	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN	
CICLO: 02	GUIA DE LABORATORIO #03	
	Nombre de la Práctica:	Introducción Angular
	MATERIA:	Diseño y Programación de Software Multiplataforma

I. OBJETIVOS

Que el estudiante:

- Diseñe aplicaciones web utilizando funciones de Angular.
- Diseñe aplicaciones con navegación.
- Aprender hacer interfaz para que el usuario pueda interactuar con la aplicación.
- Aprender a crear componente personalizado.
- Hacer uso de binding en el diseño de interfaz angular.
- Aprender hacer interfaz para que el usuario pueda interactuar con la aplicación.
- Aprender a utilizar pipes dentro de la interfaz de usuario.
- Hacer uso de los componentes en el diseño de interfaz angular.

II. INTRODUCCION TEORICA

Angular es un framework para el desarrollo de aplicaciones web. Está pensado para dividir un proyecto en componentes y ser reutilizadas en proyectos medianos y grandes.

Sus principales competidores son Vue (proyecto iniciado por Evan You en 2014) y React (proyecto iniciado por Facebook en 2013)

Angular tiene su salida al mercado en 2016 pero tiene una versión previa no compatible y solo mantenida para proyectos antiguos llamada Angular.js (2010)

El proyecto de Angular es propiedad de la empresa de Google.

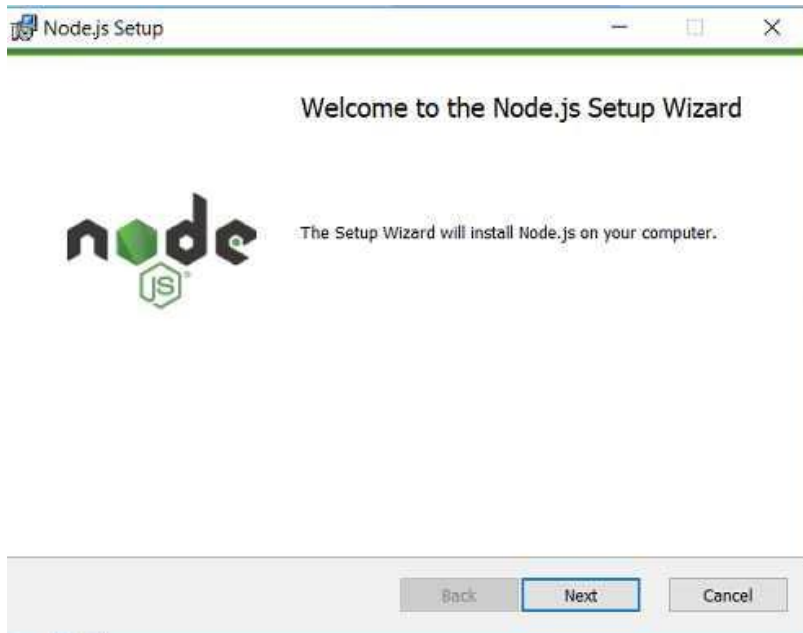
La versión de Angular que trabajamos es la 9 (salió el 6/2/2020)

Para desarrollar en forma efectiva una aplicación en Angular debemos instalar al menos dos herramientas básicas:

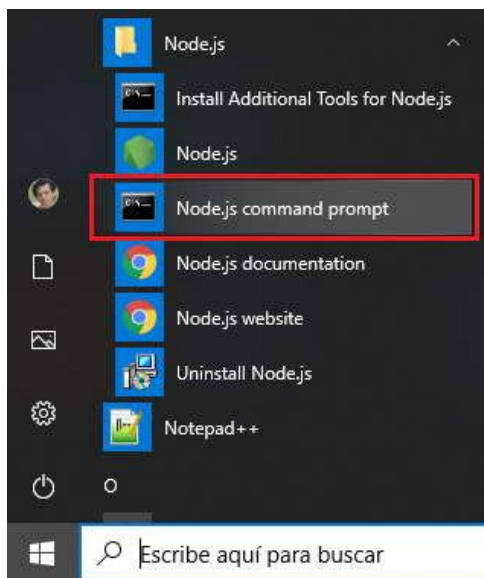
Node.js

Angular CLI (Command Line Interface - Interfaz de línea de comandos)

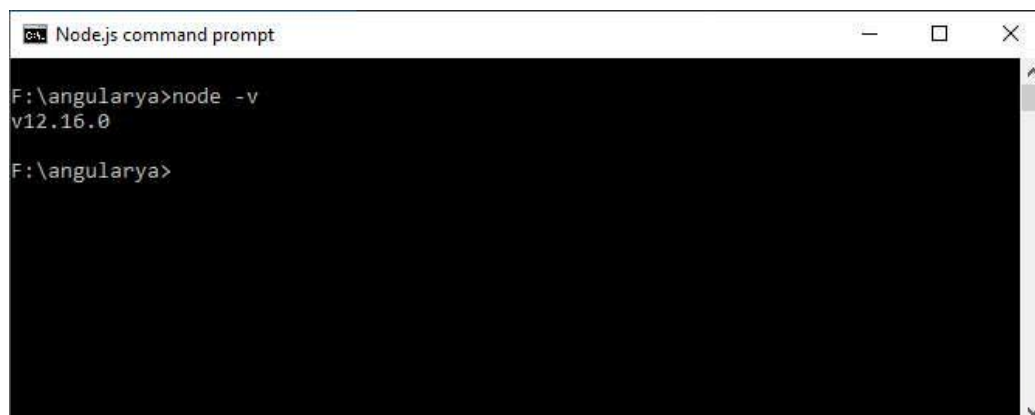
Debemos Descargar e instalar la última versión estable de Node.js:



Una vez instalado debemos ingresar a la línea de comandos que nos provee Node.js:



Para comprobar su correcto funcionamiento podemos averiguar su versión:



```

Node.js command prompt
F:\angularaya>node -v
v12.16.0
F:\angularaya>
  
```

INSTALACIÓN DE ANGULAR CLI

Para instalar este software lo hacemos desde la misma línea de comandos de Node.js (por eso lo instalamos primero), debemos ejecutar el siguiente comando

Ver las versiones disponibles Angular : `npm view @angular/cli`

instalar la versión deseada: `npm install -g @angular/cli@9.0.0`

Verificar versión : `ng --version`

Es importante el -g para que se instale en forma global.

Qué es Angular CLI

Dentro del ecosistema de Angular encontramos una herramienta fundamental llamada "Angular CLI" (Command Line Interface). Es un producto que en el momento de escribir este artículo todavía se encuentra en fase beta, pero que ya resulta fundamental para el trabajo con el framework.

los directorios y archivos generados al crear un nuevo proyecto son necesarios para que estas herramientas funcionen. Entre otras cosas tendremos:

- Un servidor para servir el proyecto por HTTP
- Un sistema de live-reload, para que cuando cambiamos archivos de la aplicación se refresque el navegador
- Herramientas para testing
- Herramientas para despliegue del proyecto
- Etc.

Un Component controla una zona de espacio de la pantalla que podríamos denominar vista. Un componente es una clase estándar de ES6 decorada con `@Component`.

El componente define propiedades y métodos que están disponibles en su template, pero eso no te da licencia para meter ahí todo lo que te parezca. Es importante seguir una aproximación de

diseño **S.O.L.I.D.**, y extraer toda la lógica en servicios para que el controlador solo se encargue de gestionar 1 única cosa: la vista.

El siguiente ejemplo, donde definimos un componente **TodoList**, que se encargará de controlar un listado de tareas para hacer (“to do” tasks).

```
//app/todos/todos-list/todo-list.component.ts
import { OnInit } from '@angular/core';
import { Todo } from '../shared/todo.model';
import { TodoService } from '../shared/todo.service';

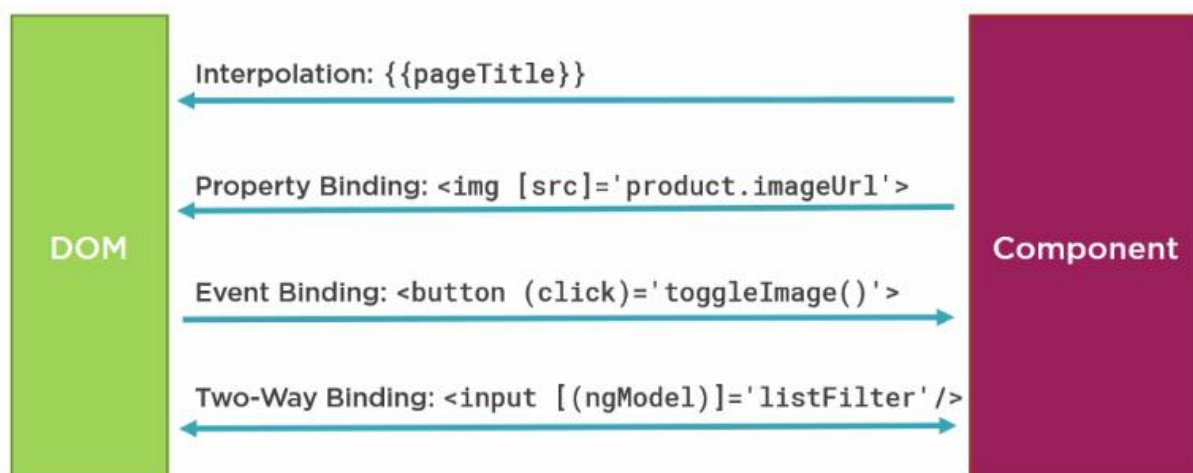
export class TodoListComponent implements OnInit {
  todos: Todo[];
  selectedTodo: Todo;
  constructor(private service: TodoService) {}
  ngOnInit() {
    this.todos = this.service.getTodos();
  }
  selectTodo(todo: Todo) { this.selectedTodo = todo; }
}
```

Como ves, el componente contiene un listado de tareas, una tarea seleccionada, el método para seleccionar la tarea y una llamada a **ngOnInit** para inicializar -gracias al servicio inyectado **TodoService**- el listado de tareas; es de mencionar que el método **ngOnInit** que usa nuestro componente porque implementa la interfaz **OnInit**, se llama cuando se crea el componente y forma parte de las llamadas del ciclo de vida de los componentes.

