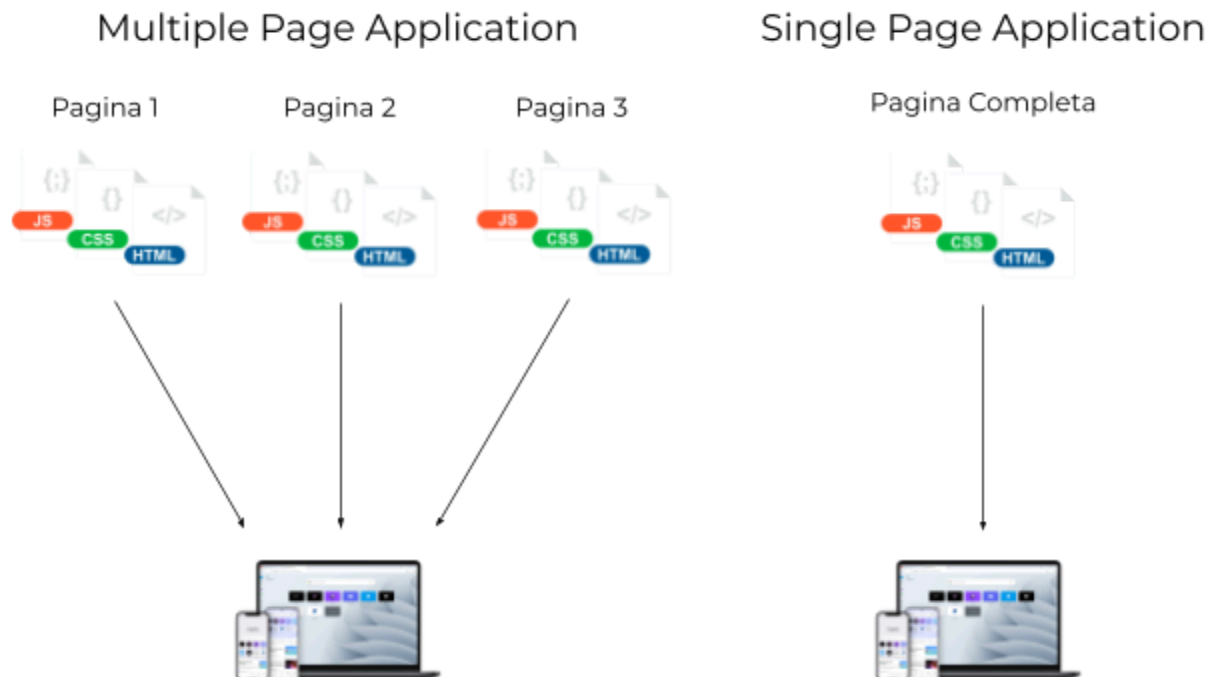


# Single page application - SPA

---

Al ingresar a una página web, el servidor envía un archivo HTML para la visualización de la misma. Al acceder a las demás secciones de la página, un nuevo archivo llega desde el servidor y el browser se refresca.

Una Single Page Application (SPA) es una página web la cual no requiere que el servidor envíe un archivo HTML por cada página, si no que envía un solo html durante la carga inicial y el contenido se actualiza dentro del HTML renderizado.



Una vez el contenido de la página web de tipo SPA es cargado, se va actualizando para reflejar los cambios dentro de la página. Esto implica que el browser no recarga la página y la navegación dentro de la misma es mucho más rápida.

## Algunos beneficios de las SPA:

---

- Mejor experiencia de usuario al navegar por la página.
- Reusabilidad de contenido.
- Facilidad de desarrollo para aplicaciones complejas.
- Facil debugging.

Pueden encontrar más información [aquí](#).

## Frameworks

---

Durante el desarrollo de una página web nos encontramos con la tarea de definir una estructura general, organización de las páginas que componen nuestra web y otros aspectos más complejos como seguridad.

Un Framework es una especie de plantilla, esquema o estructura conceptual con la cual podemos definir muchos de los aspectos anteriormente mencionados de una manera mucho más sencilla. De esta forma, se evitan posibles errores de programación y se facilita la tarea de desarrollo.

Entre los frameworks populares actualmente podemos destacar por ejemplo Angular, React o Vue.

## Angular

---

Angular es un framework de Typescript de código abierto para el desarrollo de interfaces de usuario. Es desarrollado por Google y su objetivo principal es desarrollar [SPAs](#).

Tal como lo hace React, basa su arquitectura en componentes reutilizables que nos permiten crear pequeñas partes de la web a desarrollar, como un botón de compra, un header o una sección de información, para ser reutilizadas en cualquier lugar que se requiera.

# Instalación

---

Para instalar el angular CLI (Command Line Interface o interfaz de líneas de comando), podemos hacerlo desde NPM

Para ello abriremos una consola:

Inicio > Buscamos “CMD” > Hacemos click en “Símbolo de sistema”

Allí ejecutaremos el siguiente comando:

```
npm install -g @angular/cli
```

Esto nos permitirá utilizar todos los comandos de angular.

