

# UNIVERSIDAD TECNOLÓGICA DEL VALLE DEL MEZQUITAL

## **TITULO DEL TRABAJO:**

Practica 02 Angular y Tailwind

## **MATERIA:**

Desarrollo Web

Ing. Santiago Labra Hernández

## **NOMBRE:**

Alison Yuridia Pérez Juárez

## **GRUPO:**

9 "B"

## **CARRERA:**

INGENIERÍA EN DESARROLLO Y GESTIÓN DE  
SOFTWARE

## **CUATRIMESTRE:**

MAYO – AGOSTO 2025

## Introducción

En esta práctica se realizó con Angular junto con la herramienta de diseño Tailwind CSS para crear una aplicación web sencilla, es un tipo lista de tareas "To do App". Aprendimos a configurar Tailwind en Angular, crear componentes, trabajar con señales (signals) para manejar datos de forma reactiva y aplicar filtros dinámicos. Se usaron formularios reactivos con validaciones, manejo del estado de cada tarea (como completada o en edición) y almacenamiento local en el navegador para guardar los cambios. Esta práctica permitió comprender cómo se combinan diseño y funcionalidad en una aplicación real con Angular.

Ponemos el siguiente comando para instalar Tailwind CSS “npm install -D tailwindcss postcss autoprefixer”

Abrimos el siguiente link <https://tailwindcss.com/docs/guides/angular> y creamos un archivo en raíz con el nombre de “.postcssrc.json” y pegamos el código que viene alado del archivo

03 Configure PostCSS Plugins

Create a `.postcssrc.json` file in the root of your project and add the `@tailwindcss/postcss` plugin to your PostCSS configuration.

```
.postcssrc.json
{
  "plugins": {
    "@tailwindcss/postcss": {}
  }
}
```

```
{ } .postcssrc.json X
{ } .postcssrc.json > ...
1  {
2    "plugins": {
3      "@tailwindcss/postcss": {}
4    }
5  }
```

En el styles.css de raíz ponemos el import de Tailwind CSS

```
# styles.css X
src > # styles.css > @tailwindcss
1  /* You can add global styles to this file, and also import other style files */
2  @import "tailwindcss";
3
```

Creamos el componente “todo”

```
src
└─ app
   └─ components \ todo
      # todo.component.css
      <> todo.component.html
      TS todo.component.spec.ts
      TS todo.component.ts
```

En “app.routes.ts” ponemos la ruta de todo, aquí se va a ver nuestro cuadro.

```
TS app.routes.ts X
C:\proyectos\practica2\src\app\app.routes.ts (preview)
1  import { Routes } from '@angular/router';
2  import { TodoComponent } from '../components/todo/todo.component';
3
4  export const routes: Routes = [
5    {
6      path: 'todo', component: TodoComponent},
7    {
8      path: "**", pathMatch: 'full', redirectTo: 'todo'
9    },
10
11  ];
```

En “app.component.html” dejamos solo la etiqueta de <router-outlet></router-outlet> para que se pueda dirigir a nuestra vista del archivo de “todo”

```
<> app.component.html X
src > app > <> app.component.html > router-outlet
Go to component
1  <router-outlet></router-outlet>
2
```

Creamos el modelo de “todo” para generar los campos que ocuparemos.

```
TS todo.ts X
src > app > modelos > TS todo.ts > Filtertype
1  export interface TodoModel {
2    id: number;
3    title: string;
4    completed: boolean;
5    editing: boolean;
6  }
7
8  export interface TodoModel {
9    id: number;
10   title: string;
11   completed: boolean;
12   editing: boolean;
13 }
14
15 export type Filtertype = 'all' | 'active' | 'completed';
```

En el `todo.component.ts` ponemos el siguiente código dentro de un `<section></section>` tenemos una sección donde vamos a poder crear una nueva tarea y filtrar las tareas que tengamos. Ocupamos el `<ul></ul>` para enlistar los datos que tenemos dentro del `checkbox`.