Los “Data Bindings” o enlace de datos es una técnica general que une una fuente de datos entre el origen de datos y la vista, y se encarga de sincronizarlos. Esto provoca que cada cambio de datos se refleje automáticamente en los elementos que están sincronizados.

Los databinding son la sincronización automática de los datos entre los componentes del modelo y la vista. La vista es una proyección del modelo en todo momento. Cuando el modelo cambia, la vista refleja el cambio, y viceversa. Por eso, debes conocer cómo tratar los datos y su enlace con las vistas.

Es decir nos interesa transmitir las propiedades de los Component al DOM (la vista) para por ejemplo cambiar el color del fondo de la página de blanco a negro. Y también necesitamos que los eventos de la vista sean comunicados al Component, por ejemplo, para implementar un botón con el cual mostrar / ocultar las imágenes de la lista de productos.



III. MATERIALES Y EQUIPO

Para la realización de la guía de práctica se requerirá lo siguiente:

No.	Requerimiento	Cantidad
1	Guía de práctica #6: Manejo de Hojas de Estilo con el DOM	1
2	Computadora con editor HTML y navegadores instalados	1
3	Memoria USB o disco flexible	1

IV. PROCEDIMIENTO

Creación de un proyecto y prueba de su funcionamiento

Para crear un proyecto vamos a utilizar la aplicación Angular CLI que acabamos de instalar en el concepto anterior.

Desde la línea de comandos de Node.js procedemos a ejecutar el siguiente comando, pero antes ubicarnos en la ruta donde deseamos crear el proyecto.

```
ng new proyecto001
```

Se nos pide si queremos crear rutas (tema que veremos más adelante), elegiremos 'N'.

Luego seleccionaremos que utilizaremos archivos CSS para los estilos (valor seleccionado por defecto):

Este comando crea la carpeta proyecto001 e instala una gran cantidad de herramientas que nos auxiliarán durante el desarrollo del proyecto. Como Angular está pensado para aplicaciones de complejidad media o alta no hay posibilidad de instalar menor herramientas.

El proceso de generar el proyecto lleva bastante tiempo ya que deben descargarse de internet muchas herramientas.

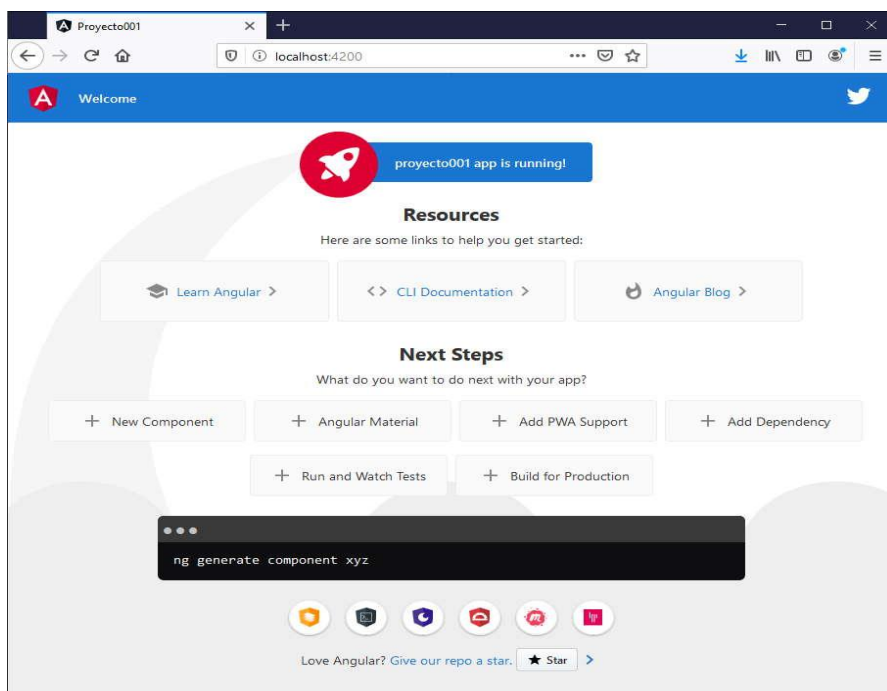
Se genera una aplicación con el esqueleto mínimo, para probarlo debemos descender a la carpeta que se acaba de crear y lanzar el siguiente comando desde Node:

```
ng serve -o
```



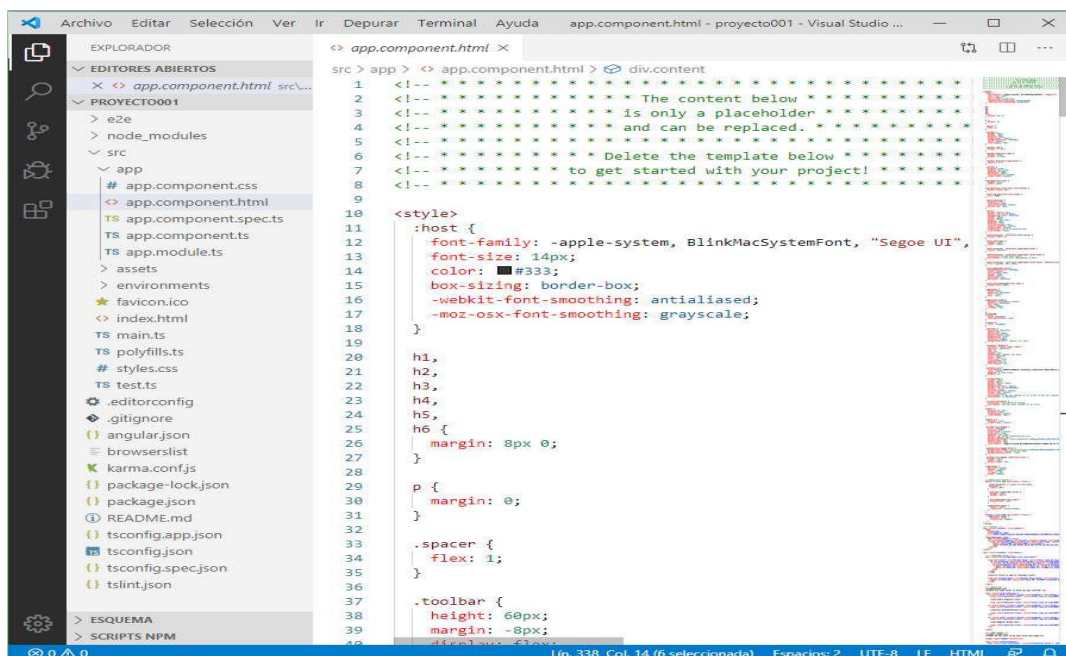
Este comando arranca un servidor web en forma local y abre el navegador para la ejecución de la aplicación.

En el navegador tenemos como resultado:



Debemos utilizar un editor de texto para codificar la aplicación. Yo recomiendo el Visual Studio Code, puede visitar luego un tutorial completo del editor VS Code

Si utilizamos este editor podemos elegir la opción: Archivo -> Abrir carpeta y proceder a buscar la carpeta 'proyecto001':



En este concepto no me interesa ver todos las carpetas y archivos generados. Solo efectuaremos un cambio para ver como se reflejan en el navegador.

En la carpeta 'proyecto001' hay una subcarpeta llamada 'src' y dentro de esta una llamada 'app', busquemos el archivo 'app.component.html' y procedamos a borrar las más de 500 líneas.