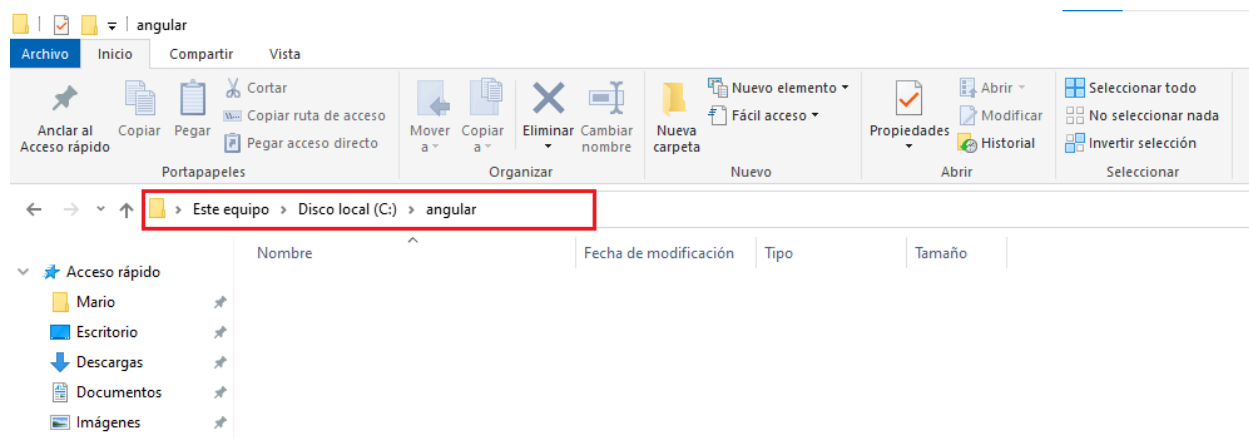
Nota: Como podrás intuir, es necesario tener instalado Node en nuestro equipo.

## Nuestro primer Proyecto

---

Para generar un proyecto de Angular los pasos son:

1. Crear una nueva carpeta para nuestro proyecto.
2. Hacer click en la barra de direcciones en el explorador de windows.



3. Copiar la dirección de la carpeta.
4. Abrir una nueva ventana de CMD.
5. Dirigirse a la carpeta con el comando CD

a. Ej: `cd C:\angular`

6. Una vez se encuentre en la carpeta creada, ejecute el siguiente comando:

a. `ng new nombreDelProyecto`

Esto iniciará el proceso de creación de proyectos de angular.

```
CA% Símbolo del sistema
Microsoft Windows [Versión 10.0.19044.1766]
(c) Microsoft Corporation. Todos los derechos reservados.
C:\Users\Mario>cd C:\angular
C:\angular>ng new primerProyecto_
```

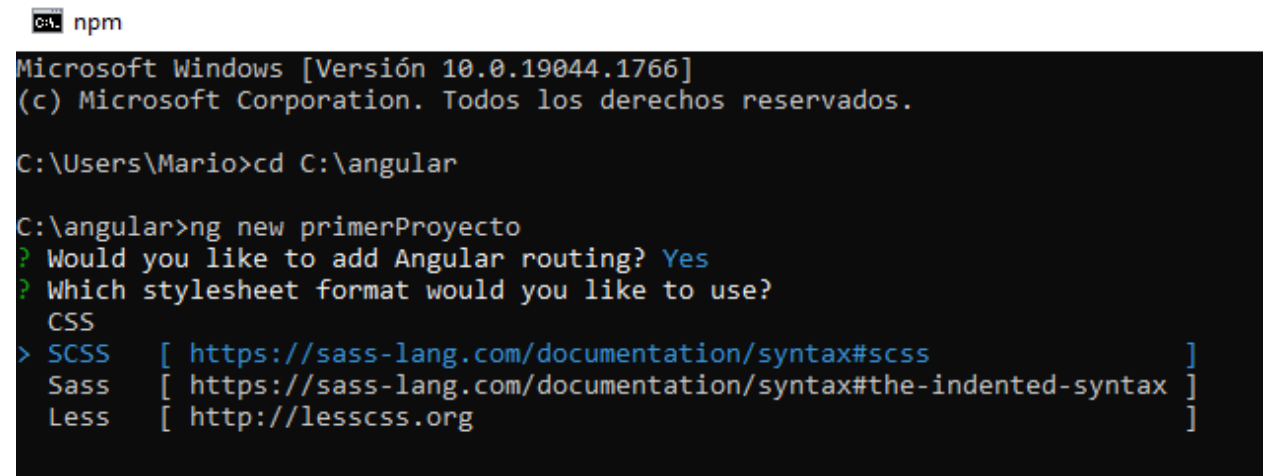
Nota: puede que nos pregunte si queremos enviar información anónima a google. Esto es a elección propia.

Nos preguntará si deseamos incluir Routing (se detallará que es más adelante). Ingresamos "Y" y presionamos Enter.

```
CA% npm
Microsoft Windows [Versión 10.0.19044.1766]
(c) Microsoft Corporation. Todos los derechos reservados.
C:\Users\Mario>cd C:\angular
C:\angular>ng new primerProyecto
? Would you like to add Angular routing? (y/N) y_
```

Nos preguntará qué procesador de CSS (hojas de estilo) deseamos usar.

Nos moveremos con las flechas hasta la opción 2 (SCSS)



```
cmd npm
Microsoft Windows [Versión 10.0.19044.1766]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Mario>cd C:\angular

C:\angular>ng new primerProyecto
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use?
  CSS
> SCSS [ https://sass-lang.com/documentation/syntax#scss ]
  Sass [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less [ http://lesscss.org ]
```

Luego comenzará la creación del proyecto con la configuración ingresada.

