

### Creando nuestra primera aplicación

Usaremos **angular-cli** para crear y generar nuestros componentes. Generará servicios, enrutadores, componentes y directivas.

La **sintaxis** para crear un nuevo proyecto Angular con *Angular-cli* es:

```
ng new nombre_del_proyecto
```

- Debes sustituir **nombre\_del\_proyecto** por el nombre que tendrá tu proyecto.
- El proyecto se generará automáticamente.

Creemos nuestra aplicación de **tareas pendientes**, para ello ejecuta en la terminal y presiona **enter**:

```
ng new todo-app
```

Selecciona CSS como formato de hojas de estilo que se utilizarán en el proyecto

```
~/Documentos/angular
ng new todo-app
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS [ https://developer.mozilla.org/docs/Web/CSS ]
  Sass (SCSS) [ https://sass-lang.com/documentation/syntax#scss ]
  Sass (Indented) [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less [ http://lesscss.org ]
```

No utilizaremos SSR (Server Side Rendering), así que cuando nos pregunte colocamos N y simplemente presionamos Enter.

```
~/Documentos/angular
ng new todo-app
✓ Which stylesheet format would you like to use? CSS [ https://developer.mozilla.org/docs/Web/CSS ]
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? (y/N) N
```

Nos notificará cuando el proyecto este finalizado.

Luego, abre los archivos en tu editor VSCode.

Cambiate de directorio con el comando **cd** y el nombre de tu proyecto:

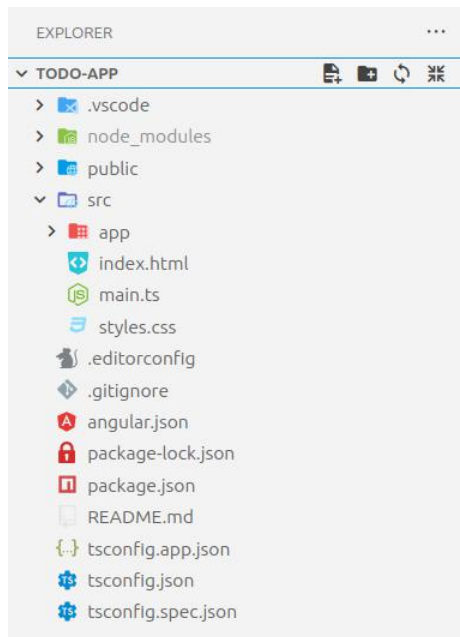
```
cd todo-app
```

Una vez dentro del proyecto ejecuta el comando **code .** y presiona **enter**:

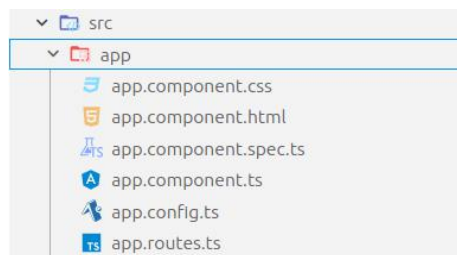
```
todo-app>code .
```

\* **nota que hay un espacio y luego un punto (.) en la instrucción.**

Así es como se debería ver la estructura de tu aplicación:



La mayor parte de nuestro trabajo estará en la carpeta **app**. Contiene seis archivos:



### Instalación de TailwindCSS

Para crear una interfaz agradable para nuestra aplicación, usaremos la Tailwindcss.



Para instalar Tailwind CSS, en la terminal ingresa el siguiente comando:

```
npm install tailwindcss @tailwindcss/postcss postcss --force
```


Una vez finalice el proceso te mostrará algo similar a esto:

```
~/Documentos/angular/todo-app git:(master) (4.092s)
npm install tailwindcss @tailwindcss/postcss postcss --force
npm warn using --force Recommended protections disabled.

added 13 packages, removed 2 packages, changed 1 package, and audited 946 packages in 4s

160 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Luego tenemos que crear un archivo de configuración postcss. En la raíz de tu proyecto y utilizando la opción “Nuevo archivo...” , agrega uno llamado **.postcssrc.json** (lleva un punto al inicio) Abre el archivo recién creado y agrega el código siguiente, guarda los cambios y cierra el archivo:

```
1 {  
2   "plugins": {  
3     "@tailwindcss/postcss": {}  
4   }  
5 }
```

Abre tu archivo `src/app/styles.css`, borra el comentario y agrega lo siguiente:

```
1 @import "tailwindcss";
```

Guarda los cambios y cierra el archivo.

Ahora ejecutaremos la aplicación desde tu terminal:

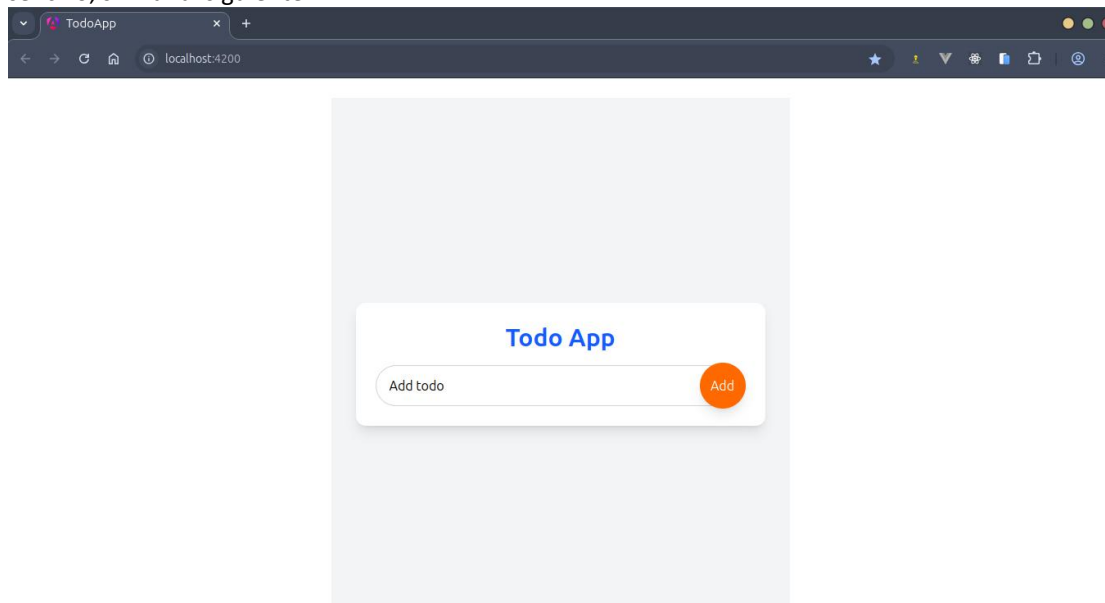
```
ng serve -o
```

La aplicación correrá en <http://localhost:4200/> y se mostrará en tu navegador predeterminado

¡¿Todo está bien?!

### Diseño inicial

Ahora hagamos una estructuración HTML. Usaremos clases de TailwindCSS para crear un formulario sencillo, similar al siguiente:



Abre el archivo `src/app/app.component.html` y **elimina completamente su contenido**, sustituyelo por el siguiente:

```
1 <div class="container mx-auto p-2">
2   <form>
3     <div class="flex justify-center items-center min-h-screen bg-gray-100">
4       <div class="bg-white shadow-lg rounded-xl p-6 w-[500px]">
5         <h1 class="text-center text-blue-600 text-3xl font-bold">Todo App</h1>
6
7         <!-- Input y Botón -->
8         <div class="flex items-center mt-4 relative">
9           <input type="text" class="flex-1 p-3 border border-gray-300 rounded-full outline-none px-4"
10             placeholder="Add todo">
11           <button
12             class="w-14 h-14 flex items-center justify-center bg-orange-500 text-white rounded-full
13             shadow-lg hover:bg-orange-600 absolute right-0 top-1/2 transform -translate-y-1/2">
14             Add
15           </button>
16         </div>
17       </div>
18     </div>
19   </form>
20 </div>
```

Abre el archivo `src/app/app.component.css` agrega las reglas siguientes:

```
1 body{
2   padding: 0;
3   margin: 0;
4 }
5
6 form{
7   max-width: 35em;
8   margin: 1em auto;
9 }
10
```

### Funcionalidad inicial

Agregaremos una vinculación de propiedad ([Property binding](#)) que enlaza el formulario de la plantilla (`<form>`) con un objeto **FormGroup** que será definido en el componente TypeScript.

Para obtener el valor de entrada en Angular cuando utilizamos **formularios reactivos**, podemos usar la directiva **formControlName**. Esta propiedad es la encargada de hacer el enlace con la lógica ([Directive binding](#)).

`formControlName="todo"` vincula un campo de entrada con un **FormControl** dentro de un **FormGroup**, permitiendo que Angular maneje su estado y validaciones de forma reactiva.

Luego, tenemos que agregar un evento de clic a nuestro botón, que empuja (push) el valor capturado en `"todoArray"` ([Una propiedad que crearemos luego en el typescript](#)).

Aprovecharemos y realizaremos modificaciones en las clases de estilos para proporcionarle un mejor aspecto a nuestro formulario:

```
1 <div class="container mx-auto p-2">
2   <form [formGroup]="todoForm">
3     <div class="flex justify-center items-center min-h-screen bg-gray-100">
4       <div class="bg-white shadow-lg rounded-xl p-6 w-[500px]">
5         <h1 class="text-center text-blue-600 text-3xl font-bold">Todo App</h1>
6
7         <!-- Input y Botón -->
8         <div class="flex items-center mt-4 relative">
9           <input type="text" class="flex-1 p-3 border border-gray-300 rounded-full outline-none px-4"
10             placeholder="Add todo" formControlName="todo">
11           <button (click)="add()"
12             class="w-14 h-14 flex items-center justify-center bg-orange-500 text-white rounded-full
13             shadow-lg hover:bg-orange-600 absolute right-0 top-1/2 transform -translate-y-1/2">
14             Add
15           </button>
16         </div>
17       </div>
18     </div>
19   </form>
20 </div>
```

Ahora tenemos que almacenar el valor capturado en la entrada. Podemos crear un arreglo vacío en `src/app/app.component.ts` dentro de la clase **AppComponent**.

Agregaremos lo necesario para trabajar con formularios reactivos:

- 1- Asegurate de revisar los imports en la parte superior del archivo
- 2- Agrega el módulo de formularios reactivos **ReactiveFormsModule** y **elimina el modulo RouterOutlet**.
- 3- Agrega la inyección de dependencias, el arreglo `todoArray`, la definición del formulario y el método para agregar un elemento al arreglo.

```

1  import { CommonModule } from '@angular/common';
2  import { Component, inject } from '@angular/core';
3  import { FormBuilder, ReactiveFormsModule, Validators } from '@angular/forms';
4
5  @Component({
6    selector: 'app-root',
7    imports: [CommonModule, ReactiveFormsModule],
8    templateUrl: './app.component.html',
9    styleUrls: ['./app.component.css']
10 })
11 export class AppComponent {
12   // Inyección de dependencias
13   private fb = inject(FormBuilder)
14
15   // Atributos o propiedades
16   todoArray: string[] = []
17
18   // Definición del formulario reactivo y sus validaciones
19   todoForm = this.fb.nonNullable.group({
20     todo: ['', Validators.required]
21   })
22
23   // Método para adicionar un todo
24   add(){
25     // Obtiene el valor del campo 'todo' dentro del formulario reactivo
26     const value = this.todoForm.get('todo')?.value
27
28     // Verifica que el valor no sea una cadena vacía ('') ni 'undefined'
29     if(value !== '' && value !== undefined){
30       // Agrega el valor al array 'todoArray'
31       this.todoArray.push(value)
32       // Imprime en la consola el array actualizado
33       console.log(this.todoArray)
34       // Limpia el input después de agregar
35       this.todoForm.reset();
36     }
37   }
38 }
39

```

### Obtener datos de “todoArray”

Ahora tenemos que recuperar los datos guardados en “todoArray.” Usaremos una directiva de control de flujo ([@For](#)) para recorrer el arreglo y extraer los datos.

También agregaremos las propiedades de enlace para el formulario reactivo, de tal forma que podamos obtener el valor enviado al invocar el evento click

En src/app/app.component.html:

```

1 <div class="container mx-auto p-2">
2   <form [formGroup]="todoForm">
3     <div class="flex justify-center items-center min-h-screen bg-gray-100">
4       <div class="bg-white shadow-lg rounded-xl p-6 w-[500px]">
5         <h1 class="text-center text-blue-600 text-3xl font-bold">Todo App</h1>
6
7         <!-- Input y Botón -->
8         <div class="flex items-center mt-4 relative">
9           <input type="text" class="flex-1 p-3 border border-gray-300 rounded-full outline-none px-4"
10             placeholder="Add todo" formControlName="todo">
11           <button (click)="add()"
12             class="w-14 h-14 flex items-center justify-center bg-orange-500 text-white rounded-full
13             shadow-lg hover:bg-orange-600 absolute right-0 top-1/2 transform -translate-y-1/2">
14             Add
15           </button>
16         </div>
17
18         <!-- Lista de tareas -->
19         <ul class="mt-6 space-y-3">
20           @for (todo of todoArray; track $index) {
21             <li class="flex items-center bg-gray-100 p-3 rounded-lg shadow-sm relative">
22               <!-- Barra lateral -->
23               <div class="w-1 bg-orange-500 absolute left-0 top-0 h-full rounded-l-lg"></div>
24
25               <!-- Contenido del ítem -->
26               <span class="flex-1 text-gray-700 pl-3">{{ todo }}</span>
27
28               <!-- Botón de eliminar -->
29               <button >
30                 <span class="text-gray-500 hover:text-red-500">del</span>
31               </button>
32             </li>
33           }
34         </ul>
35       </div>
36     </div>
37   </form>
38 </div>

```

Los datos ahora se recuperarán automáticamente cuando hagamos clic en el botón Agregar.

### Mejorando el diseño de la aplicación

Utilizaremos Google-fonts y los íconos de Material, que son de uso gratuito.

Incluye las fuentes de Google-fonts dentro de app.component.css:

```

1 @import url('https://fonts.googleapis.com/css?family=Raleway');
2
3 body{
4   padding: 0;
5   margin: 0;
6 }
7
8 form{
9   max-width: 35em;
10  margin: 1em auto;
11  font-family: "Raleway";
12 }

```

```
@import url('https://fonts.googleapis.com/css?family=Raleway');
```



Y los íconos de material dentro de `src/index.html`:

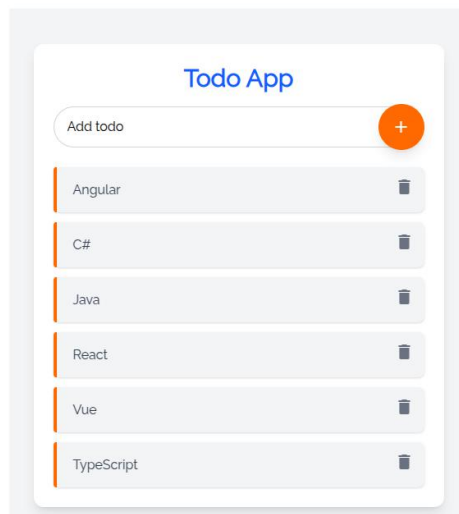
```

1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>TodoApp</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    <!-- Iconos material -->
10   <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
11 </head>
12 <body>
13   <app-root></app-root>
14 </body>
15 </html>

```

`<link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">`

Después de agregar algo de estilo a nuestra aplicación, esta se verá así:



Para usar los íconos de material:

Agrega los íconos “delete” y “add” en `src/app/app.component.html`:



```

1 <div class="container mx-auto p-2">
2   <form [formGroup]="todoForm">
3     <div class="flex justify-center items-center min-h-screen bg-gray-100">
4       <div class="bg-white shadow-lg rounded-xl p-6 w-[500px]">
5         <h1 class="text-center text-blue-600 text-3xl font-bold">Todo App</h1>
6
7         <!-- Input y Botón -->
8         <div class="flex items-center mt-4 relative">
9           <input type="text" class="flex-1 p-3 border border-gray-300 rounded-full outline-none px-4"
10             placeholder="Add todo" formControlName="todo">
11           <button (click)="add()"
12             class="w-14 h-14 flex items-center justify-center bg-orange-500 text-white rounded-full
13             shadow-lg hover:bg-orange-600 absolute right-0 top-1/2 transform -translate-y-1/2">
14             <span class="material-icons text-3xl">add</span>
15           </button>
16         </div>
17
18         <!-- Lista de tareas -->
19         <ul class="mt-6 space-y-3">
20           @for (todo of todoArray; track $index) {
21             <li class="flex items-center bg-gray-100 p-3 rounded-lg shadow-sm relative">
22               <!-- Barra lateral -->
23               <div class="w-1 bg-orange-500 absolute left-0 top-0 h-full rounded-l-lg"></div>
24
25               <!-- Contenido del ítem -->
26               <span class="flex-1 text-gray-700 pl-3">{{ todo }}</span>
27
28               <!-- Botón de eliminar -->
29               <button >
30                 <span class="material-icons text-gray-500 hover:text-red-500">delete</span>
31               </button>
32             </li>
33           }
34         </ul>
35       </div>
36     </div>
37   </form>
38 </div>

```

Nuestra aplicación está casi lista, pero necesitamos agregar algunas funciones. Una función de **Eliminación**, debería permitir a los usuarios hacer clic en el ícono y remover un elemento del arreglo.

También sería genial tener la opción de ingresar un nuevo elemento si el usuario presiona la tecla Enter, en lugar de hacer clic en el botón Agregar.

#### Eliminar elementos:

Para agregar la funcionalidad de eliminación, usaremos el elemento rastreado (tracked) de nuestra estructura de control de flujo @for, lo que permitirá que el usuario elimine del arreglo solo el elemento seleccionado pasando de manera enlazada (Binding) al método delete.

Agrega un evento (click) para el ícono eliminar y pasaremos "\$index" como parámetro:

```

1 <!-- Botón de eliminar -->
2 <button (click)="remove($index)">
3   <span class="material-icons text-gray-500 hover:text-red-500">delete</span>
4 </button>

```

En app.component.ts:

```
1  export class AppComponent {
2    // Inyección de dependencias
3    private fb = inject(FormBuilder)
4
5    // Atributos o propiedades
6    todoArray: string[] = []
7
8    // Definición del formulario reactivo y sus validaciones
9    todoForm = this.fb.nonNullable.group({
10     todo: ['', Validators.required]
11   })
12
13   // Método para adicionar un todo
14   add(){
15     // Obtiene el valor del campo 'todo' dentro del formulario reactivo
16     const value = this.todoForm.get('todo')?.value
17
18     // Verifica que el valor no sea una cadena vacía ('') ni 'undefined'
19     if(value !== '' && value !== undefined){
20       // Agrega el valor al array 'todoArray'
21       this.todoArray.push(value)
22       // Imprime en la consola el array actualizado
23       console.log(this.todoArray)
24       // Limpia el input después de agregar
25       this.todoForm.reset();
26     }
27   }
28
29   // Método para eliminar un todo
30   remove(index: number) {
31     this.todoArray.splice(index, 1);
32   }
33 }
```

### Capturando el envío del formulario

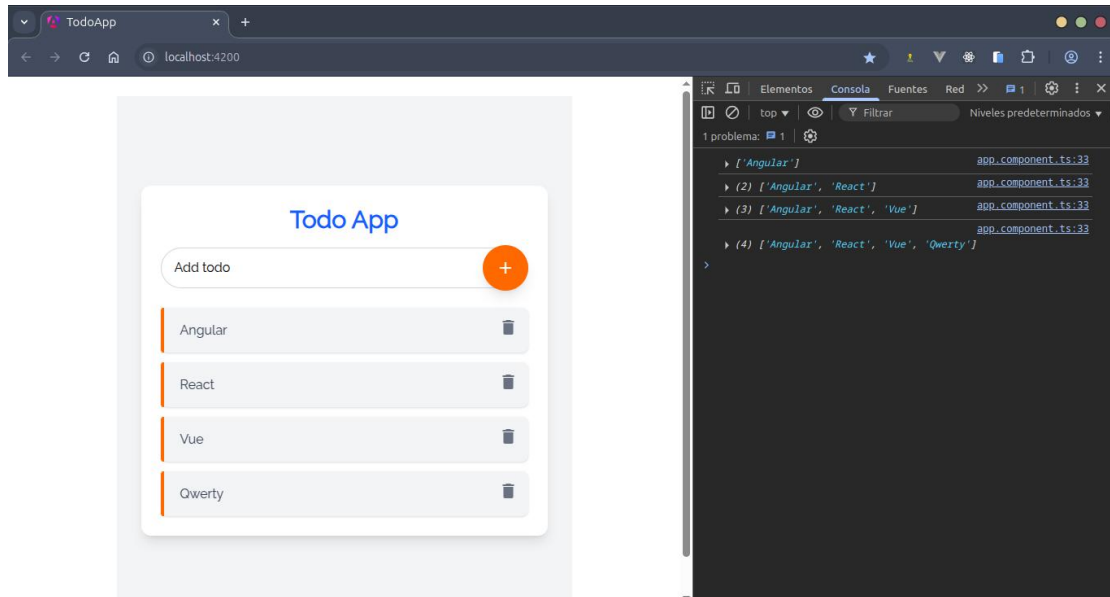
Podemos agregar un evento de envío de formulario en src/app/app.component.html:

```
1  <form [formGroup]="todoForm" (ngSubmit)="onSubmit()">
```

En src/app/app.component.ts justo debajo de donde agregaste el método remove, debes colocar lo siguiente:

```
1  // Método para adicionar un todo si se envía el formulario
2  onSubmit(){
3    // Capturamos todo el valor del formulario
4    const value = this.todoForm.value
5    // Imprimimos en consola el valor de la constante
6    console.log(value);
7  }
```

Verifica la consola (clic derecho en el navegador y luego selecciona inspeccionar). Devolverá un objeto de valores:



Así que ahora tenemos que enviar el valor devuelto a "todoArray":

```
1 // Método para adicionar un todo si se envía el formulario
2 onSubmit(){
3   const value = this.todoForm.get('todo')?.value
4   console.log(value);
5   if(value !== '' && value !== undefined){
6     this.todoArray.push(value)
7     this.todoForm.reset();
8   }
9   else{
10    alert('Field required **')
11  }
12 }
```

El valor se inserta sin necesidad de hacer clic en el botón Agregar, simplemente presionando "Enter".

Como habrás notado hay código repetitivo en dos métodos, realiza las modificaciones que consideres pertinentes para evitarlo.

Por último necesitamos evidenciar la autoria de tu aplicación por que será necesario que crees una nueva propiedad llamada **author** y que le asignes tu nombre completo y número de carnet por defecto.

```
1 export class AppComponent {
2   // Inyección de dependencias
3   private fb = inject(FormBuilder)
4
5   // Atributos o propiedades
6   todoArray: string[] = []
7   author: string = "John Doe Smith - DS100100"
```

Enlaza la propiedad en el html en la parte inferior del formulario, justo antes de la etiqueta de cierre `</form>`:

```
1 <!-- Autor -->
2 <div class="w-full mb-2 select-none rounded-t-lg border-t-4 border-blue-400
3   bg-blue-100 p-4 font-medium hover:border-blue-500">{{ author }}</div>
4 </form>
5 </div>
```

**Para la entrega:**

Comprime todos los archivos de tu proyecto pero evita agregar la carpeta **node\_modules** y la carpeta **.angular** ya que estas pesan demasiado y no será posible alojarlo en la plataforma virtual.