Si leemos las primeras líneas nos informa que siempre debemos modificar el archivo con los algoritmos de nuestro proyecto (se genera a modo de ejemplo):

Disponemos el siguiente código remplazando al generado en forma automática:

```
<h1 style="text-align:center">
```

```
Bienvenido a {{ title }}
```

```
</h1>
```

Una vez que grabamos los cambios en este archivo podemos ver que automáticamente se ven reflejados en el navegador (recordemos de no cerrar la ventana de Node.js que tiene el servidor web en funcionamiento en forma local):



A partir del próximo concepto comenzaremos a analizar un proyecto Angular, en este momento me interesa solo recordar los pasos que debemos dar para crear y ejecutar un proyecto.

Archivos y carpetas básicas de un proyecto en Angular

Para crear un proyecto en Angular utilizamos la herramienta Angular CLI y desde la línea de comandos escribimos:

ng new proyecto001

No haremos por el momento un estudio exhaustivo de todos los archivos y carpetas que se crean (más 55000 archivos y 4700 carpetas en la versión de Angular 9.x), sino de aquellas que se requieren modificar según el concepto que estemos estudiando.

En Angular la pieza fundamental es la 'COMPONENTE'. Debemos pensar siempre que una aplicación se construye a base de un conjunto de componentes (por ejemplo pueden ser componentes: un menú, lista de usuarios, login, tabla de datos, calendario, formulario de búsqueda etc.)

Angular CLI nos crea una única componente llamada 'AppComponent' que se distribuye en 4 archivos:

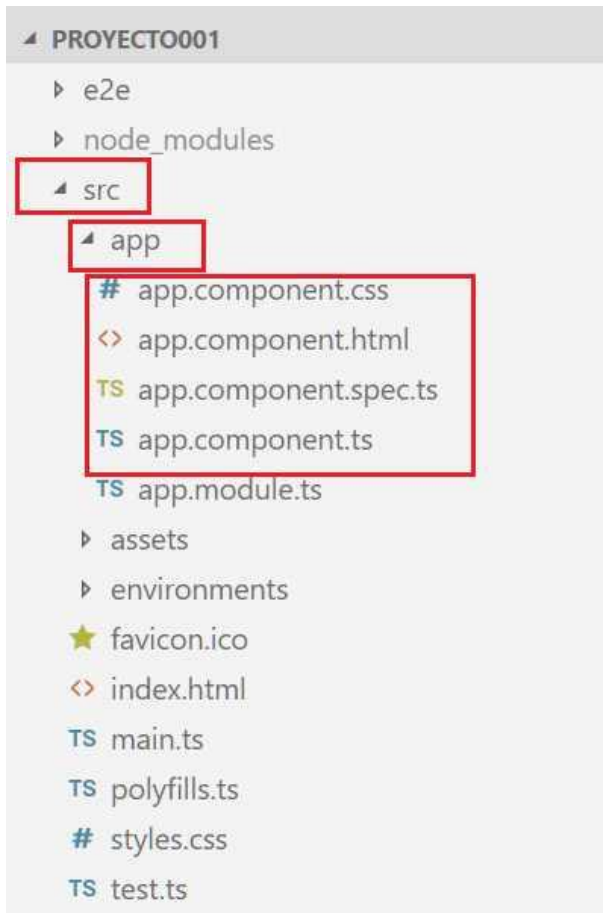
app.component.ts

app.component.html

app.component.css

app.component.spec.ts

Todos estos archivos se localizan en la carpeta 'app' y esta carpeta se encuentra aplicación de la carpeta 'src':



En Angular se programa utilizando el lenguaje **TypeScript** que vamos a ir aprendiéndolo a lo largo del curso. El archivo donde se declara la clase AppComponent es 'app.component.ts':

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'proyecto001';
}
```

La Clase **AppComponent** define un atributo llamado 'title' y lo inicializa con el string 'proyecto001' que coincide con el nombre del proyecto que creamos:

```
title = 'proyecto001';
```

Dijimos anteriormente que la clase completa se distribuye en otros archivos y podemos ver que mediante la función decoradora `@Component` le indicamos los otros archivos que pertenecen a esta componente:

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

El archivo 'app.component.html' tiene la parte visual de nuestra componente 'AppComponent' y está constituido mayormente por código HTML (cada vez que realicemos un proyecto a este código lo borraremos para resolver nuestro problema):

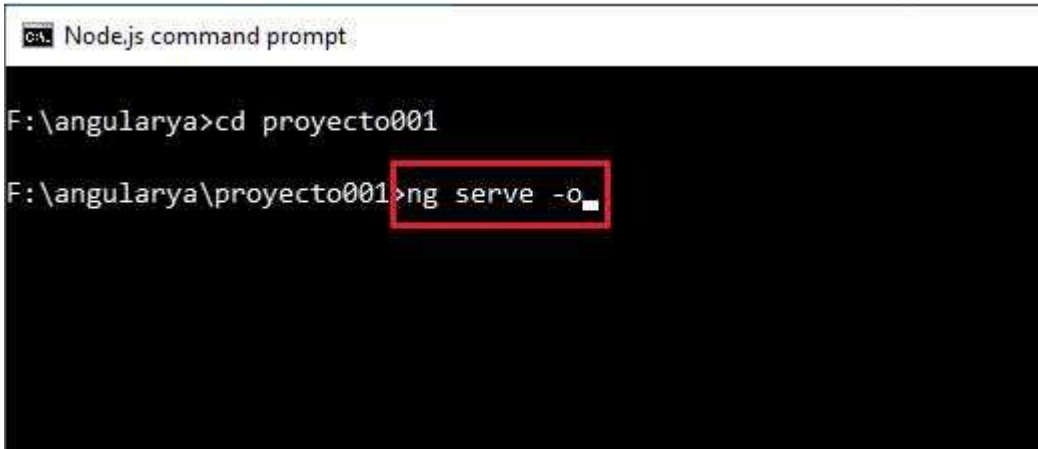
Como vimos en el concepto anterior cambiamos el contenido de este archivo por:

```
<h1 style="text-align:center">
  Bienvenido a {{ title }}
</h1>
```

Analizaremos ahora de este trozo de HTML donde aparece el atributo 'title' de la componente:

```
Bienvenido a {{ title }}
```

Cuando ejecutamos nuestra aplicación desde la línea de comandos de Node.js:



The image shows a terminal window titled "Node.js command prompt". The prompt is at "F:\angularya>". The user has entered "cd proyecto001" and the prompt has moved to "F:\angularya\proyecto001>". The user has then entered "ng serve -o", which is highlighted with a red rectangle. The command is partially executed, showing a cursor at the end.

En el navegador aparece el contenido de la propiedad 'title':



Podemos ver que aparece el string 'proyecto001' y no `{{ title }}`:

```
title = 'proyecto001';
```

Este concepto de sustitución se llama interpolación y lo veremos en forma más profunda en el concepto siguiente.

Otro archivo que se asocia a la componente 'AppComponent' es 'app.component.css' donde se almacenan todos los estilos que se van a aplicar solo a dicha componente, es decir que quedarán encapsulados en la componente 'AppComponent'.

En la carpeta raíz del proyecto hay un archivo llamado 'styles.css' donde podemos definir estilos que se aplicarán en forma global a todas las componentes de nuestra aplicación:



Ya hemos nombrado los tres archivos fundamentales que definen toda componente:

app.component.ts

app.component.html

app.component.css

Queda uno llamado 'app.component.spec.ts' que tiene por objetivo definir código de testing para medir el correcto funcionamiento de la componente (dejaremos para más adelante este concepto)

Otro archivo fundamental que nos crea Angular CLI es 'app.module.ts' en la misma carpeta donde se encuentran los 4 archivos de la componente 'AppComponent':

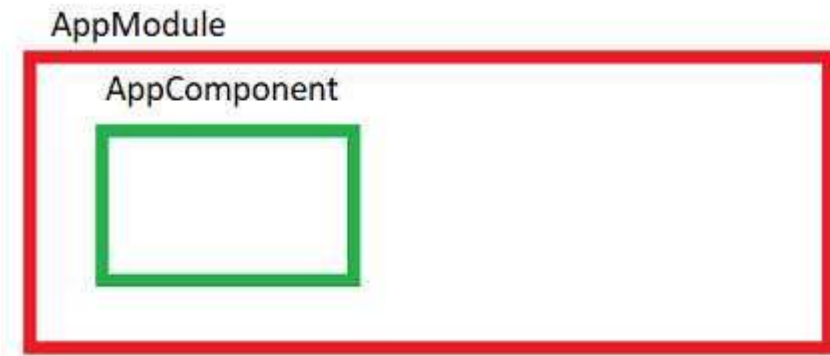


app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

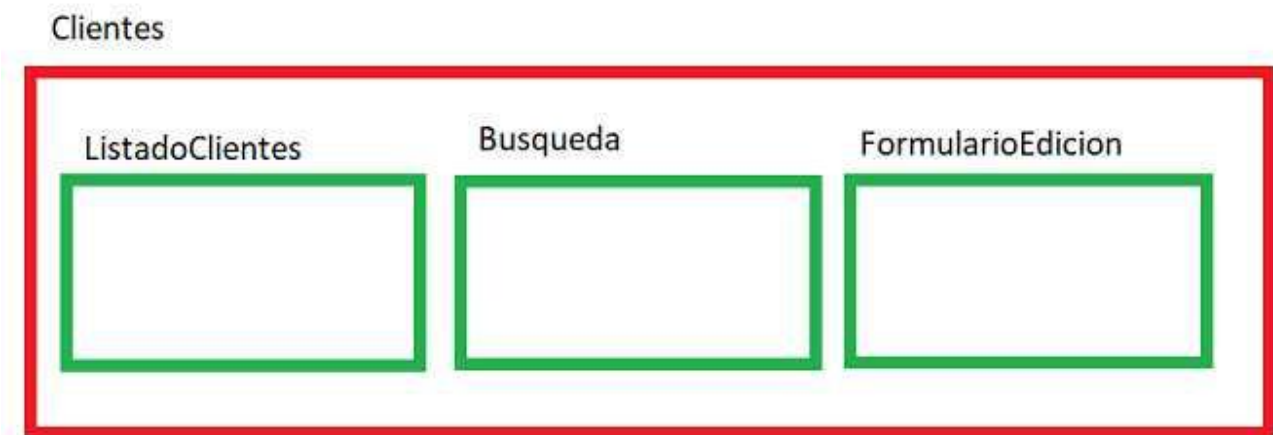
Nombramos este archivo porque como mínimo una aplicación en Angular debe tener un módulo. Podemos ver que en la función `@NgModule` en la propiedad `'declarations'` se le pasa un vector con un elemento que es nuestra componente `'AppComponent'`.

La aplicación mínima en Angular debe tener un módulo y dentro de dicho módulo como mínimo una componente:



Un proyecto grande se divide en diferentes módulos con un conjunto de componentes cada uno.

Por ejemplo podemos tener un módulo `'Clientes'` con tres componentes que resuelven distintas partes visuales de la aplicación:



En los primeros conceptos estaremos trabajando con la única componente `'AppComponent'` y más adelante veremos cómo crear más componentes en el módulo `'AppModule'` e inclusive como crear más módulos.

Hay muchos más archivos y carpetas en el proyecto que nos crea Angular CLI pero iremos viendo su objetivo a medida que avancemos.

Interpolación en los archivos HTML de Angular

Una de las características fundamentales en Angular es separar la vista del modelo de datos. En el modelo de datos tenemos las variables y en la vista implementamos como se muestran dichos datos.

Modificaremos el proyecto001 para ver este concepto de interpolación.

Abriremos el archivo que tiene la clase AppComponent (app.component.ts) y lo modificaremos con el siguiente código:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  nombre = 'Rodriguez Pablo';
  edad = 40;
  email = 'rpablo@gmail.com';
  sueldos = [1700, 1600, 1900];
  activo = true;

  esActivo() {
    if (this.activo)
      return 'Trabajador Activo';
    else
      return 'Trabajador Inactivo';
  }

  ultimos3Sueldos() {
    let suma=0;
    for(let x=0; x<this.sueldos.length; x++)
      suma+=this.sueldos[x];
  }
}
```

```
    return suma;
  }
}
```

La clase 'AppComponent' representa los datos de un empleado. Definimos e inicializamos 5 propiedades:

```
nombre = 'Rodriguez Pablo';
edad = 40;
email = 'rpablo@gmail.com';
sueldos = [1700, 1600, 1900];
activo = true;
```

Definimos dos métodos, en el primero según el valor que almacena la propiedad 'activo' retornamos un string que informa si es un empleado activo o inactivo:

```
esActivo() {
  if (this.activo)
    return 'Trabajador Activo';
  else
    return 'Trabajador Inactivo';
}
```

El segundo método retorna la suma de sus últimos 3 meses de trabajo que se almacenan en la propiedad 'sueldos':

```
ultimos3Sueldos() {
  let suma=0;
  for(let x=0; x<this.sueldos.length; x++)
    suma+=this.sueldos[x];
  return suma;
}
```


Veamos ahora el archivo html que muestra los datos, esto se encuentra en 'app.component.html':

```
<div>
  <p>Nombre del Empleado:{{nombre}}</p>
  <p>Edad:{{edad}}</p>
  <p>Los últimos tres sueldos son: {{sueldos[0]}}, {{sueldos[1]}} y {{sueldos[2]}}</p>
  <p>En los últimos 3 meses ha ganado: {{ultimos3Sueldos()}}</p>
  <p>{{esActivo()}}</p>
</div>
```

Para acceder a las propiedades del objeto dentro del template del HTML debemos disponer dos llaves abiertas y cerradas y dentro el nombre de la propiedad:

```
<p>Nombre del Empleado:{{nombre}}</p>
```

Cuando se tratan de vectores la primer forma que podemos acceder es mediante un subíndice:

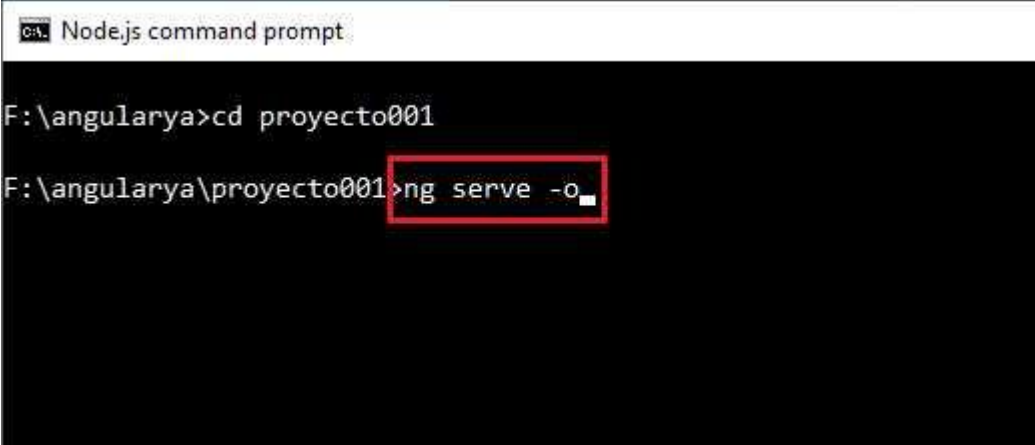
```
<p>Los últimos tres sueldos son: {{sueldos[0]}}, {{sueldos[1]}} y {{sueldos[2]}}</p>
```

Finalmente podemos llamar a métodos que tiene por objetivo consultar el valor de propiedades:

```
<p>En los últimos 3 meses ha ganado: {{ultimos3Sueldos()}}</p>
```

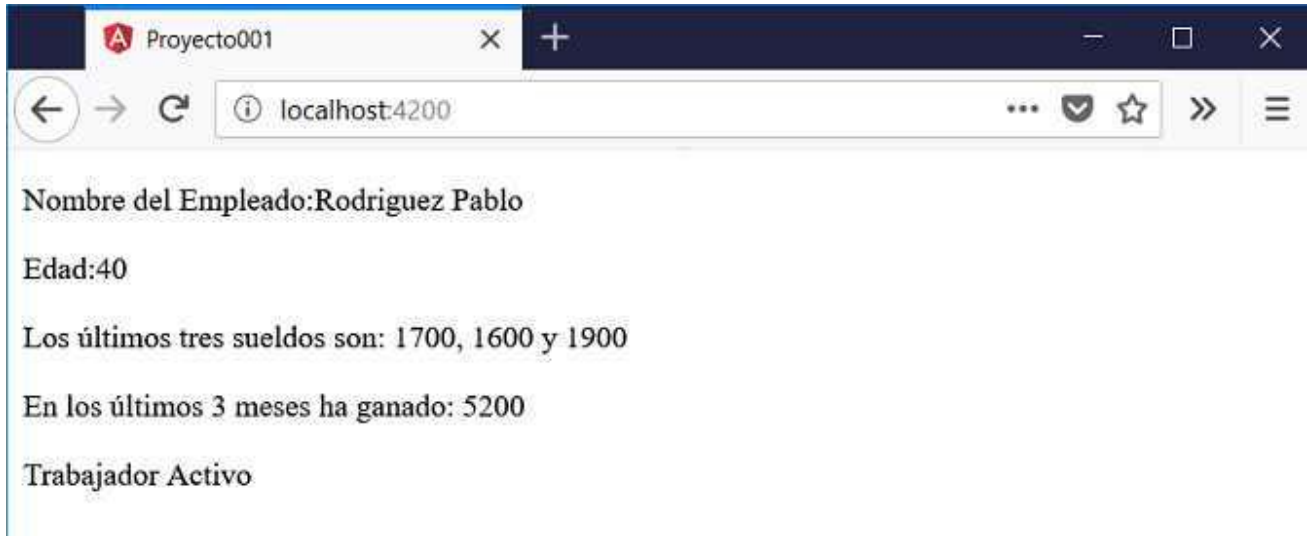
```
<p>{{esActivo()}}</p>
```

Cuando ejecutamos nuestra aplicación desde la línea de comandos de Node.js:



```
Node.js command prompt
F:\angularya>cd proyecto001
F:\angularya\proyecto001>ng serve -o
```

En el navegador aparece el contenido de la vista pero con los valores sustituidos donde dispusimos las llaves {{}}:



En principio podríamos decir que si los datos son siempre los mismos no tiene sentido definir propiedades en la clase y sustituirlos luego en el HTML, pero luego veremos que las propiedades las vamos a cargar mediante una petición a un servidor web, en esas circunstancias veremos la potencia que tiene modificar las propiedades y luego en forma inmediata se modifica la vista.