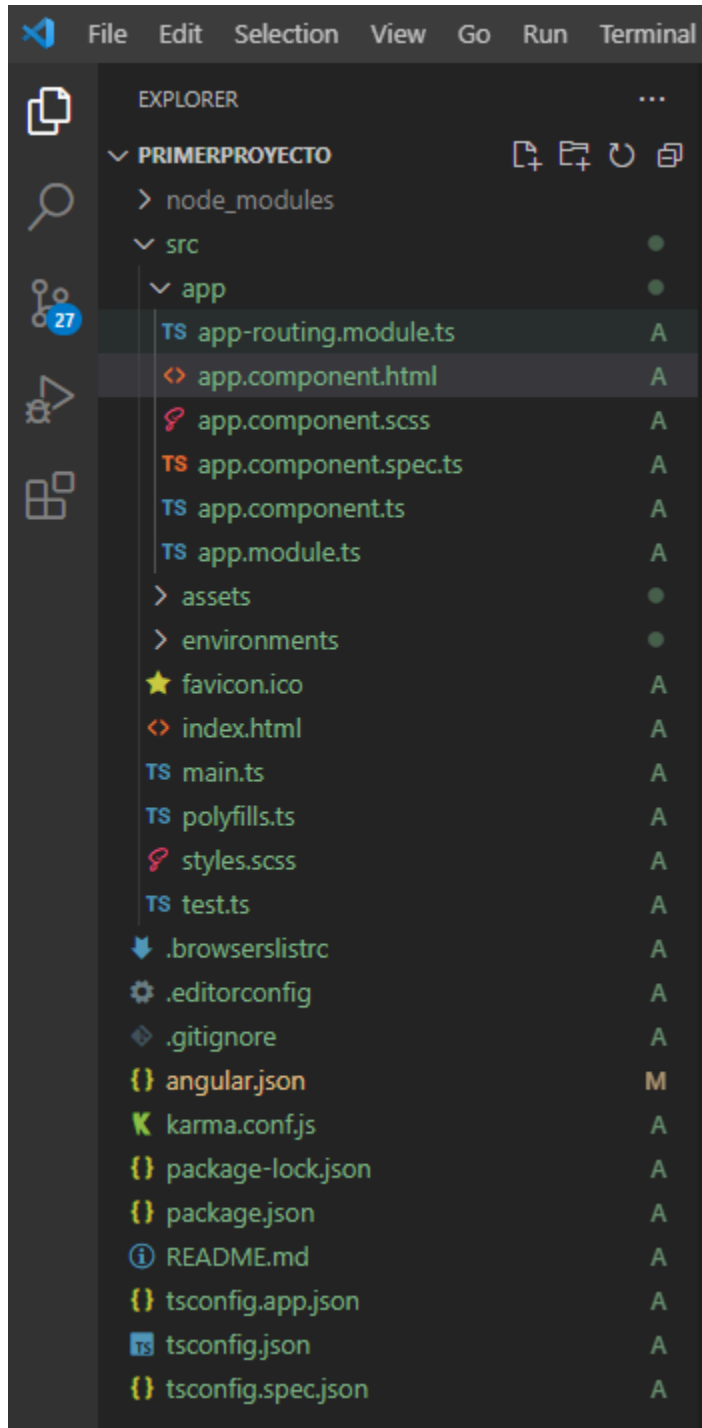
A continuación debemos abrir Visual Studio Code y seleccionar la carpeta que nos generó Angular CLI.

Nota: Se recomienda instalar el paquete de extensiones “Angular essentials by Jhon Papa” para trabajar de forma más eficiente con Angular dentro de VSCode.

Pueden descargarlo desde [aquí](#).

## Entendiendo Angular

Al abrir el proyecto nos encontraremos con una lista de archivos generados automáticamente, vamos a detallar a continuación los más importantes.



- Archivos con la extensión *.spec.ts*

Estos archivos son archivos de testing, son usados para crear tests unitarios.  
(Veremos qué es esto más adelante en este curso)

- *tsconfig.json*

Contiene la configuración de *typescript*.

- *package.json*

Contiene la lista de dependencias del proyecto.

- *package-lock.json*

Contiene la lista de dependencias con sus versiones.

- *angular.json*

Configuración de Angular.

- *.gitignore*

Archivos para ignorar de la lista de archivos que serán subidos a github

- Carpeta *src*: contiene todos los archivos pertinentes al proyecto en sí.

- Carpeta *Assets*: contiene los assets como imágenes, fuentes, iconos, sonido, etc.

- Carpeta *Environments*: contiene los archivos del entorno de trabajo.

- Carpeta *App*: contiene los componentes, módulos y demás archivos de angular del proyecto. Dentro de ella podemos encontrar los siguientes archivos.

*app-routing.module.ts*: Aquí se definen las URL que tendrá la página.

*app-component.html*: El HTML del componente principal del proyecto.

*app-component.scss*: la hoja de estilos del componente principal del proyecto.

*app-component.ts*: el archivo de lógica (parecido al JS) del componente principal del sistema.

*app-module.ts*: el archivo del módulo principal del proyecto.

## Index.html

---

Dejamos este archivo para el final pues es importante.

Como mencionamos anteriormente las [SPAs](#) son páginas web que poseen solo un HTML. Esto es cierto pero con matices.



El archivo Index.html es nuestro archivo principal, es decir la base de nuestro proyecto. Veremos a continuación su estructura.