En los botones se pone el filtro activo para aplicar estilos con la clase `active`

```
src > app > components > todo > todo.component.html > section.todolist-wrapper
Go to component
1 <section class="todolist-wrapper">
2   <h1 (element) button: HTMLButtonElement
3   <div
4     The button element represents a button labeled by its contents.
5     Widely available across major browsers (Baseline since 2015)
6   </div>
7   <div MDN Reference
8     <button (click)="changeFilter('all')" [class.active]="filter() === 'all'">All</button>
9     <button (click)="changeFilter('active')" [class.active]="filter() === 'active'">Active</button>
10    <button (click)="changeFilter('completed')" [class.active]="filter() === 'completed'">Completed</button>
11  </div>
12  <ul class="todolist">
13    <li class="todo">
14      <input type="checkbox">
15      <label>Escribe el título</label>
16      <button>Edit</button>
17      <button>Delete</button>
18    </li>
19    <li class="todo-editing">
20      <input type="text" placeholder="Edit task">
21    </li>
22  </ul>
23 </section>
```

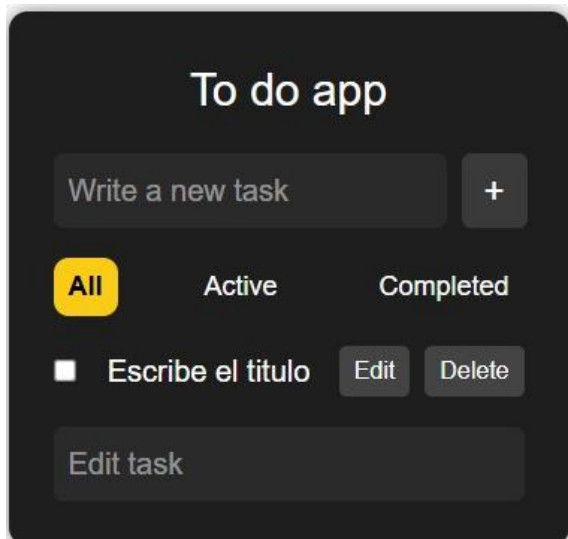
Agregamos estilos es el archivo `styles.css` principal en `src`

```
# styles.css X
src > # styles.css > .todolist-wrapper
1 /* You can add global styles to this file, and also import other style files */
2 @import "tailwindcss";
3
4 .todolist-wrapper {
5   background-color: #1e1e1e;
6   color: #ffffff;
7   padding: 1.5rem;
8   width: 300px;
9   margin: 2rem auto;
10  border-radius: 10px;
11  font-family: Arial, sans-serif;
12  box-shadow: 0 0 10px rgba(0, 0, 0, 0.5);
13 }
14
15 .title {
16   margin-bottom: 1rem;
17   font-size: 1.5rem;
18   text-align: center;
19 }
20
21 .new-todo {
22   display: flex;
23   gap: 0.5rem;
24   margin-bottom: 1rem;
25 }
```

Dentro de `todo.component.ts` vamos a poner la funcionalidad dentro de la clase `Component` ponemos `Signal` para ayudarnos a crear dos señales `todoList`: Aquí vamos a poner las tres tareas que nosotros queramos dentro de un arreglo, `filter`: aquí se pone por defecto `All` y `changeFilter`: Aquí vamos a poder actualizar el estado del filtro con `set()` cuando se haga clic en un botón de filtro.

```
TS todo.component.ts X
src > app > components > todo > TS todo.component.ts > Tod
13 export class TodoComponent {
14
15   todoList = signal<TodoModel[]>([
16     {
17       id: 1,
18       title: 'Comprar una laptop',
19       completed: true,
20       editing: false
21     },
22     {
23       id: 2,
24       title: 'Ir de compras',
25       completed: true,
26       editing: false
27     },
28     {
29       id: 3,
30       title: 'Comprar cafe',
31       completed: false,
32       editing: false
33     }
34   ]);
35
36   filter = signal<FilterType>('all');
37
38   changeFilter(filterString: FilterType) {
39     this.filter.set(filterString);
40   }
}
```

Así queda la primera parte de la practica



## Segunda parte de la practica

Agregamos lo que falta al “todo.component.html” para que tenga todas las funcionalidades en todo.component.ts agregamos distintas propiedades para la funcionalidad.