Acotaciones

Dentro de las dos llaves abiertas y cerradas Angular nos permite efectuar una operación:

`<p>En los últimos 3 meses ha ganado: {{sueldos[0]+sueldos[1]+sueldos[2]}}</p>`

Primero se opera la expresión dispuesta dentro de las llaves previo a mostrarla.

Otro ejemplo:

`<p>El empleado dentro de 5 años tendrá: {{edad+5}}</p>`

Podemos utilizar la interpolación como valor en propiedades de elementos HTML. Si en la clase tenemos definida la propiedad:

`sitio='http://www.google.com';`

Luego en la vista podemos interpolar la propiedad 'url' del elemento 'a' con la siguiente sintaxis:

`<p>Puede visitar el sitio ingresando aquí</p>`

Directivas *ngIf y *ngFor

Las directivas *ngIf y *ngFor son atributos que podemos agregarle a los elementos HTML que nos permiten en el caso del *ngIf condicionar si dicha marca debe agregarse a la página HTML.

La directiva *ngFor nos permite generar muchos elementos HTML repetidos a partir del recorrido de un arreglo de datos.

Para analizar con un ejemplo estas directivas procederemos nuevamente a modificar el proyecto001.

En el archivo 'app.component.ts' procedemos a codificar la clase AppComponent con la definición de 3 propiedades:

```
import { Component } from '@angular/core';
```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  nombre = 'Rodriguez Pablo';
  edad = 40;
  sueldos = [1700, 1600, 1900];
}
```

Hemos definido las propiedades nombre, edad y sueldos en la clase AppComponent:

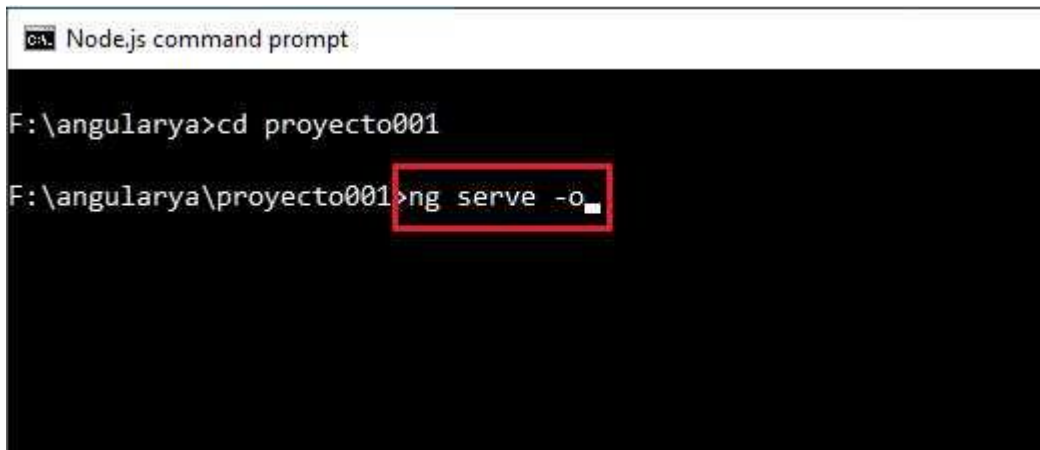
```
export class AppComponent {
  nombre = 'Rodriguez Pablo';
  edad = 40;
  sueldos = [1700, 1600, 1900];
}
```

Ahora procedemos a modificar el archivo app.component.html:

```
<div>
  <p>Nombre del Empleado:{{nombre}}</p>
  <p>Edad:{{edad}}</p>
```

```
<p *ngIf="edad>=18">Es mayor de edad.</p>
<table border="1">
  <tr>
    <td>Sueldos</td>
  </tr>
  <tr *ngFor="let sueldo of sueldos">
    <td>{{sueldo}}</td>
  </tr>
</table>
</div>
```

Ejecutemos nuestra aplicación desde la línea de comandos de Node.js:



The screenshot shows a Node.js command prompt window with the following text:

```
Node.js command prompt
F:\angularya>cd proyecto001
F:\angularya\proyecto001>ng serve -o
```

The command `ng serve -o` is highlighted with a red rectangle.

En el navegador aparece el siguiente contenido:



La directiva `*ngIf` verifica la condición que indicamos entre comillas, en el caso de verificarse verdadero se agrega el elemento HTML 'p':

```
<p *ngIf="edad>=18">Es mayor de edad.</p>
```

Probemos de modificar la propiedad `edad` en la clase `AppComponent` por el valor 7:

```
export class AppComponent {
  nombre = 'Rodriguez Pablo';
  edad = 7;
  sueldos = [1700, 1600, 1900];
}
```

Al recargar la página podemos comprobar que no aparece el mensaje contenido en dicho párrafo: 'Es mayor de edad.'

La directiva `*ngFor` nos genera posiblemente muchos elementos HTML repetidos, en este ejemplo una serie de filas de una tabla HTML:

```
<tr *ngFor="let sueldo of sueldos">
  <td>{{sueldo}}</td>
</tr>
```

En cada repetición en la variable 'sueldo' se almacena una componente del arreglo 'sueldos'. De esta forma podemos mostrar los datos del arreglo mediante la directiva `*ngFor`.

Acotaciones

La directiva *ngIf podemos plantear un else con la siguiente sintaxis:

```
<p *ngIf="edad>=18; else menor">Es mayor de edad.</p>
<ng-template #menor><p>Es un menor de edad.</p></ng-template>
```

En el caso que la condición del *ngIf se verifique falso le indicamos con un else un nombre que debe luego especificarse en un elemento ng-template. Lo que disponemos dentro del elemento ng-template es lo que se muestra.

Captura de eventos

Otra actividad muy común en una aplicación es la captura de eventos. La presión de un botón, la presión de una tecla, el desplazamiento de la flecha del mouse etc. son eventos que podemos capturar.

El evento más común que podemos encontrar en cualquier aplicación es la presión de un botón. Modificaremos nuevamente el proyecto001 para que la componente AppComponent muestre un etiqueta con un número 0 y luego dos botones que permitan incrementar o decrementar en uno el contenido de la etiqueta.

Nuevamente debemos modificar el archivo 'app.component.ts' con el siguiente código:

```
import { Component } from '@angular/core';
```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  contador = 1;

  incrementar() {
    this.contador++;
  }
}
```

```
decrementar() {
  this.contador--;
}
}
```

Definimos en la clase la propiedad 'contador' y lo iniciamos con el valor '1':

```
export class AppComponent {
  contador = 1;
```

Luego otros dos métodos de la clase AppComponent, que serán llamados al presionar alguno de los botones, incrementan en uno o decrementan en uno el valor almacenado en la propiedad contador:

```
  incrementar() {
    this.contador++;
  }
  decrementar() {
    this.contador--;
  }
}
```

Recordar que las propiedades dentro de los métodos debemos anteceder la palabra clave 'this'

El segundo archivo donde se encuentra la vista de la componente es app.component.html:

```
<div>
  <p>{{contador}}</p>
  <button (click)="incrementar()">Sumar 1</button>
  <button (click)="decrementar()">Restar 1</button>
</div>
```

Como ya conocemos mostramos el contenido de la propiedad contador mediante interpolación de string:

```
<p>{{contador}}</p>
```

Luego definimos dos elementos HTML de tipo 'button' y definimos los eventos click (deben ir entre paréntesis los nombres de los eventos) y luego entre comillas el nombre del método que se llama:

```
<button (click)="incrementar()">Sumar 1</button>
<button (click)="decrementar()">Restar 1</button>
```