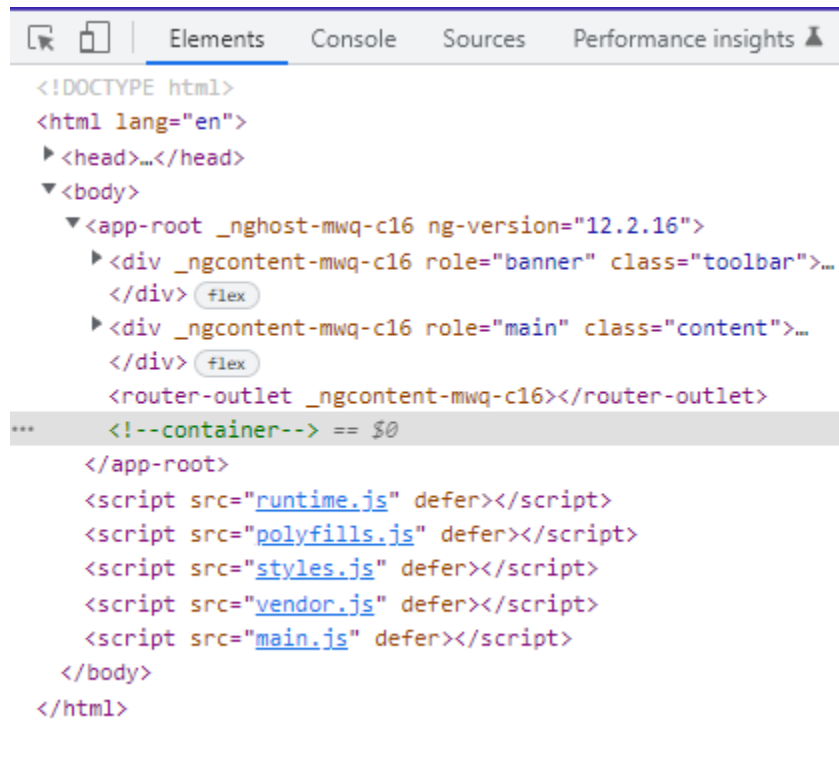
```
src > index.html > ...
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>PrimerProyecto</title>
6    <base href="/">
7    <meta name="viewport" content="width=device-width, initial-scale=1">
8    <link rel="icon" type="image/x-icon" href="favicon.ico">
9  </head>
10 <body>
11   <app-root></app-root>
12 </body>
13 </html>
14
```

Como podemos observar a simple vista, no dista mucho de cualquier archivo HTML que conozcamos, pero la magia sucede dentro la tag `<body>`.

Como podemos observar, no nos encontramos con una etiqueta HTML que podamos reconocer. Nos encontramos con una etiqueta `<app-root>`.

Esta etiqueta es una etiqueta personalizada, utilizada por angular para indicar dónde debe colocar el contenido del proyecto. Es decir que angular reemplazará dicha etiqueta con el contenido que nosotros desarrollemos.

Veamos la etiqueta en acción.

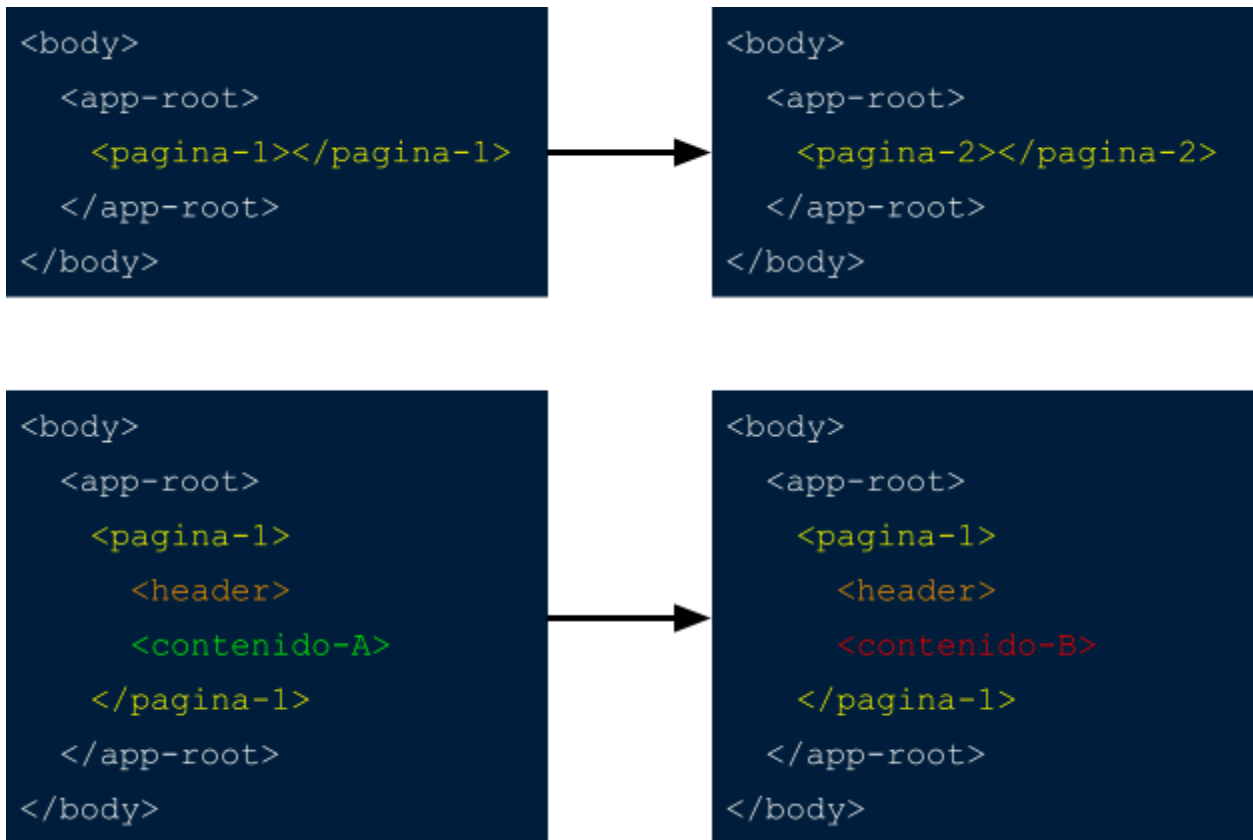


```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <app-root _ngghost-mwq-cl6 ng-version="12.2.16">
      <div _ngcontent-mwq-cl6 role="banner" class="toolbar">...
    </div>
      <div _ngcontent-mwq-cl6 role="main" class="content">...
    </div>
      <router-outlet _ngcontent-mwq-cl6></router-outlet>
    <!--container--> == $0
  </app-root>
  <script src="runtime.js" defer></script>
  <script src="polyfills.js" defer></script>
  <script src="styles.js" defer></script>
  <script src="vendor.js" defer></script>
  <script src="main.js" defer></script>
</body>
</html>
```

