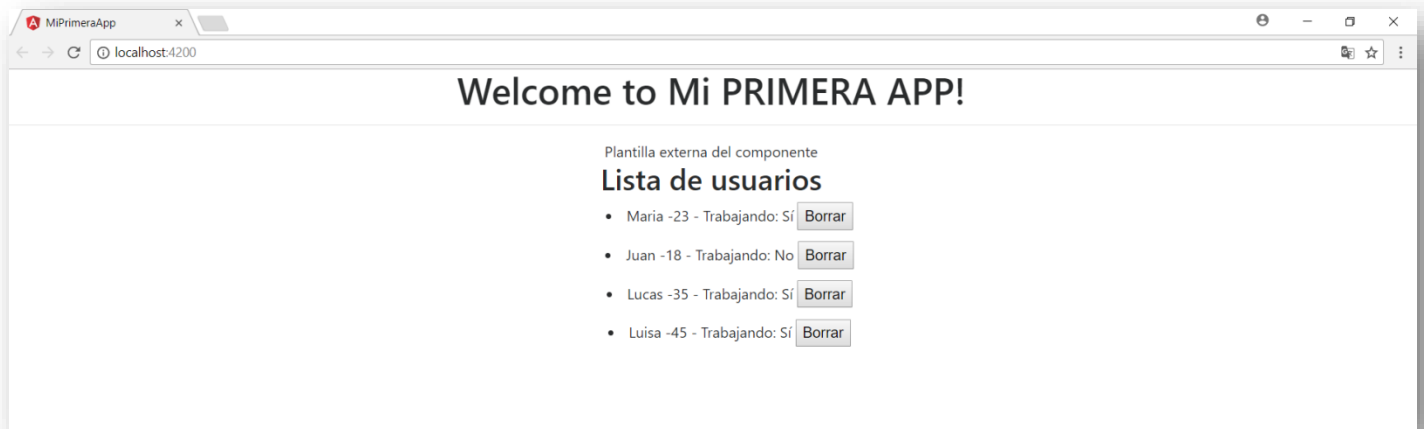


EJERCICIO 1:

A partir de la solución “[2.-Solución Ej2-UserList-con MODELO DATOS INCLUIDO](#)” que tenéis en el TEMA 17 – COMPONENTES Y PLANTILLAS DE ANGULAR que tenéis en el campus FP distancia que hace referencia a mostrar una lista de usuarios (su nombre, edad y si trabaja o no) usando para ello un componente hijo usuario, tenemos que modificarlo para que se muestre un botón ELIMINAR USUARIO que al pulsarlo me borre el usuario al cual acompaña.

Cómo debe aparecer en pantalla es así:



PASOS A SEGUIR:

Si seguisteis mis pasos, partiremos de esta situación:



Fijaros en la ubicación de las viñetas. Eso es porque por defecto se sitúan fuera del elemento `` (valor **outside** de la propiedad **list-style-position** aplicado al elemento ``). Vamos a cambiarlo para que quede al lado del texto, como en la imagen superior.

1º) Añadimos la regla CSS al archivo .css del componente “user-list.component”. Es decir, lo escribimos en el archivo “**user-list.component.css**”:

```
ul{  
  list-style-position: inside;  
}
```

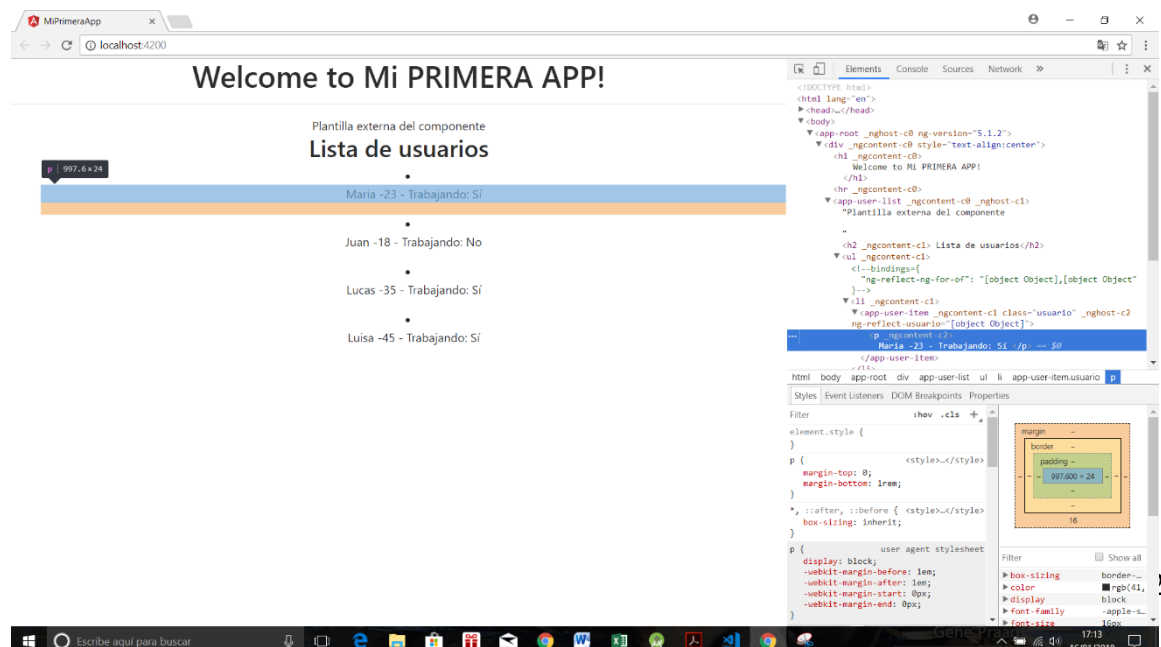
Si ejecutamos, veremos:



¿Qué ha pasado? ¿Por qué sale encima de cada usuario?

RESPUESTA: Ahora, nuestra viñeta esté dentro del elemento ``, pero este elemento contiene un componente también, que es: `<app-user></app-user>` y, **por defecto, los componentes se muestran con la propiedad “display:block”**, es decir, nuestro “María-23-Trabajando: Sí” ocupa todo el ancho y como no queda sitio, el punto se dibuja solo en otra línea.