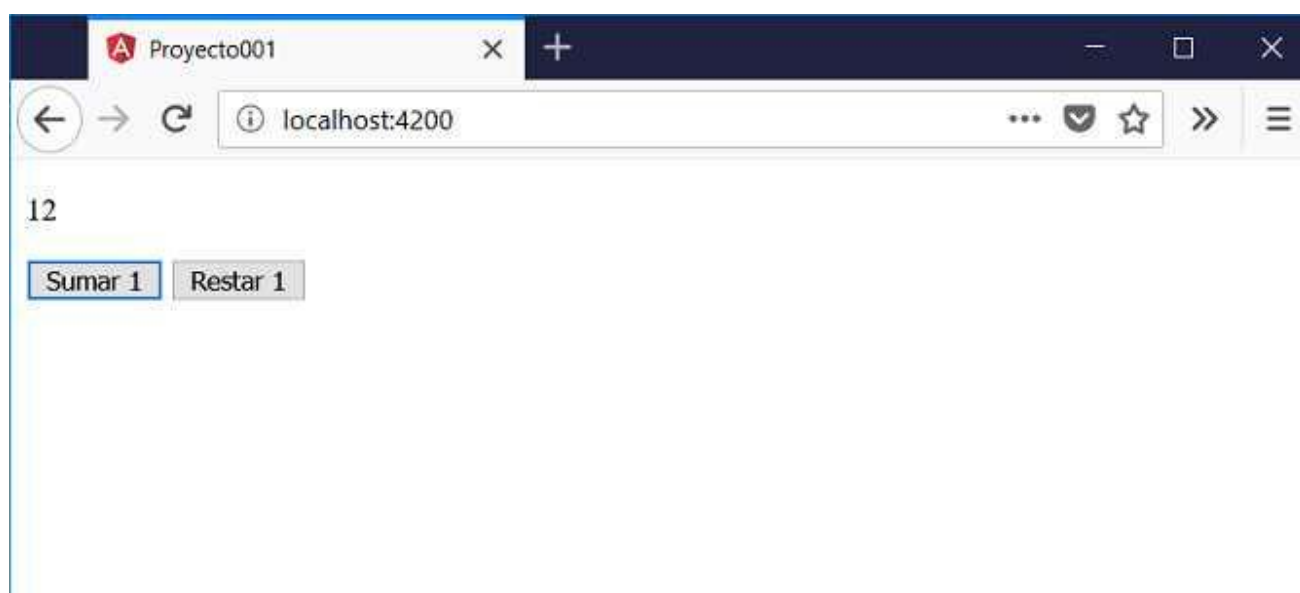
Ejecutemos nuestra aplicación desde la línea de comandos de Node.js:



```

Node.js command prompt
F:\angularya>cd proyecto001
F:\angularya\proyecto001>ng serve -o
  
```

En el navegador aparece la siguiente interfaz:



Cuando se presiona el botón 'Sumar 1' se llama el método 'incrementar()', en dicho método si recordamos se modifica el contenido de la propiedad 'contador':

```

incrementar() {
  this.contador++;
}
  
```

Lo más importante notar que Angular detecta cuando se modifican valores almacenados en propiedades y automáticamente se encarga de actualizar la interfaz visual sin tener que llamar a algún método.

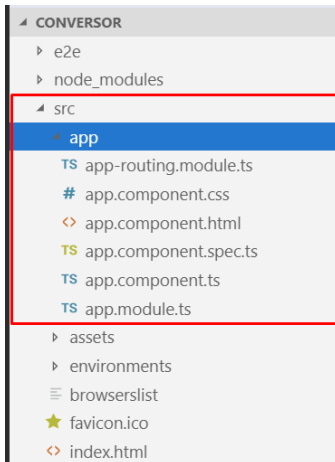
Este concepto se conoce como 'binding' en una dirección (cambio en atributos de la clase se actualizan en la vista)

Ejercicio 1: Conversor de longitud

1. Cree un proyecto angular llamado "Conversor".

```
C:\Users\k2m\Desktop>ng new Conversor
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
```

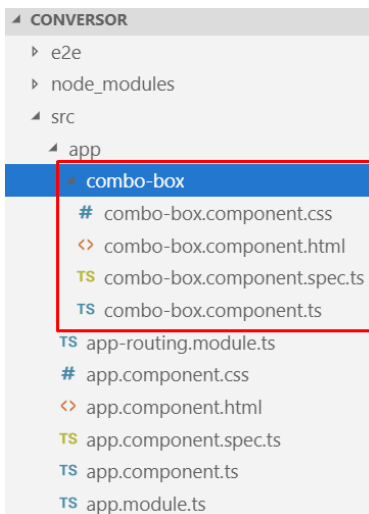
2. Abra la carpeta del proyecto "Conversor" dentro de visual code.
3. Verifique que presente la siguiente estructura:



4. Desde la terminal procederemos a crear un nuevo componente con el siguiente comando:

```
C:\Users\k2m\Desktop\Conversor>ng g c combo-box
```

5. Deberá presentar la siguiente estructura:



6. Ahora abrimos el documento "combo-box.component.html" y lo modificaremos con el siguiente código:

```
<div class="header">
  <h1>Conversion de Longitud</h1>
  <p>Ingrese los datos solicitados para realizar la conversion</p>
</div>
<label>Valor:</label>
<input type="text" [(ngModel)]="valorcm">
<label>Longitud:</label>
<select [(ngModel)]="opcionSeleccionado" (change)="capturar()">
  <option value="Selecciona">Selecciona</option>
  <option *ngFor="let u of unidades" value="{{u}}" >
    {{u}}
  </option>
</select>
<label [hidden]="opcionSeleccionado == 'Selecciona'">Su valor en {{ opcionSeleccionado }} es:
{{valorconversion}}</label>
```

- Ahora debemos moveremos al archivo combo-box.component.ts y debe quedar de la siguiente manera:

```
import { Component, OnInit } from '@angular/core';
//Agregar las siguientes modulos
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'

@Component({
  selector: 'app-combo-box',
  templateUrl: './combo-box.component.html',
  styleUrls: ['./combo-box.component.css']
})
export class ComboBoxComponent implements OnInit {
  //crear las siguientes variables
  unidades;
  opcionSeleccionado: string ;
  valorcm:number;
  valorconversion:number;

  constructor() {
```

```
}

ngOnInit() {
  //inicializar las variables
  this.unidades = ["Pulgada", "Metro", "Kilometro"];
  this.opcionSeleccionado = "Selecciona";
  this.valorcm = 0;
  this.valorconversion = 0;
}

//funcion que realiza los calculos
capturar() {
  switch(this.opcionSeleccionado){
    case 'Pulgada':
      this.valorconversion = this.valorcm * 0.39370;
      break;
    case 'Metro':
      this.valorconversion = this.valorcm / 100;
      break;
    case 'Kilometro':
      this.valorconversion = this.valorcm / 100000;
      break;
  }
}

export class AppComponent {
}
```

8. Crearemos nuestras reglas CSS para nuestra agregando el siguiente código en el archivo `combo-box.component.css`

```
.button {
  background-color: #4CAF50; /* Green */
  border: none;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
}

input,select {
  width: 100%;
  padding: 12px 20px;
  margin: 8px 0;
  box-sizing: border-box;
  font-size: 2vw;
}

label{
  font-size: 2vw;
}

.header {
  padding: 40px;
  text-align: center;
  background: #1abc9c;
  color: white;
  font-size: 30px;
}
```

9. Para poder trabajar con la captura de datos desde el formulario debemos realizar los siguientes cambios en el archivo en `app.module.ts`
 - Primero agregar el modulo de formulario de angular

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { ComboBoxComponent } from './combo-box/combo-box.component';
//agregar modulo de formulario
import {FormsModule} from '@angular/forms';
```

```
@NgModule({
```

- Finalmente importar el modulo al proyecto

```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AppRoutingModule } from './app-routing.module';
5 import { AppComponent } from './app.component';
6 import { ComboBoxComponent } from './combo-box/combo-box.compone
7 //agregar modulo de formulario
8 import {FormsModule} from '@angular/forms';
9
10 @NgM (alias) class AppComponent
11   de import AppComponent
12     AppComponent,
13     ComboBoxComponent
14   ],
15   imports: [
16     BrowserModule,
17     AppRoutingModule, FormsModule
18   ]
```

10. Ahora solo debemos agregar nuestro componente a la aplicación, y eso lo haremos en el archivo app.component.html, donde agregaremos el siguiente código:

<app-combo-box></app-combo-box>

11. Proceda a levantar los servicios de angular y deberá presentarse la siguiente pantalla:

Conversion de Longitud

Ingrese los datos solicitados para realizar la conversion

Valor:

Longitud:

Pulgada

Su valor en Pulgada es: 0.3937

Ejercicio 2: Validación de formularios

1. Cree un nuevo proyecto angular llamado “Alumnos”
2. Procederemos a crear nuevo componente llamado “Registro”.
3. Ahora abrimos el documento “Registro.component.html” y lo modificaremos con el siguiente código:

```
<div class="header">
  <h1>Datos de Alumno</h1>
  <p>Ingrese los datos de los alumnos</p>
</div>
<label>Nombre:</label>
<input type="text" [(ngModel)]="nombre">
<label>Edad:</label>
<input type="text" [(ngModel)]="edad">
<button class="button" (click)="ingresar()">Ingresar</button>
<br/><br/>
<table [hidden]="contador == 0">
  <thead>
    <tr>
      <th>Nombre</th>
      <th>Edad</th>
      <th>Mayor de Edad</th>
    </tr>
  </thead>
```