Si observamos el HTML al ejecutar el proyecto (no funciona el hacer doble click al HTML ;) nos encontramos con que dentro de la etiqueta `<app-root>` se ha generado contenido nuevo, este es el contenido que nosotros vamos a programar.

Bajo dicha etiqueta nos encontramos con varios archivos javascript. Estos fueron creados por angular y son los que se encargan de hacer dicho reemplazo además de contener toda la lógica que nosotros desarrollemos.

Para comprender mejor las SPAs, veamos el siguiente ejemplo



En la primera fila podemos observar cómo suceden los cambios de página dentro de una SPA.

El contenido interno de la tag `<app-root>` es la propia página 1, pero al cambiar de página, es decir, al navegar por esta web, lo que sucede es que el contenido de la tag `<app-root>` cambia para convertirse ahora en la página 2. Es por esto que al navegar en una SPA, no se refresca el navegador, pues no lo necesita. Basta con que Angular actualice el contenido por la página nueva.

Este comportamiento se puede replicar también dentro de la propia página:

Podemos observar en la segunda fila, que hay elementos que dejamos fijos como un header, mientras que el contenido A lo podemos reemplazar por el contenido

B. Esta es la fortaleza de las SPAs, el poder modificar y reemplazar el contenido mientras se ejecuta la aplicación sin la necesidad de refrescar el browser.

## Primeros pasos

---

Como podemos observar, dentro de la carpeta app es donde debemos comenzar.

Los archivos que nos interesan son:

```
app-component.html
```

```
app-component.scss
```

```
app-component.ts
```

Angular por defecto usa 3 archivos para sus componentes (dejaremos de lado el archivo .spec.ts, dado que forma parte de un tópico más avanzado).

Mencionamos anteriormente que una SPA contiene solo un archivo HTML pero esta afirmación tiene un pequeño matiz. Si bien es solo un archivo HTML principal (el Index.html) todo el resto del contenido de la página será guardado en archivos HTML los cuales usará angular para reemplazar el contenido del archivo html principal.

El primer archivo es el archivo html del componente. Allí haremos nuestra maquetación de la misma manera que lo haríamos en cualquier página web.

El segundo archivo es el archivo SCSS. Aquí colocaremos nuestros estilos CSS para usarlos desde el archivo HTML. Usamos este archivo con extensión SCSS pues funciona igual que un archivo .CSS pero con algunas funciones extra.

El tercer archivo es el archivo con extensión TS. Dicho archivo se encargará de contener la lógica de nuestra aplicación. Es como si adjuntamos un archivo .JS a la página.

En el archivo `app-component.html` encontraremos mucho código generado automáticamente por Angular. Esto es simplemente una pequeña página de bienvenida que debemos eliminar antes de comenzar.

Antes de eliminarla vamos a ver que nos muestra.

## Iniciando la aplicación

---

Para iniciar la aplicación angular debemos crear un servidor local para correr la aplicación. Angular nos provee una manera simple de crear uno.

Mientras estamos en VS Studio Code:

Terminal -> Nueva Terminal

Aquí ejecutaremos el siguiente comando.