Si pulsamos F12 lo podemos ver:



Para solucionarlo, debemos decirle en CSS, que la propiedad display el elemento <app-user> tenga valor: “inline-block”. Para ello:

- a) Creo en el archivo “**user-list.component.css**” una nueva clase “usuario” con la propiedad display:inline-block:

```
.usuario{  
display: inline-block;  
}  
  
ul{  
list-style-position: inside;  
}
```

- b) En el archivo “**user-list.component.html**” indico que <app-user> pertenece a la clase “usuario”:

```
<app-user-item class="usuario" [usuario]="usuario"></app-user-item>
```

Y ya tenemos bien colocado nuestro HTML. Vamos con el ejercicio.

Puesto que el listado de usuarios está formado por 2 componentes: el padre que representa el listado () y el hijo que representa cada uno de los usuarios (<app-user-item>), **HAY 2 FORMAS POSIBLES DE AÑADIR EL BOTÓN BORRAR:**

- A) **Añadir el botón en el componente padre (archivo *user-list.component.html*),** con lo cual iría dentro del elemento que es el que se repite con cada usuario. En este componente tenemos también la lista de usuarios (el array), con lo cual sería muy fácil borrar ese usuario de la lista.
- B) **Añadir el botón en el componente hijo (archivo *user-item.component.html*),** entendiendo que cada hijo (usuario) gestiona su propio botón de borrar. En este caso se añade el botón dentro del <p> que escribe los datos del usuario y el borrado ya no es tan sencillo, pues no tenemos acceso a la lista de usuario que está en el componente padre para autoborrarnos. Por otro lado, **tampoco tiene sentido que un elemento se autoborre a si mismo. Lo lógico es que un componente le diga a su padre, bórrame!**. Para conseguir que el hijo avise a su padre de que quiere ser borrado, se hace con una **SALIDA (OUTPUT)** que debemos definir en el hijo. Eso representa un evento que el hijo emite cuando alguien pulse su botón de borrar. El padre recogerá ese evento y actuará en consecuencia, borrándolo del array de usuario (o si fuese de una BD).

Os voy a mostrar cómo serían las dos soluciones (lo haré con 2 proyectos distintos, pues no son compatibles ambas). Partiremos para las dos de la situación en este momento.

SOLUCIÓN MODO A: Añadir el botón BORRAR en el componente PADRE

2º) Partimos de la situación inicial conseguida al final del punto 1º) en la página 3. Debemos añadir un botón en HTML al lado del usuario, por lo tanto, lo escribimos en el archivo “**user-list.component.html**” y lo situamos dentro del elemento :

Plantilla externa del componente

```
<h2> {{nombre_componente}}</h2>

<ul>
  <li *ngFor="let usuario of usuarios">
    <app-user-item class="usuario" [usuario]="usuario"></app-user-item>
    <button>Borrar</button>
  </li>
</ul>
```

3º) Debemos añadir funcionalidad al botón, de manera que al pulsarlo me borre el usuario que tiene al lado. Para ello, debemos programar el evento (click) del botón y eso se hace en dos pasos:

3.1) En el archivo “**user-list.component.html**” indicamos qué método va a responder al evento (clic). Será un método que llamaré “**borrarUsuario(usuario)**”, y que recibe el usuario a borrar:

```
<button (click)="borrarUsuario(usuario)">Borrar</button>
```

3.2) En el archivo correspondiente a la lógica de nuestro componente, es decir, en el archivo “**user-list.component.ts**” definimos nuestro método “**borrarUsuario(usuario)**” así:

```
import { Component, OnInit } from '@angular/core';
import { User } from '../user';

@Component({
  selector: 'app-user-list',
  templateUrl: './user-list.component.html',
  styleUrls: ['./user-list.component.css']
})
export class UserListComponent implements OnInit {
  public nombre_componente = 'Lista de usuarios';
  usuarios: Array<User>;

  constructor() {
    this.usuarios = [
```

```
new User('Maria', 23, true),
new User('Juan', 18, false),
new User('Lucas', 35, true),
new User('Luisa', 45, true)
];
}
borrarUsuario(user: User) {
  this.usuarios.splice(this.usuarios.indexOf(user), 1);
}

ngOnInit() {
}

}
```

Y ya está.

SOLUCIÓN MODO B: Añadir el botón BORRAR en el componente HIJO y usar SALIDAS OUTPUT

2º) Partimos de la situación inicial conseguida al final del punto 1º) en la página 3. Debemos añadir un botón en HTML al lado del usuario, por lo tanto, lo escribimos en el archivo “**user-item.component.html**” y lo situamos dentro del elemento `<p>`:

```
<p>
  {{usuario.nombre}} - {{usuario.edad}} -
  Trabajando: {{usuario.trabaja ? "Sí": "No"}}
  <button>Borrar</button>
</p>
```

3º) Debemos añadir funcionalidad al botón, de manera que al pulsarlo me borre el usuario que tiene al lado. Para ello, debemos programar el evento (click) del botón y eso se hace en tres pasos:

3.1) En el archivo “**user-item.component.html**” indicamos qué método va a responder al evento (clic). Será un método que llamaré “**onBorrar()**”:

```
<button (click)="onBorrar()">Borrar</button>
```

3.2) En el archivo correspondiente a la lógica de nuestro componente, es decir, en el archivo “**user-item.component.ts**” definimos nuestro método “**onBorrar()**”. Pero

para definirlo, primero necesitamos definir una salida OUTPUT que represente el aviso o evento que queremos lanzar y dentro del método onBorrar() lo emitiremos:

- a) Declaramos una variable de instancia con **@OUTPUT** que representa el **evento** y la llamaremos “borrar” y sobre ella crearemos el objeto **EventEmitter**.