```
<tbody>
<tr *ngFor="let talumno of registro">
  <td>{{talumno.nombre}}</td>
  <td>{{talumno.edad}}</td>
  <td>{{talumno.mayor}}</td>
</tr>
</tbody>
</table>
```

4. Ahora debemos moveremos al archivo Registro.component.ts y debe quedar de la siguiente manera:

```
import { Component, OnInit } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { BrowserModule } from '@angular/platform-browser'
@Component({
  selector: 'app-registro',
  templateUrl: './registro.component.html',
  styleUrls: ['./registro.component.css']
})
export class RegistroComponent implements OnInit {
  registro=[];
  alumno:any;
  nombre:string;
  mayor:string;
  edad:number;
  contador:number;
  constructor() { }

  ngOnInit() {
    this.edad=0;
    this.nombre="";
    this.contador=0;
  }

  ingresar(){
    if(this.edad>0 && this.edad<18){
      this.mayor='No'
    }else{
```

```

    this.mayor='Si'
  }
  this.alumno={"nombre":this.nombre,"edad":this.edad,"mayor":this.mayor};
  this.registro.push(this.alumno);
  this.contador++;
}
}

```

5. Crearemos nuestras reglas CSS para nuestra agregando el siguiente código en el archivo `combo-box.component.css`

```

.button {
  background-color: #4CAF50; /* Green */
  border: none;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
}

input,select {
  width: 100%;
  padding: 12px 20px;
  margin: 8px 0;
  box-sizing: border-box;
  font-size: 2vw;
}

label{
  font-size: 2vw;
}

.header {
  padding: 40px;
  text-align: center;
  background: #1abc9c;
  color: white;
  font-size: 30px;
}

```


}

6. Para poder trabajar con la captura de datos desde el formulario debemos realizar los siguientes cambios en el archivo en app.module.ts

- Primero agregar el modulo de formulario de angular

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { RegistroComponent } from './registro/registro.component';
import { FormsModule } from '@angular/forms';

@NgModule({
  declarations: [
    AppComponent,
    RegistroComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule, FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

- Finalmente importar el modulo al proyecto

```
9  declarations: [
10     AppComponent,
11     RegistroComponent
12 ],
13 imports: [
14     BrowserModule,
15     AppRoutingModule, FormsModule
16 ],
17 providers: [],
18 bootstrap: [AppComponent]
19 })
20 export class AppModule { }
21
```

7. Ahora solo debemos agregar nuestro componente a la aplicación, y eso lo haremos en el archivo app.component.html, donde agregaremos el siguiente código:

<app-registro></app-registro>

8. Proceda a levantar los servicios de angular y deberá presentarse la siguiente pantalla:

Datos de Alumno

Ingrese los datos de los alumnos

Nombre:

Edad:

Nombre	Edad	Mayor de Edad
Karens Medrano	50	Si
Rafael Torres	90	Si
Pedro Picapiedra	15	No

Ejercicio 3: Validación de formularios

12. Cree un proyecto angular llamado "Formulario".
13. Desde la terminal procederemos a crear un nuevo componente de nombre Cliente:
ng generate component Cliente
14. Desde terminal procedemos a crear una nueva clase con nombre Cliente:
ng generate class Cliente
15. Dentro del proyecto instale bootstrap 4 con el siguiente comando:
npm install bootstrap jquery --save
16. Debemos configurar Bootstrap y JQuery en el archivo **angular.json**:

```
"styles": [
  "src/styles.css",
  "node_modules/bootstrap/dist/css/bootstrap.min.css"
],
"scripts": [
  "node_modules/jquery/dist/jquery.min.js",
  "node_modules/bootstrap/dist/js/bootstrap.min.js"
]
```

17. Agregamos el selector app-cliente en el archivo app.component.html:

```
<div class="container">
  <div class="row">
    <div class="col-sm-4">
      <app-clientes></app-clientes>
    </div>
  </div>
</div>
```

18. Debemos importar FormsModule en el archivo AppModule.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { ClientesComponent } from './clientes/clientes.component';
import { FormsModule } from '@angular/forms';
```

```
@NgModule({
  declarations: [
    AppComponent,
    ClientesComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
```

19. Modificar el archivo Cliente.ts:

```
export class Cliente {
  constructor(
    public id: number,
    public nombre: string,
    public nacionalidad: string,
    public edad?: number
  ) {}
}
```

20. Modificar el archivo Cliente.component.ts:

```
import { Component, OnInit } from '@angular/core';
import { Cliente } from '../cliente'

@Component({
  selector: 'app-clientes',
  templateUrl: './clientes.component.html',
  styleUrls: ['./clientes.component.css']
})
export class ClientesComponent implements OnInit {
  nacionalidad = ['', 'El Salvador', 'Mexico', 'Rusia', 'Mongolia'];
  cliente = new Cliente(1, '', '', 23);
  enviar=false;
  constructor() { }
  ngOnInit() {
  }
  onSubmit(){
    this.enviar=true;
  }
}
```

21. Modificar el template del componente cliente:

```
<div>
  <h2>Formulario Cliente</h2>
  <form>

    <div class="form-group">
      <label for="nombre">Nombre</label>
      <input type="text" class="form-control" id="nombre">
    </div>

    <div class="form-group">
      <label for="edad">Edad</label>
      <input type="text" class="form-control" id="edad">
    </div>

    <div class="form-group">
      <label for="nacionalidades">Nacionalidad</label>
      <select class="form-control" id="nacionalidades" required>
        <option *ngFor="let nacion of nacionalidad" [value]="nacion">{{nacion}}</option>
      </select>
    </div>
  </div>
```

```
<button type="submit" class="btn btn-success">Submit</button>
</form>
</div>
```

22. Para usar ngModel en directivas de two-way realizamos el siguiente cambio:

```
<div>
  <h2>Formulario Cliente</h2>
  <form #clienteform="ngForm">

    <div class="form-group">
      <label for="nombre">Nombre</label>
      <input type="text" class="form-control" id="nombre" required [(ngModel)]="cliente.nombre"
name="nombre" >
    </div>

    <div class="form-group">
      <label for="edad">Edad</label>
      <input type="text" class="form-control" id="edad" [(ngModel)]="cliente.edad" name="edad">
    </div>

    <div class="form-group">
      <label for="nacionalidades">Nacionalidad</label>
      <select class="form-control" id="nacionalidades"
        required [(ngModel)]="cliente.nacionalidades" name="nacionalidades" >
        <option *ngFor="let nacion of nacionalidad" [value]="nacion">{{nacion}}</option>
      </select>
    </div>

    <button type="submit" class="btn btn-success"
[disabled]="!clienteform.form.valid">Submit</button>
  </form>
</div>
<div>{{cliente | json}}</div>
```

23. Realice las pruebas respectivas en la aplicación.

24. Nuestra aplicación ya funciona correctamente para ello modificaremos primero el archivo cliente.component.css con el siguiente código:

```
.ng-valid[required], .ng-valid.required {
  border-left: 5px solid rgba(32, 77, 32, 0.623);
}

.ng-invalid:not(form) {
  border-left: 5px solid rgb(148, 27, 27);
}
```

25. Modificaremos el archivo cliente.component.html con el siguiente código:

```
<div [hidden]="enviar" >
  <h2>Formulario Cliente</h2>
  <form (ngSubmit)="onSubmit()" #clienteform="ngForm">

    <div class="form-group">
      <label for="nombre">Nombre</label>
      <input type="text" class="form-control" id="nombre" required [(ngModel)]="cliente.nombre"
name="nombre" #name="ngModel">
      <div [hidden]="name.valid || name.pristine">
        Nombre es requerido
      </div>
    </div>

    <div class="form-group">
      <label for="edad">Edad</label>
      <input type="text" class="form-control" id="edad" [(ngModel)]="cliente.edad" name="edad">
    </div>

    <div class="form-group">
      <label for="nacionalidades">Nacionalidad</label>
      <select class="form-control" id="nacionalidades"
        required [(ngModel)]="cliente.nacionalidades" name="nacionalidades"
#nacionalidades="ngModel">
        <option *ngFor="let nacion of nacionalidad" [value]="nacion">{{nacion}}</option>
```

```

</select>
<div [hidden]="nacionalidades.valid || nacionalidades.pristine" class="alert alert-danger">
  Nacionalidad es requerida
</div>
</div>

<button type="submit" class="btn btn-success"
[disabled]="!clienteform.form.valid">Submit</button>
</form>
</div>

<div [hidden]="!enviar">
  <h2>Enviar Cliente</h2>
  <hr>
  <div><span class="badge badge-info">Id</span> -> {{cliente.id}}</div>
  <div><span class="badge badge-info">Nombre</span> -> {{cliente.nombre}}</div>
  <div><span class="badge badge-info">Edad</span> -> {{cliente.edad}}</div>
  <div><span class="badge badge-info">Nacionalidad</span> -> {{cliente.nacionalidades}}</div>
  <br>
  <button class="btn btn-primary" (click)="enviar=false">Edit</button>
</div>

```

26. Realizar las respectivas pruebas a la aplicación la cual debe presentar la siguiente vista:

Ejercicio 4: Realizar aplicación web con angular CLI

1. Verificar que tenga instalado node versión 6 o superior, con el siguiente comando
node -v
2. Verificar que tenga instalado npm versión 3 o superior con el siguiente comando:
npm -v

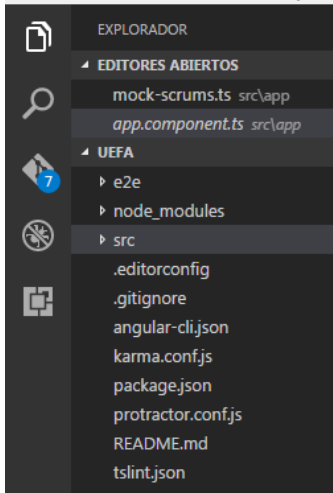
NOTA: si no está instalado proceda a instalar el node 6 desde la página oficial.