**`ng serve --open`**

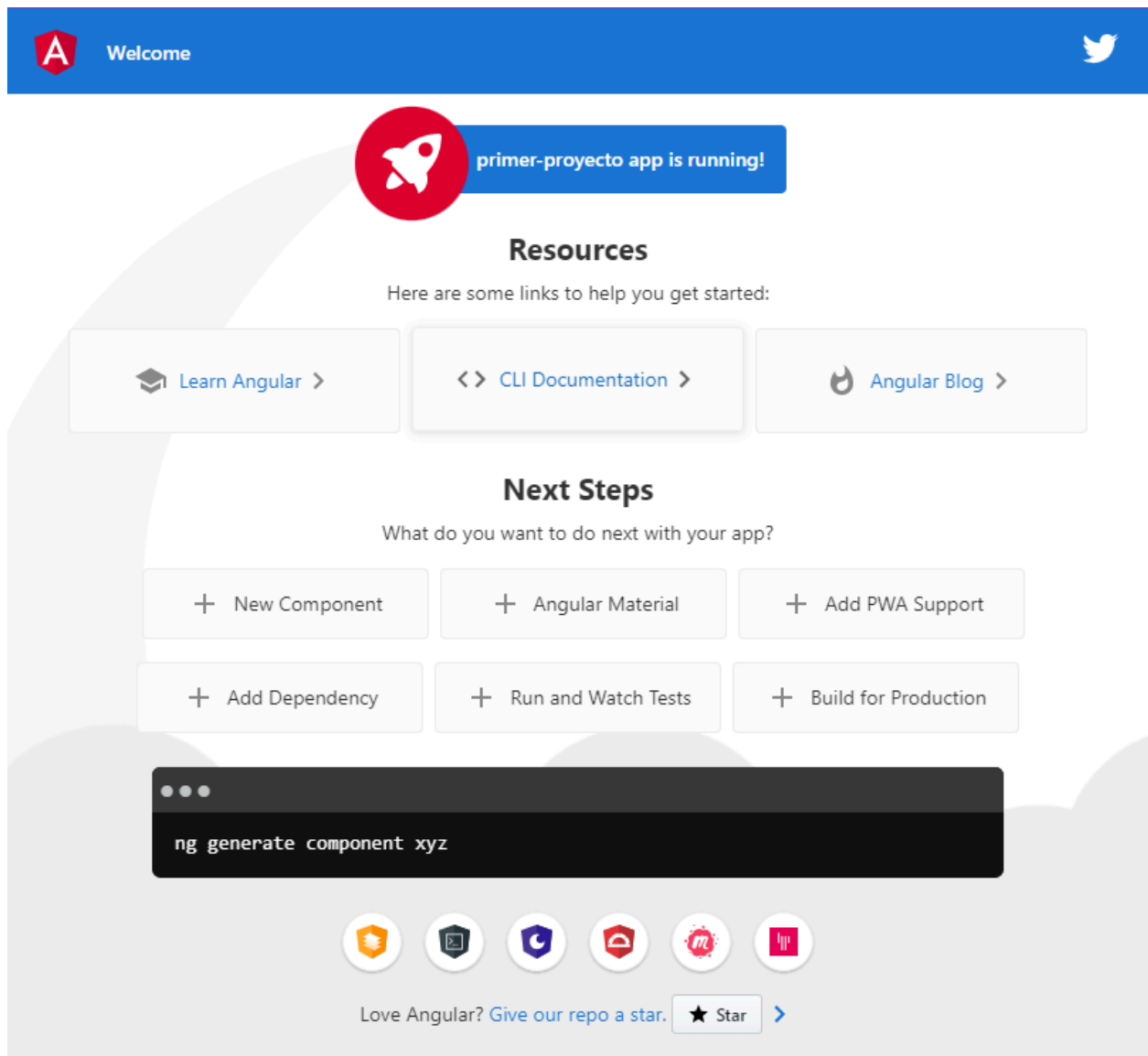
Ng serve se encargará de generar el servidor, revisar los archivos en busca de cambios y recargará la aplicación cuando detecte cambios.

La opción **`--open`** simplemente se encarga de abrir el proyecto en el navegador.

El proyecto normalmente podemos encontrarlo en la URL

<http://localhost:4200/>

Con esto, veremos como se abre en nuestro navegador una nueva ventana con la aplicación de angular que se generó de forma automática.



El código de esta página está en app-component.html. Podemos eliminar dicho código completo.

```
app.component.html X
src > app > app.component.html > ...
1 <!-- * * * * * -->
2 <!-- * * * * * The content below * * * * * -->
3 <!-- * * * * * is only a placeholder * * * * * -->
4 <!-- * * * * * and can be replaced. * * * * * -->
5 <!-- * * * * * -->
6 <!-- * * * * * Delete the template below * * * * * -->
7 <!-- * * * * * to get started with your project! * * * * * -->
8 <!-- * * * * * -->
9
10 <style>
11   :host {
12     font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, Helvet
13     font-size: 14px;
14     color: #333;
15     box-sizing: border-box;
16     -webkit-font-smoothing: antialiased;
17     -moz-osx-font-smoothing: grayscale;
18   }
19
20   h1,
21   h2,
22   h3,
23   h4,
24   h5,
25   h6 {
```

## Clicker!

Una vez eliminado el código vamos a armar un pequeño proyecto mientras explicamos algunos conceptos básicos de Angular.

Haremos un pequeño clicker, donde dispondremos de un botón y un contador de clicks.

Comenzamos con el archivo `app-component.html` donde colocaremos el siguiente código:

```
<button>+1</button>
<div>👍 0</div>
```

Si guardamos los cambios, el browser refrescará la página y veremos lo siguiente:



Simplemente un botón y el contador.

¿Pero cómo hacemos que al hacer click en el botón se sume un valor?

Anteriormente mencionamos que el archivo TS es el archivo donde programaremos toda lógica de la página web. Por lo tanto nuestro interés es ahora el archivo `app-component.ts`.

Dentro del mismo encontraremos código escrito, echemos un vistazo:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  title = 'primer-proyecto';
}
```

Un poco más adelante en este curso detallaremos qué es cada línea. Comencemos simplemente eliminando la línea:

```
title = 'primer-proyecto';
```



Nos quedará algo como el siguiente ejemplo. Aquí colocaremos nuestro código.

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  //el código va aquí
}
```

Crearemos una variable para mantener el conteo actual de clicks y una función que incrementa el valor en 1 cada vez que la llamamos.

```
export class AppComponent {
  contador = 0;

  sumarUno() {
    this.contador = this.contador + 1;
    // También puede escribirse así: this.contador++;
  }
}
```

Por si solos esta función y variable no hacen nada, debemos crear una conexión entre el HTML y el archivo TS.

Lo haremos de la siguiente manera:

```
<button (click)="sumarUno()">+1</button>
<div>👍 {{contador}}</div>
```

Como podemos observar, no fue necesario importar ningún archivo JS ni realizar ninguna conexión manual con el archivo TS, angular se encarga de eso por nosotros. Veamos entonces qué hace cada línea.

## Event Binding

---

En la primer línea, podemos observar una referencia a “click” y a la función “sumarUno”.