CUIDADO CON LOS IMPORTS AUTOMÁTICOS:

```
import { Output } from '@angular/core';
import { EventEmitter } from '@angular/core';
import { Component, OnInit, Input } from '@angular/core';
import { User } from '../user';
```

```
export class UserItemComponent implements OnInit {
  @Input() usuario:User;
  @Output()
  borrar=new EventEmitter<User>();

  constructor() {
    // otros datos
  }
```

- b) Definimos el método **onBorrar()** para que emita el evento “borrar” que hemos declarado en el punto anterior:

```
import { Output } from '@angular/core';
import { EventEmitter } from '@angular/core';
import { Component, OnInit, Input } from '@angular/core';
import { User } from '../user';

@Component({
  selector: 'app-user-item',
  templateUrl: './user-item.component.html',
  styleUrls: ['./user-item.component.css']
})
export class UserItemComponent implements OnInit {
  @Input() usuario:User;
  @Output()
  borrar=new EventEmitter<User>();

  constructor() {
    // otros datos
  }

  onBorrar(){
    this.borrar.emit(this.usuario); // emite a su padre el usuario entero
  }
```

```
ngOnInit() {  
  }  
  
}
```

3.3) Ahora debemos decirle al componente padre que recoja ese evento cuando suceda. Por tanto, debemos modificar tanto el .html como el .ts del padre. Es decir **user-list.component**. Vamos a comenzar con el html del padre (**user-list.component.html**). Tendremos que modificar la etiqueta correspondiente al componente hijo, es decir `<app-user-item>` para indicar que este componente emite una salida o evento llamado “borrar”:

```
Plantilla externa del componente  
<h2>{{ nombre_componente }}</h2>  
  
<ul>  
  <li *ngFor="let usuario of usuarios">  
    <app-user-item class="usuario"  
      [usuario]="usuario"  
      (borrar)="borrarUsuario($event)"></app-user-item>  
  </li>  
</ul>
```

`$event` → representa “aquello” que nos entregue el hijo cuando emita el evento. Puede ser un objeto, un tipo primitivo,...

Además, debemos programar la lógica de ese método **borrarUsuario(\$event)** que será el que se ejecute cuando se pulse el botón BORRAR sobre el hijo. Lo haremos en el fichero **user-list.component.ts**:

```
import { Component, OnInit } from '@angular/core';  
import { User } from '../user';  
  
@Component({  
  selector: 'app-user-list',  
  templateUrl: './user-list.component.html',  
  styleUrls: ['./user-list.component.css']  
})  
export class UserListComponent implements OnInit {  
  public nombre_componente = 'Lista de usuarios';  
  usuarios: Array<User>;  
  
  constructor() {  
    this.usuarios = [  
      new User('Maria', 23, true),  
      new User('Juan', 18, false),  
      new User('Lucas', 35, true),  
      new User('Luisa', 45, true)
```

```
];  
}  
  
borrarUsuario(user:User){  
  this.usuarios.splice(this.usuarios.indexOf(user),1);  
}  
  
ngOnInit() {  
}  
}
```

EJERCICIO 2:

Modificar el ejercicio 1 que acabáis de hacer para que sólo muestre aquellos usuarios que trabajan. Es decir, Juan no debe salir.

PASOS A SEGUIR:

Hay dos posibles formas de hacerlo diferentes:

- 1) Usando la directiva estructural ***ngIf**, con lo que **el elemento del DOM sobre el que se sitúe, no será generado** si no trabaja. Es decir, realmente, **no existirá**.
- 2) Usando la **propiedad [hidden]** que **oculta** el elemento . Por lo tanto, **SÍ EXISTIRÁ**, sólo que no lo veremos. Esta opción tiene un **riesgo**, y es que **si hay alguna regla CSS en la app que afecte a todos los , puede ser que haga que nos aparezca**, aunque aquí hayamos puesto [hidden].

Vamos a hacerlo primero de la forma 1 y luego de la 2, para que veáis las diferencias:

FORMA 1: usando *ngIf:

Se usa de forma muy parecida al *ngFor. Se sitúa sobre un elemento HTML y la sintaxis es:

***ngIf** = “condición”

Lo que hace es que sólo se mostrará ese elemento si la condición es cierta.

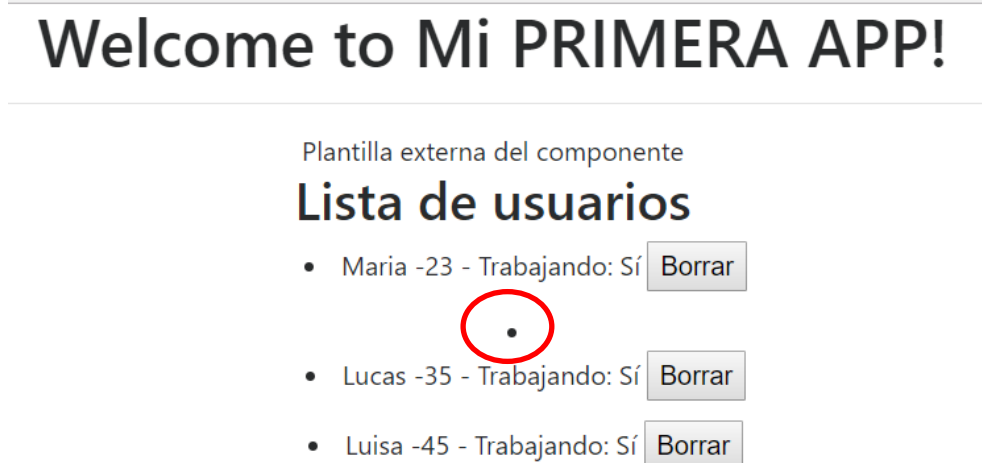
OJO: No se pueden poner dos directivas estructurales al mismo elemento. Es decir, *ngFor y *ngIf no pueden ir en el mismo elemento.

En nuestro caso, la condición será que trabaje.