<https://nodejs.org/en/>

3. Deberemos instalar angular-cli, lo haremos a través de la consola, escribiendo el siguiente comando:
npm install -g angular-cli
1. En consola debemos ubicarnos en el directorio en el cual queremos crear nuestro proyecto angular.
2. Para crear nuestro proyecto lo haremos escribiendo en consola:

ng new uefa

4. Una vez creado el proyecto inicial, entrar en la carpeta del proyecto con el siguiente comando:
cd uefa
5. Ahora abriremos el nuestro proyecto con visual studio code, digitando la siguiente instrucción en la consola:
Code .
6. Una vez abierto visual studio code, encontrara un listado de archivos similares a este.



7. Definiremos nuestra primera clase, dentro de la carpeta src y dentro de la carpeta app, crear el archivo **scrum.ts**, y escribir el siguiente código:

```
export class Scrum{
  id:number;
  name:string;
  players:string[];
}
```

8. Siempre dentro de la carpeta app, crearemos el archivo **mock-scrums.ts**, donde tendremos un arreglo de equipos de la Liga de Campeones de la UEFA. Escriba el siguiente código:

```
import {Scrum} from './scrum';
```

```
export const SCRUMS: Scrum[]=[
  {id:1, name:'Real Madrid',players:['James Rodríguez','Gareth Bale','Cristiano Ronaldo']},
  {id:2, name:'Barcelona',players:['Lionel Messi','Neymar','Luis Suarez']},
  {id:3, name:'Mónaco',players:['Marcos Lopes','Hélder Costa']},
  {id:4, name:'Sevilla',players:['Paulo Henrique','Samir Nasri']},
  {id:5, name:'Arsenal',players:['Hiroshi']},
  {id:6, name:'Juventus',players:['Gonzalo Higuaín','Paulo Dybala']},
```



```
{id:7, name:'Bayer',players:['Javier Hernández','Hakan']},
{id:8, name:'Dortmund',players:['Marco Reus','Pierre Emerick','Mario Götze']},
{id:9, name:'Leicester City',players:['Jamie Vardy','Riyad Mahrez']},
{id:10,name:'Napoli',players:['Arkadiusz Milik','Marek Hamsík']}
];
```

9. Dentro de la carpeta app, crearemos el servicio de angular y obtendremos los datos atreves del archivo **scrum.service.ts**, crearlo y escribir el siguiente código dentro de el:

```
import{Injectable} from '@angular/core';
import {Scrum} from './scrum';
import{SCRUMS} from './mock-scrums';

@Injectable()
export class ScrumService{
  getScrums():Promise<Scrum[]>{
    return Promise.resolve(SCRUMS);
  }

  getScrumsSlowly(): Promise<Scrum[]> {
    return new Promise(resolve => {
      // Simulate server latency with 2 second delay
      setTimeout(() => resolve(this.getScrums()), 2000);
    });
  }
}
```

10. Ahora que ya está todo listo para montar la lista de equipos, empezaremos a crear la vista de detalle de cada equipo, creando el archivo **scrum-detail.component.ts** digite el siguiente código:

```
import { Component, Input } from '@angular/core';
import { Scrum } from './scrum';

@Component({
  selector: 'my-scrum-detail',
  template: `
    <div *ngIf="scrum">
      <h2>{{scrum.name}} Detalles:</h2>
    </div>
```

```

    <label>id:</label>{{scrum.id}}
  </div>
</div>
<label>nombre:</label>
<input [(ngModel)]="scrum.name" placeholder="name">
</div>
<div>
<label>Jugadores:</label>
<ul>
  <li *ngFor="let item of scrum.players">
    <span >{{item}}</span>
  </li>
</ul>
</div>
</div>`
})
export class ScrumDetailComponent {
  @Input()
  scrum:Scrum;
}

```

11. Para generar la lista dinámicamente a partir de nuestro servicio debemos consumir nuestro archivo mock para ello abriremos el archivo **app.component.ts**, digite el siguiente código:

```

import { Component, OnInit } from '@angular/core';
import { Scrum } from './scrum';
import { ScrumService } from './scrum.service';

```

```

@Component({
  selector: 'app-root',
  template: `
<div class="container">
  <div class="jumbotron">
    <div class="media">
      <div class="media-left">
        <a href="#">
          

```

```

    </a>
  </div>
  <div class="media-body">
    <h1>{{title}}</h1>
  </div>
</div>
</div>
<div class="row">
  <div class="col-sm-4">
    <h2>Equipos</h2>
    <ul class="scrums">
      <li *ngFor="let scrum of scrums"
        [class.selected]="scrum === selectedScrum"
        (click)="onSelect(scrum)">
        <span class="badge">{{scrum.id}}</span>{{scrum.name}}
      </li>
    </ul>
  </div>
  <div class="col-sm-8">
    <my-scrum-detail [scrum]="selectedScrum"></my-scrum-detail>
  </div>
</div>
</div>
  `;
  styleUrls: ['./app.component.css'],
  providers:[ScrumService]
})

export class AppComponent implements OnInit{
  title = 'Liga de Campeones de la UEFA';
  scrums : Scrum[];
  selectedScrum:Scrum;
  constructor(private scrumservice:ScrumService){ }

  getScrums():void{
    this.scrumservice.getScrums().then(scrums => this.scrums = scrums);
  }
}

```

```

ngOnInit():void{
    this.getScrums();
}

onSelect(scrum: Scrum): void {
    this.selectedScrum = scrum;
}

}

```

12. Ahora debemos modificar el archivo **app.module.ts** el contiene todos nuestros modulos y debe verse de la siguiente forma:

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/http';

import { AppComponent } from './app.component';
import { ScrumDetailComponent } from './scrum-detail.component';

@NgModule({
  declarations: [
    AppComponent,
    ScrumDetailComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

13. Ahora para que funcione nos falta el archivo index.html (en la raíz del proyecto), escribir el siguiente código:

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Uefa</title>
  <base href="/">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.6/css/bootstrap.min.css" integrity="sha384-
  rwoIResjU2yc3z8GV/NPeZWAv56rSmLldC3R/AZzGRnGxQQKnKkoFVhFQhNUwEyJ"
  crossorigin="anonymous">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

14. Para que nuestra pagina se vea con un buen diseño agregaremos en código css dentro del archivo app.component.css en el cual digitaremos el siguiente código:

15. Probar pagina desde consola digite el siguiente comando:
ng server

V. DISCUSION DE RESULTADOS

1. Crear una aplicación que determine el sueldo neto n empleados, tomando en cuenta que se le aplican los siguientes descuentos de ley a su sueldo base: + ISSS: tasa del 7.3% + Renta: 11% + AFP: 5.1% Muestre los montos de cada descuento y el sueldo final. Utilice buena presentación de la tabla y validación de la entrada del usuario.

2. Crear una aplicación web, donde el usuario pueda seleccionar entre tres tipos de combustible (regular, especial y diesel) haciendo uso de la siguiente información: La gasolina especial tiene un costo de \$4.25, la gasolina regular tiene un costo de \$4.05 y el diesel un costo de \$3.96. Además, debe haber un campo tipo number que permita un valor mínimo de 0.05 galones y un máximo de 150 galones. Además el tamaño de paso en las cantidades de galones debe ser de 0.05. cuando el usuario indique el tipo de combustible que se servirá y la cantidad de galones que se servirá (en variación de 0.05 ctvs) debe actualizarse otro campo de texto de solo lectura donde el usuario podrá ver el monto a pagar por el tipo y la cantidad de gasolina que se va a servir.

VII. BIBLIOGRAFIA

- Flanagan, David. JavaScript La Guía Definitiva. 1ra Edición. Editorial ANAYA Multimedia. 2007. Madrid, España.
- Tom Negrito / Dori Smith. JavaScript y AJAX para diseño web. Editorial Pearson Prentice Hall. Madrid, España. 2007.