En Angular podemos escuchar un evento de HTML simplemente escribiendo su nombre entre paréntesis: `(click)`. Esto se denomina “Event binding” o en español “Vincular eventos”.

Al hacer esto, angular “escuchará” dicho evento y ejecutará las instrucciones que le indiquemos entre las comillas dobles: “”

Es así que nos queda la siguiente línea:

```
<button (click)="sumarUno()">+1</button>
```

Los eventos a los que nos podemos suscribir son los mismos eventos que existen en HTML solo que se le debe quitar el sufijo “on”. Podemos ver la lista de eventos en el siguiente [enlace](#).

Dentro de las comillas dobles, podemos escribir comandos simples para ejecutar cuando el evento suceda. En este caso se llama a la función `sumarUno()` pero podemos llamar a funciones, realizar asignaciones de variables y muchas más cosas dependiendo de nuestras necesidades.

¿Por qué no pruebas a cambiar el evento “click” por el evento “mouseover”? Solo será necesario pasar el mouse por encima del botón para incrementar el valor del contador.

## Interpolation

---

En la segunda línea, podemos observar una estructura algo extraña: `{{ }}`

Esto se denomina “Interpolación” y sirve para mostrar valores dentro del HTML.

Angular se encarga de escribir el valor de las expresiones o variables que coloquemos dentro de la estructura de llaves dobles.

```
<div>👍 {{contador}}</div>
```

En este caso, el valor numérico de la variable `contador` será mostrado. Si escribimos “contador” sin las llaves dobles, solo nos mostrará dicha palabra.

Como mencionamos anteriormente, también se pueden ingresar expresiones, por lo que el siguiente ejemplo es válido:

```
<div>👍 {{contador > 5}}</div>
```

En este caso, nos mostrará la palabra “false” hasta que el contador alcance un valor mayor a 5 y pasará a ser “true”.

## Estilos

---

Lo último que nos queda es aplicar estilos para mejorar el aspecto visual de nuestro clicker. Los estilos se aplican en el archivo `app-component.scss`

Pero antes aplicaremos un pequeño cambio en el HTML:

```
<div class="container">
  <button (click)="sumarUno()">+1</button>
  <div>👍 {{contador}}</div>
</div>
```

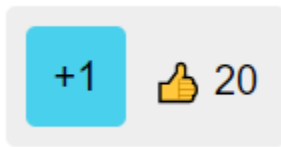
Con este cambio hecho, podemos aplicar estilos en el archivo .SCSS

```
.container {
  margin: 20px;
  padding: 10px 5px;
  display: flex; // hacer que el contenido se vea horizontal
  background-color: #eee;
  width: fit-content; // evitar que el gris ocupe todo el ancho
  border-radius: 5px; // redondeamos bordes
}

.container button { // cualquier elemento button dentro de container
  border-radius: 5px; // redondeamos bordes
  border: none; // quitamos los bordes del button
  background-color: #49d0ec;
  padding: 10px;
  margin: 0 5px;
  width: 50px;
  height: 50px;
  font-size: 20px;
  cursor: pointer; // cambia el cursor al pasar el mouse por el botón
}

.container div { // cualquier elemento div dentro de container
  min-width: 70px; // usamos min width para permitir que crezca
  height: 50px;
  display: flex; // cambiamos el display a flex para alinear
  justify-content: center; // alinear al centro horizontalmente
  align-items: center; // alinear al centro verticalmente
  font-size: 20px;
}
* {font-family: sans-serif}
```

Con esto tenemos un clicker con un poco más de personalidad:



## Atributos y Directivas

---

Un atributo es un parámetro de una *TAG HTML* que define ciertas características de la misma. Podemos usar atributos para que una *TAG HTML* tenga un comportamiento distinto al habitual, extender sus funcionalidades y/o cambiar su funcionamiento completamente.

Veamos un ejemplo con una tag *INPUT* y un *IMG*

```
<input type="text" placeholder="User">
<input type="password" placeholder="Password">


```

El atributo **type** define el comportamiento del input, si es **text** entonces funcionará como un input normal, pero si es **password** ocultará los caracteres que ingresemos. El atributo **placeholder** define la palabra a mostrar dentro del input si no se ha ingresado nada.

Para el caso del tag *IMG* no mostrará ninguna imagen si no completamos el atributo **src** pues en él ingresamos la URL de la imagen que queremos mostrar.

Las directivas tal como los atributos, permiten modificar el comportamiento de los elementos y se escriben de la misma manera que los atributos.

Vamos a ver un ejemplo de directivas.

## \*ngIf

---

La directiva **\*ngIf** añade lógica condicional a los elementos HTML. De esta manera, podemos mostrar un elemento o no basados en la expresión que ingresemos.

```
<div class="container">
  <button (click)="sumar(1)" *ngIf="!bonus">+1</button>
  <button (click)="sumar(10)" *ngIf="bonus">+10</button>

  <div>👍 {{contador}}</div>
</div>
```

```
export class AppComponent {

  contador = 0;
  bonus = false;

  sumar(cantidad: number): void {
    this.contador = this.contador + cantidad;

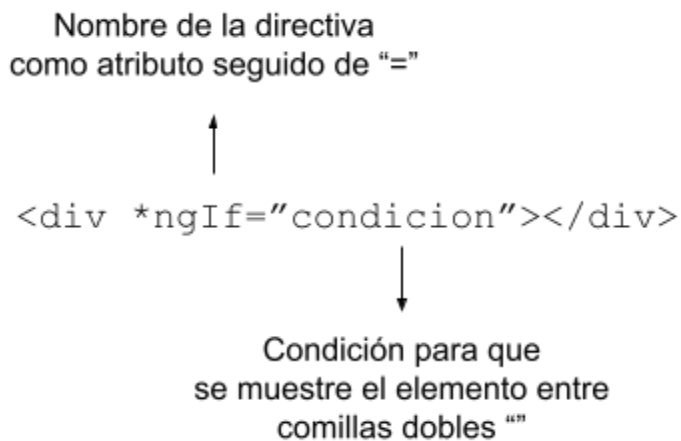
    // cuando alcanzamos 10, habilita un botón de bonus
    if (this.contador === 10) {
      this.bonus = true;

      // timeout de 5 segundos, pasado este tiempo el botón volverá a
      su estado original
      window.setTimeout(() => {
        this.bonus = false;
      }, 5000);
    }
  }
}
```