Yo quiero no generar el `` si el usuario trabaja, pero no puedo poner `*ngIf` en `` porque ya tengo un `*ngFor`. Un truco es englobar el `` dentro de un `<div>` y poner el `*ngIf` a ese `<div>`. Pero si hacemos eso, tenemos otro problema: que estamos fuera del `for` y, por tanto, la variable `USUARIO` es desconocida. Con lo cual, sólo podemos hacerlo dentro del ``, por ejemplo en `<app-user>`, o añadiendo un `div` con la clase `usuario` para que `display` no sea `block`. Nos quedará:

```
<ul>
  <li *ngFor="let usuario of usuarios">
    <div class="usuario" *ngIf="usuario.trabaja">
      <app-user-item class="usuario" [usuario]="usuario"></app-user-item>
      <button (click)="borrarUsuario(usuario)">Borrar</button>
    </div>
  </li>
</ul>
```

Esto funcionará a medias. No veremos el usuario, pero se muestra esto:



Eso es porque **no se genera el div, con lo cual ni `<app-user>` ni el botón, pero el `` que es el que lleva “•” SÍ SE GENERA.**

PODEMOS SOLUCIONARLO usando el elemento de agrupamiento `<ng-container>` que ofrece ANGULAR para estos casos y que tiene las siguientes ventajas:

Ventajas de `<ng-container>`:

- ✓ No aparece en el DOM, pues es de Angular sólo.
- ✓ No interfiere con las CSS que pudiera haber; no le afectan.
- ✓ No es una directiva, ni clase, ni componente
- ✓ Permite simular un `*ngIf` combinado con `*ngFor` como si estuviesen en el mismo elemento.

De esta forma nos quedaría:

```
Plantilla externa del componente

<h2>{{ nombre_componente }}</h2>

<ul>
  <ng-container *ngFor="let usuario of usuarios">
    <ng-container *ngIf="usuario.trabaja">
      <li>
        <app-user-item class="usuario" [usuario]="usuario"></app-user-item>
        <button (click)="borrarUsuario(usuario)">Borrar</button>
      </li>
    </ng-container>
  </ng-container>
</ul>
```

FORMA 2: usando [hidden]:

Debemos ponerla en el elemento que queramos ocultar. Su sintaxis es:

[hidden] = “condición a cumplirse o valor booleano”.

En nuestro caso será “!usuario.trabaja”:

```
<ul>
  <li *ngFor="let usuario of usuarios" [hidden]="!usuario.trabaja">
    <app-user-item class="usuario" [usuario]="usuario"></app-user-item>
    <button (click)="borrarUsuario(usuario)">Borrar</button>
  </li>
</ul>
```

Ahora está PERFECTO también. Pero es peor solución:

Welcome to Mi PRIMERA APP!

Plantilla externa del componente

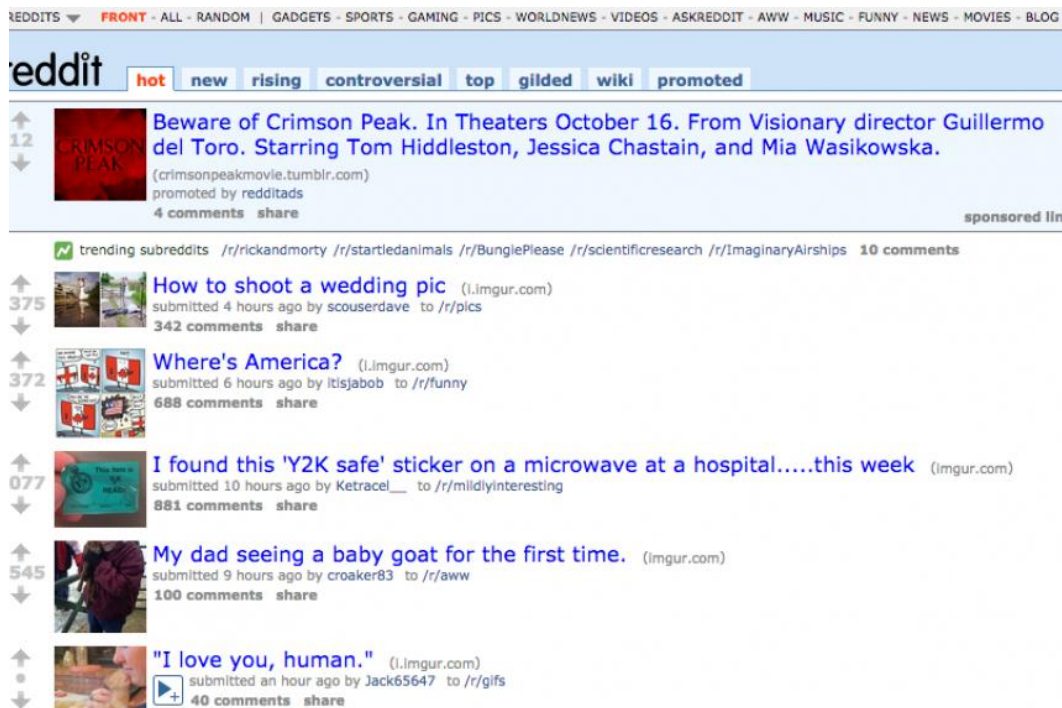
Lista de usuarios

- Maria -23 - Trabajando: Sí
- Lucas -35 - Trabajando: Sí
- Luisa -45 - Trabajando: Sí

EJERCICIO 3:

Vamos a construir un clon de Reddit. Para los que no lo conozcáis, Reddit es un sitio web de marcadores sociales y agregador de noticias donde los usuarios pueden dejar enlaces a contenidos web. Otros usuarios pueden votar a favor o en contra de los enlaces, haciendo que aparezcan más o menos destacados. Se trata de un mapa de discusión.

El aspecto de Reddit es:



Nuestro clon tendrá el siguiente aspecto:

Angular 4 Mi Reddit

Add a Link

Title

Link

Submit

3
POINTS

Angular 4
↑ upvote ↓ downvote

2
POINTS

FullStack
↑ upvote ↓ downvote

1
POINTS

Angular HomePage
↑ upvote ↓ downvote

¿Cómo lo hacemos?

1º) Pensar la descomposición en componentes:

En nuestro caso habrá 2:

- 1- Que contiene la aplicación entera: el formulario superior y la lista de artículos. Esto en realidad serían dos componentes. Pero de momento por no complicar la comunicación entre ellos, lo haremos así.
- 2- Cada fila de artículo será un componente que muestra un artículo y lo gestiona.

PASOS A SEGUIR:

1º) Crear una nueva app. Desde consola escribimos:

➤ `ng new angular-reddit`

2º) Buscamos una imagen que sea logo de nuestro Reddit, la descargamos y la guardamos en `assets/imágenes`.

3º) Instalamos Bootstrap:


First install `bootstrap` from npm:

```
$ npm install bootstrap --save
```

The simplest method is by using the `src/styles.css` file and adding the following code:

```
@import "~bootstrap/dist/css/bootstrap.css";
```

4º) Buscamos en Bootstrap 4 cómo se hace una barra de navegación que lleve una imagen y el nombre de la empresa. Será algo como:

 Bootstrap

```
<!-- Image and text -->
<nav class="navbar navbar-light bg-faded">
  <a class="navbar-brand" href="#">
    
    Bootstrap
  </a>
</nav>
```

4º) Copiamos ese código en nuestro `src/index.html` y lo modificamos cambiando la imagen por la nuestra y nuestra ruta y el texto por nuestro texto: “Angular 4 Mi Reddit”. Nos quedará:

```
<!doctype html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>AngularReddit</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
```

```
<body>
  <nav class="navbar navbar-light bg-faded">
    <a class="navbar-brand" href="#">
       Angular 4 Mi Reddit
    </a>
  </nav>
  <app-root></app-root>
</body>

</html>
```

5º) Vamos a src/app/app.component.html y quitamos todo el código que viene de ejemplo:

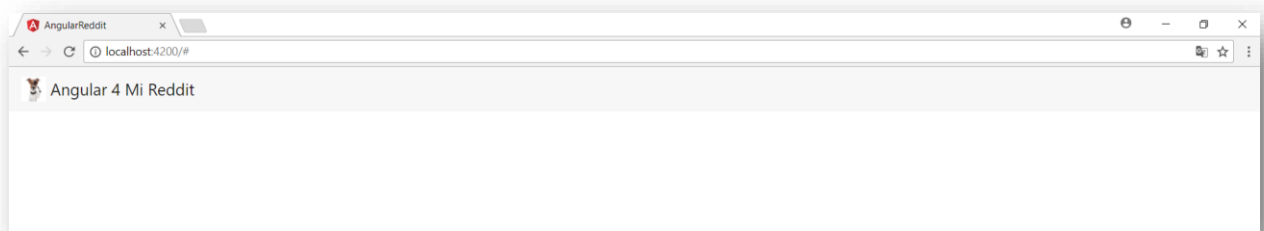
```
<!--The content below is only a placeholder and can be replaced.-->
```

6º) Vamos a src/app/app.component.ts y quitamos la propiedad title y su asignación de valor.
No la necesitamos. Nos quedará:

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

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

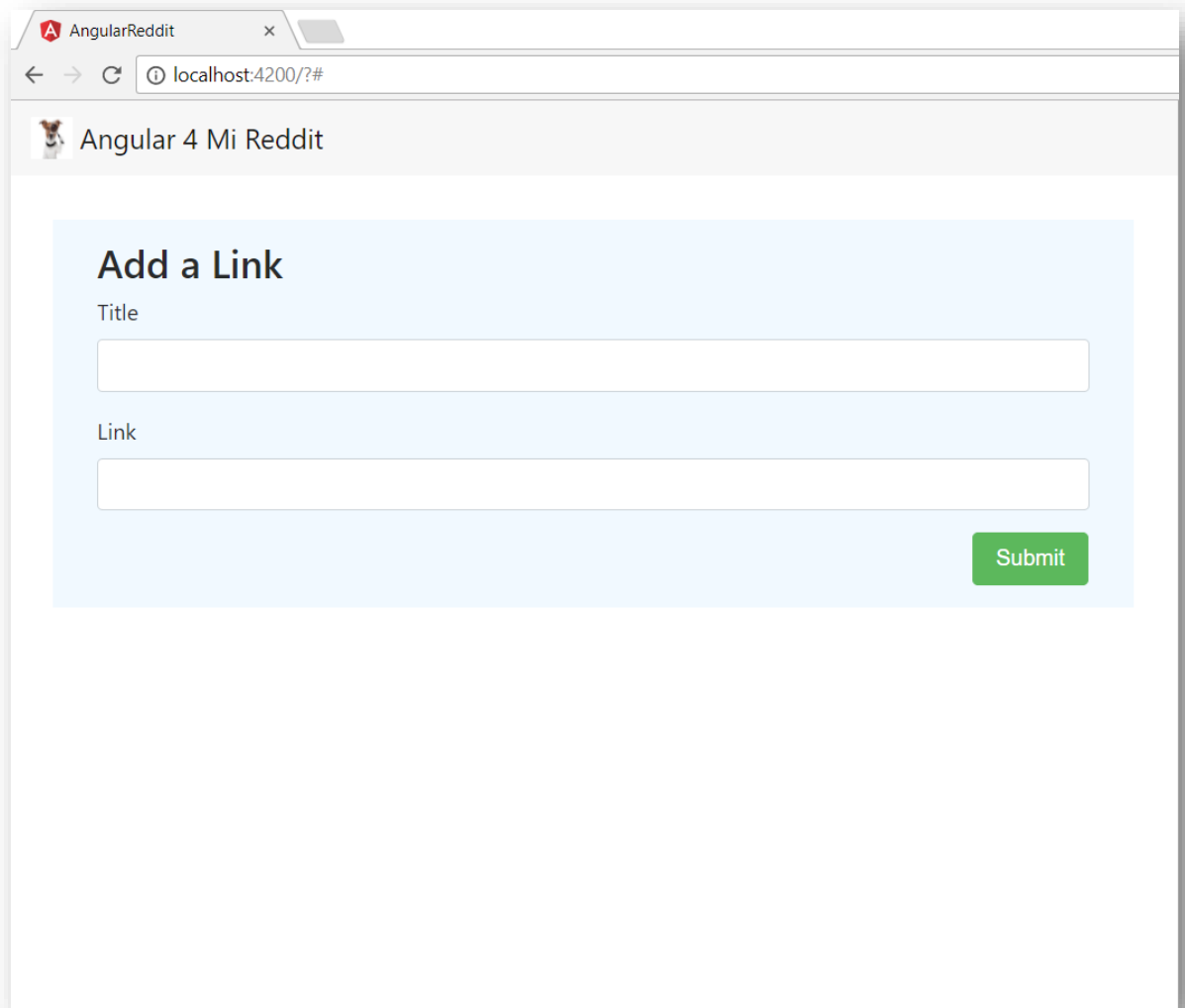
Si lo ejecutamos con “ng serve” veremos:



7º) Empezamos a diseñar nuestro **PRIMER COMPONENTE**: que incluirá un formulario para introducir los datos de un link y debajo una lista de artículos. NO VAMOS A

CREAR UN NUEVO COMPONENTE. Vamos a **REUTILIZAR** `src/app/app.component`. Por tanto:

7.1. Vamos a hacer el **diseño primero**, entonces nos situamos en `src/app/app.component.html` y creamos un formulario como éste:

A screenshot of a web browser window. The browser's address bar shows 'localhost:4200/?#'. The page title is 'Angular 4 Mi Reddit'. The main content area has a light blue background and contains a form titled 'Add a Link'. The form has two input fields: 'Title' and 'Link'. Below the 'Link' field is a green 'Submit' button.

El código será:

```
<form>
  <h3>Add a Link</h3>
  <div class="form-group">
    <label for="title">Title</label>
    <input class="form-control" type="text" id="title">
  </div>
  <div class="form-group">
    <label for="link">Link</label>
    <input class="form-control" type="url" id="link">
  </div>
  <button type="submit">Submit</button>
</form>
```

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

A parte de este código hay que añadir algo de ayuda para diseño en CSS: márgenes y padding. ¿A qué CSS lo ponemos? Como el form va a ser la página principal, se puede poner en **src/styles.css**:

```
form{
  background-color: #F1F9FF;
  margin: 2em;
  padding-left: 2em;
  padding-right: 2em;
  padding-top:1em;
  padding-bottom:1em;
}

button[type="submit"]{
  margin-right:1em;
}
```

7.2. De momento tenemos un formulario con un botón que no hace nada. Vamos a añadir código al botón:

- Tenemos que modificar el html para enlazar el <button> con el evento “onClick”
- Tenemos que poner el código Typescript que ejecute lo que pasa al hacer click.

Vamos al archivo **src/app/app.component.html** y añadimos al botón lo siguiente:

(click)="addArticle(¿?????)

Tenemos que pasar a AddArticle los datos del artículo a añadir. Pero ¿de dónde los sacamos? Pues de los dos <inputs> que hay en el HTML. Y ¿cómo lo hacemos? Pues, siguiendo lo explicado en los apuntes 6-User-Input, podemos crear una variable y asociarla a los inputs con la sintaxis:

#variable

Por tanto, nos quedará:

```
<form>
  <h3>Add a Link</h3>
  <div class="form-group">
    <label for="title">Title</label>
    <input #title class="form-control" type="text" id="title">
  </div>
```



```
<div class="form-group">
  <label for="link">Link</label>
  <input #link class="form-control" type="url" id="link">
</div>
<div class="row">
  <button type="submit" (click)="addArticle(title,link)" class="btn btn-
success ml-auto">Submit</button>
</div>
</form>
```

Al hacer #title y #link estamos enlazando el elemento <input> a una variable de nombre title en un caso y de nombre link en el otro. Cada una de esas variables van a ser **objetos que representan ese elemento DOM input**, así, el tipo de datos de #title y de #link es: **HTMLInputElement**. No son el valor del INPUT.

Como #title y #link son objetos que representan <inputs>, contendrán toda la información (atributos) de los correspondientes inputs. Por eso habrá que hacer luego:

title.value

para extraer el valor escrito en el input

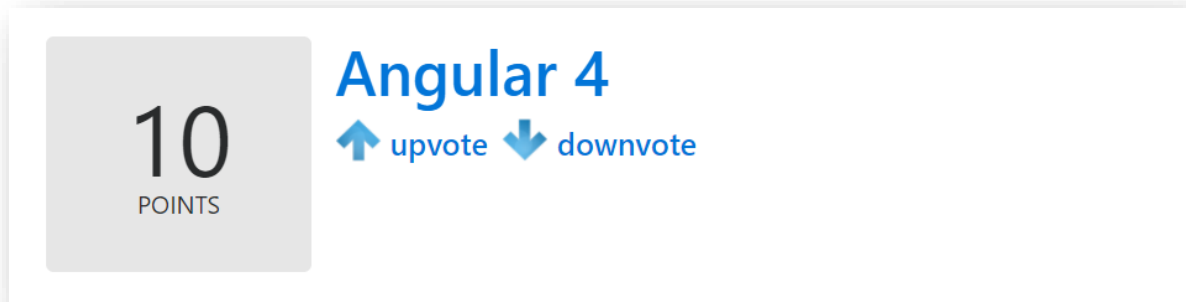
Ahora, en nuestro código .ts tenemos que definir la función addArticle(...) que será llamada cuando se pulse el botón submit. Nos vamos al archivo **src/app/app.component.ts** y escribimos de momento

```
export class AppComponent {

  addArticle(titleParam: HTMLInputElement, linkParam: HTMLInputElement):boolean
{
  console.log(`añadiendo artículo ${titleParam} y link: ${linkParam}`);
  return false;
}
}
```

8º) **AÑADIENDO EL COMPONENTE ARTÍCULO**

Ahora que ya tenemos el formulario para añadir nuevos artículos, necesitamos ir creando una lista de artículos. Pero para eso, primero necesitamos un nuevo COMPONENTE que represente cada uno de los artículos de la lista. Tendrá la forma:



Desde consola, tecleamos:

ng generate component article

Esto nos generará un componente llamado “article.component” con sus tres archivos:

- Vista (el .html)
- Estilos (el .css)
- Lógica o controlador (el .ts)

Vamos a empezar como antes definiendo el diseño del artículo.