En este ejemplo, tenemos dos botones que poseen un **\*ngIf** el primero estará activo si la variable *bonus* es falsa, y el segundo si la variable es verdadera.

En el código al llegar a la cantidad de 10, la variable *bonus* pasará a ser verdadera, activando el botón de bonus. Finalmente usamos [setTimeout](#) para setear la variable *bonus* a falsa en 5 segundos.

La sintaxis de **\*ngIf** es bastante simple.



## **\*ngFor**

---

La directiva **\*ngFor** nos permite iterar en una array y “multiplicar” la tag en la que ingresemos dicha directiva. Veamos un ejemplo

```

<div class="container">
  <button (click)="sumar(1)" *ngIf="!bonus">+1</button>
  <button (click)="sumar(10)" *ngIf="bonus">+10</button>

  <div>👍 {{contador}}</div>
</div>

<p>Puntuaciones más altas</p>

<table>
  <tr>
    <th>Nombre</th>
    <th>Puntuación</th>
  </tr>
  <tr *ngFor="let jugador of tablaPuntuaciones">
    <td>{{jugador.nombre}}</td>
    <td>{{jugador.puntuacion}}</td>
  </tr>
</table>

```

```

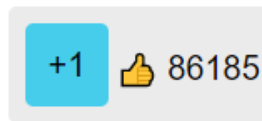
tablaPuntuaciones = [
  {nombre: 'Juan', puntuacion: 855},
  {nombre: 'Pedro', puntuacion: 803},
  {nombre: 'Luis', puntuacion: 720},
];

```

Tenemos un elemento *table* con una fila (<tr>) para los títulos (<td>).

Luego una nueva fila la cual posee la directiva **\*ngFor**. Esto implica que el elemento de filas (<tr>) será multiplicado dependiendo de la cantidad de elementos que posea el array *tablaPuntuaciones*. Veamos cómo se comporta nuestra aplicación:

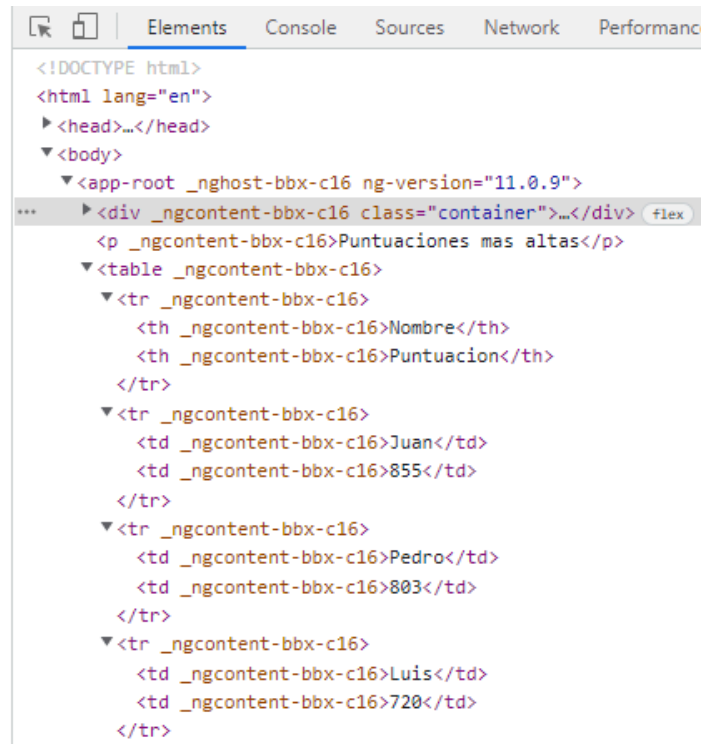




Puntuaciones mas altas

#### Nombre Puntuacion

Juan	855
Pedro	803
Luis	720



Podemos observar que el elemento `<tr>` se ha multiplicado 3 veces y ha colocado los datos del array correspondientes en su lugar.

Veamos su sintaxis:

```
<tr *ngFor="let jugador of tablaPuntuaciones">
```

Esta escritura es igual al ciclo for usando la palabra clave **of** en javascript:

```
let iterable = [10, 20, 30];

for (let value of iterable) {
  console.log(value);
}
// 10
// 20
// 30
```

En primer lugar podemos observar que debemos usar la palabra clave **let** seguido de una variable. Jugador es en realidad una variable que contiene el valor del array según la iteración en la que se encuentre. Por último tenemos la palabra clave **of** la cual precede al nombre de la variable que contiene el array que deseamos iterar.

Por último cabe mencionar que la variable jugador nos permite acceder a las propiedades del objeto, es por eso que podemos usar [Interpolación](#) para ver los valores de la variable que se está iterando. También es importante saber que la variable jugador se puede usar dentro de la etiqueta `<tr>` que contiene la directiva **\*ngFor**. No se puede usar fuera de dicha etiqueta.