8.1. Nos vamos a **src/app/article/article.component.html** y escribimos:

```
<div class="container">
  <div class="col-3 align-self-center votes">
    <div class="statistic">
      <div class="value">
        {{votes}}
      </div>
      <div class="label">
        Points
      </div>
    </div>
  </div>

  <div class="col-9 infoArticulo">
    <h1>
      <a href="{{link}}">
        {{title}}
      </a>
    </h1>
    <ul class="list-inline voters ">
      <li class="list-inline-item">
        <h5>
```

```
        <a href (click)="voteUp()">
             upvote
        </a>
    </h5>
</li>
<li class="list-inline-item">
    <h5>
        <a href (click)="voteDown()">
             downvote
        </a>
    </h5>
</li>
</ul>
</div>
</div>
```

Hemos tenido que añadir clases CSS también en src/styles.css:

```
form{
    background-color: #F1F9FF;
    margin: 2em;
    padding-left: 2em;
    padding-right: 2em;
    padding-top:1em;
    padding-bottom:1em;
}

button[type="submit"]{
    margin-right:1em;
}

.container{
    display:flex;
}

.statistic {
    margin: 1em 0;
    text-transform: uppercase;
    display: inline-block;
}

.value {
    font-size: 4em;
    line-height: 1em;
    text-align: center;
}
```

```
.label {
  color: inherit;
  display: block;
  font-size: 1em;
  text-align: center;
}

.infoArticulo{
  display:inline-block;
}

.posts {
  margin-top: 2em;
}

.votes {
  display: flex;
  align-items: center;
  justify-content: center;
  background-color: #E6E6E6;
  padding: 1em 0;
  border-radius: 5px;
}

.voters {
  clear: both;
  width: 100%;
}
```

8.2. Ahora vamos a definir la lógica del componente y sus propiedades. Por tanto, nos vamos al archivo **src/app/article/article.component.ts** y escribimos:

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

@Component({
  selector: 'app-article',
  templateUrl: './article.component.html',
  styleUrls: ['./article.component.css']
})
export class ArticleComponent implements OnInit {
  votes: number;
  title: string;
  link: string;

  constructor() {
```

```
this.title = 'Angular 4';
this.link = 'http://angular.io';
this.votes = 10; // representa la suma de los votos + menos los -
}

voteUp() {
  this.votes += 1;
}

voteDown() {
  this.votes -= 1;
}

ngOnInit() {
}
}
```

8.3. Ahora nos falta usar este componente “app-article” para que sea visible en la aplicación. Para ello tenemos que añadir la etiqueta `<app-article></app-article>` en algún html de la app. En nuestro caso, queremos que el AppComponent renderice este nuevo componente, Así que en la plantilla del AppComponent (*archivo `src/app/app.component.html`*) y justo después de cerrar `</form>`, añadimos:

```
<div class="post">
  <app-article class="row"></app-article>
</div>
```

SI LO DEJAMOS ASÍ, y pulsamos en VOTEUP o VOTEDOWN, veremos que se recarga la página. Esto sucede porque JavaScript, por defecto, PROPAGA LOS EVENTOS A TODOS LOS PADRES DE LOS COMPONENTES. Y entonces, el navegador está intentando seguir un enlace vacío. Para evitar esto:

- **Debemos modificar las funciones `voteDown()` y `voteUp()` para que devuelvan un booleano y será `false`.**

Nos quedará el archivo `src/app/article/article.component.ts`:

```
voteUp(): boolean {
  this.votes += 1;
  return false;
}

voteDown(): boolean {
```

```
this.votes -= 1;  
return false;  
}
```

8.4. **MODELO DE DATOS** artículo:

Vamos a crear una clase Article que será nuestro modelo de datos. Para eso, en la terminal escribimos:

➤ ng generate class article.model

Esto nos creará un fichero “article.model.ts” en src/app

En ese fichero, escribimos:

```
export class Article {  
  title: string;  
  link: string;  
  votes: number;  
  
  constructor(title: string, link: string, votes?: number) {  
    this.title = title;  
    this.link = link;  
    this.votes = votes || 0;  
  }  
}
```

8.5. Ahora tenemos que actualizar nuestro ArticleComponent (fichero: src/app/article/article.component.ts) para que use esta clase Article. Para ello:

1º) Importamos la clase:

```
import {Article} from '../article.model';
```

2º) En vez de almacenar las propiedades: title, votes y link sueltas, ahora tendremos una propiedad de tipo Article. Y el constructor por tanto, cambia también. Nos quedará el archivo:

```
import { Component, OnInit } from '@angular/core';  
import {Article} from '../article.model';  
  
@Component({  
  selector: 'app-article',  
  templateUrl: './article.component.html',  
})
```

```
    styleUrls: ['./article.component.css']
  })
  export class ArticleComponent implements OnInit {
    article: Article;

    constructor() {
      this.article = new Article('Angular 4', 'http://angular.io', 10);
    }

    voteUp(): boolean {
      this.article.votes += 1;
      return false;
    }

    voteDown(): boolean {
      this.article.votes -= 1;
      return false;
    }

    ngOnInit() {
    }
  }
}
```

3º) Al cambiar las propiedades del componente, también nos va a cambiar la plantilla: donde llamábamos votes, ahora será article.votes. Por tanto, el archivo **src/app/article/article.component.html** quedará:

```
<div class="container">
  <div class="col-3 align-self-center votes">
    <div class="statistic">
      <div class="value">
        {{article.votes}}
      </div>
      <div class="label">
        Points
      </div>
    </div>
  </div>

  <div class="col-9 infoArticulo">
    <h1>
      <a href="{{link}}">
        {{article.title}}
      </a>
    </h1>
    <ul class="list-inline voters ">
```

```
<li class="list-inline-item">
  <h5>
    <a href (click)="voteUp()">
       upvote
    </a>
  </h5>
</li>
<li class="list-inline-item">
  <h5>
    <a href (click)="voteDown()">
       downvote
    </a>
  </h5>
</li>
</ul>
</div>
</div>
```

8.6.La aplicación funciona pero **ALGO ESTÁ MAL**, los métodos `voteUp()` y `voteDown()` que están en el componente `article.component` **ROMPEN LA ENCAPSULACIÓN AL ACCEDER DIRECTAMENTE A PROPIEDADES DE LA CLASE ARTICLE como son votes**. Un componente no tiene por qué saber cómo está hecha por dentro una clase, sólo verá sus métodos, no sus propiedades.

SOLUCIÓN:

Trasladar los métodos `voteUp()` y `voteDown()` a la clase `Article` (pero devolviendo `void` en vez de `boolean`, pues aquí no hay eventos). Y luego en el componente `ArticleComponent` crear otros métodos que llamen a los de `Article`:

CÓDIGO DE `src/app/article.model.ts`:

```
export class Article {
  title: string;
  link: string;
  votes: number;

  constructor(title: string, link: string, votes?: number) {
    this.title = title;
    this.link = link;
    this.votes = votes || 0;
  }

  voteUp(): void {
    this.votes += 1;
  }
}
```



```
    voteDown(): void {  
        this.votes -= 1;  
    }  
}
```

CÓDIGO DE `src/app/article/article.component.ts`:

```
import { Component, OnInit } from '@angular/core';  
import { Article } from '../article.model';  
  
@Component({  
    selector: 'app-article',  
    templateUrl: './article.component.html',  
    styleUrls: ['./article.component.css']  
})  
export class ArticleComponent implements OnInit {  
    article: Article;  
  
    constructor() {  
        this.article = new Article('Angular 4', 'http://angular.io', 10);  
    }  
  
    voteUp(): boolean {  
        this.article.voteUp();  
        return false;  
    }  
  
    voteDown(): boolean {  
        this.article.voteDown();  
        return false;  
    }  
  
    ngOnInit() {  
    }  
}
```

8.7. Ahora vamos a ALMACENAR MÚLTIPLES ARTÍCULOS:

Vamos a cambiar el AppComponent para tener una lista de artículos, un array de artículos. Para ello, primero debemos importar la clase Article. El código de `src/app/app.component.ts` será:

```
import { Component } from '@angular/core';  
import { Article } from '../article.model';
```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  articles: Array<Article>; // propiedad

  constructor() {
    this.articles = [
      new Article('Angular 4', 'http://angular.io',3),
      new Article('FullStack', 'http://fullstack.io',2),
      new Article('Angular HomePage', 'http://angular.io', 1)
    ];
  }

  addArticle(titleParam: HTMLInputElement, linkParam:
HTMLInputElement):boolean {
    console.log(`añadiendo artículo ${titleParam} y link: ${linkParam}`);
    return false;
  }
}
```

8.8. Ahora debemos **modificar el componente ArticleComponent** para indicarle que reciba INPUTS desde fuera para el artículo mediante el paso de “un parámetro”. Será el archivo src/app/article/article.component.ts. Nos quedará:

```
import { Component, OnInit,
  Input } from '@angular/core';
import {Article} from '../article.model';

@Component({
  selector: 'app-article',
  templateUrl: './article.component.html',
  styleUrls: ['./article.component.css']
})
export class ArticleComponent implements OnInit {
  @Input() article: Article;

  constructor() {
    // sin código
  }

  voteUp(): boolean {
    this.article.voteUp();
    return false;
  }
}
```

```
}

voteDown(): boolean {
  this.article.voteDown();
  return false;
}

ngOnInit() {
}
}
```

8.9. Ahora debemos modificar el componente principal AppComponent para que muestre la lista de artículos mediante la llamada al componente ArticleComponent. El archivo **src/app/app.component.html** nos quedará:

```
<form>
  <h3>Add a Link</h3>
  <div class="form-group">
    <label for="title">Title</label>
    <input #title class="form-control" type="text" id="title">
  </div>
  <div class="form-group">
    <label for="link">Link</label>
    <input #link class="form-control" type="url" id="link">
  </div>
  <div class="row">
    <button type="submit" (click)="addArticle(title,link)" class="btn btn-success ml-auto">Submit</button>
  </div>
</form>

<div class="post">
  <app-article *ngFor="let article of articles"
    [article] = "article" class="row"></app-article>
</div>
```

Además, añadiremos a styles.css otra regla para row para separar los artículos unos de otros:

```
.row{
  margin-bottom: 2em;
}
```

8.10. Nos falta implementar **AÑADIR UN NUEVO ARTÍCULO:**

En el componente **AppComponent** tenemos una función `addArticle()` que no hace nada, salvo mostrar un mensaje en consola. Vamos a modificarlo para que coja los datos del formulario y lo añada al array `articles`. Tendremos:

```
addArticle(titleParam: HTMLInputElement, linkParam: HTMLInputElement):boolean
{
    console.log(`añadiendo artículo ${titleParam} y link: ${linkParam}`);
    this.articles.push(new Article(titleParam.value, linkParam.value, 0));
    titleParam.value = ''; // limpio pantalla
    linkParam.value = ''; // limpio pantalla
    return false;
}
```

[illegible]

I. Mostrar el dominio del artículo (de su url):

Lo mostraremos debajo del título de cada artículo. Para ello, en el modelo `Article` (archivo `src/app/article.model.ts`) creamos una nueva función que a partir del link extraiga el dominio:

```
domain(): string {
  try {
    // ejemplo: http://www.google.com/path/to/otro
    const domainPath: string = this.link.split('///')[1];
    // ejemplo: www.google.com/path/to/otro
    return domainPath.split('/')[0];
  } catch (err) {
    return null;
  }
}
```

Y la llamamos en la plantilla del componente `ArticleComponent` (archivo `src/app/article/article.component.html`):

```
<div class="meta">({{article.domain()}})</div>
```

II. ORDENAR LOS ARTÍCULOS POR PUNTUACIÓN:

Cuando votamos, no se reordenan.

Vamos a crear una función o método en AppComponent que ordene el array de artículos:

```
sortedArticles(): Array <Article> {  
    return this.articles.sort((a: Article, b: Article) => b.votes -  
a.votes);  
}
```

Y ahora, en la plantilla del AppComponent, cuando hagamos el for, iteraremos sobre el array ordenado:

```
<div class="post">  
  <app-article *ngFor="let article of sortedArticles()"  
    [article] = "article" class="row"></app-article>  
</div>
```