Se agrega la clase “todolist-wrapper” para estilizar con Tailwind que ya habíamos descargado, “input” para poner una tarea nueva y esta enlazado con FormControl, un evento “(keydown.enter)=”addTodo()” para poder editar una tarea al presionar la tecla de enter, el signo de + se agrega “(click)=”addTodo()” para poder añadir tareas nuevas, en el for tenemos una propiedad “todoListFiltered()” para poder mostrar las tareas, tenemos “@if todo.editing” para que si la tarea se esta editando nos muestre un input para escribir y guarda un nuevo titulo con “saveTitleTodo(todo.id, \$event)”, y “removeTodo(todo.id)” para poder eliminar tareas.

```
todo.component.html 1 x
src > app > components > todo > todo.component.html > section.todolist-wrapper > ul.todolist > li.todo > button.text-red-500
Go to component
1  <div class="todolist-wrapper">
2    <h1 class="title">To do app</h1>
3    <div class="new-todo">
4      <input type="text" (keydown.enter)="addTodo()" placeholder="Write a new task" [formControl]="newtodo">
5      <button (click)="addTodo()">+</button>
6    </div>
7    <div class="filters">
8      <button (click)="changeFilter('all')" [class.active]="filter() === 'all'">All</button>
9      <button (click)="changeFilter('active')" [class.active]="filter() === 'active'">Active</button>
10     <button (click)="changeFilter('completed')" [class.active]="filter() === 'completed'">Completed</button>
11   </div>
12   <ul class="todolist">
13     @for (todo of todoListFiltered(); track todo.id) {
14       @if (todo.editing) {
15         <li class="todo-editing">
16           <input type="text" placeholder="Edit task" [value]="todo.title" (keydown.enter)="saveTitleTodo(todo.id, $event)">
17         </li>
18       } @else {
19         <li class="todo">
20           <input type="checkbox" [checked]="todo.completed" (click)="toggleTodo(todo.id)">
21           <label>{{todo.title}}</label>
22           @if (!todo.completed) {<button (click)="updateTodoEditingMode(todo.id)">Edit</button>}
23           <button class="text-red-500" (click)="removeTodo(todo.id)">Delete</button>
24         </li>
25       }
26     }
27   </ul>
28 </div>
```

Agregamos para poder filtrar dinámicamente la lista, se utiliza una señal computarizada “todoListFiltered” esta analiza el valor actual del filtro y devuelve solo las tareas correspondientes al estado que este seleccionado.

```
TS todo.component.ts x
src > app > components > todo > TS todo.component.ts > TodoComponent
13 export class TodoComponent {
14
15
42   todoListFiltered = computed(() => {
43     const filter = this.filter();
44     const todos = this.todos();
45
46     switch (filter) {
47       case 'active':
48         return todos.filter((todo) => !todo.completed);
49       case 'completed':
50         return todos.filter((todo) => todo.completed);
51       default:
52         return todos;
53     }
54   })
55 }
```

En todo component.ts agregamos la propiedad “newTodo” aquí será donde se va escribir el nombre de una nueva tarea que queramos agregar.

```
TS todo.component.ts X
src > app > components > todo > TS todo.component.ts > TodoComponent
13 export class TodoComponent {
56     newtodo = new FormControl('', {
57       nullable:true,
58       validators:[
59         Validators.required, Validators.minLength(5)
60       ]
61     });
62 }
```

Agregamos “addTodo” aquí toma el valor del input, lo valida, y lo añade a la señal todolist usando el método update, y genera un nuevo identificador con Date.now() y luego limpia el campo.

```
TS todo.component.ts X
src > app > components > todo > TS todo.component.ts > TodoComponent > toggleTodo
13 export class TodoComponent {
63   addTodo() {
64     const newTodoTitle = this.newtodo.value.trim();
65     if (this.newtodo.valid && newTodoTitle !== '') {
66       this.todolist.update((prev_todos) => {
67         return [
68           ...prev_todos,
69           { id: Date.now(), title: newTodoTitle, completed: false, editing: false }
70         ];
71       });
72       this.newtodo.reset();
73     }
74 }
```

Agregamos “toggleTodo()” para poner una tarea como completada.

```
TS todo.component.ts X
src > app > components > todo > TS todo.component.ts > TodoComponent
13 export class TodoComponent {
76   toggleTodo (todoId:number) {
77     return this.todolist.update ((prev_todos) => {
78       prev_todos.map ((todo) => {
79         if (todo.id === todoId) {
80           return {
81             ...todo,
82             completed:!todo.completed,
83           };
84         }
85         return {...todo}
86       })
87     });
88 }
```

Agregamos “removeTodo()” para eliminarla de la lista.

```
TS todo.component.ts X
src > app > components > todo > TS todo.component.ts > TodoComponent > toggleTodo
13 export class TodoComponent {
90   removeTodo (todoId: number) {
91     this.todolist.update((prev_todos) => prev_todos.filter((todo) => todo.id !==todoId))
92   }
}
```



Agregamos “updateTodoEditingMode()” para poder activar la edición en una tarea específica.

```
TS todo.component.ts X
src > app > components > todo > TS todo.component.ts > TodoComponent
13 export class TodoComponent {
94   updateTodoEditingMode (todoId:number) {
95     return this.todoList.update ((prev_todos) =>
96       prev_todos.map((todo) => {
97         return todo.id === todoId
98           ? {...todo, editing: true}
99           : {...todo, editing: false}
100       })
101   )
102 }
```

Agregamos “saveTitleTodo()” que va a guardar el nuevo título cuando se presione la tecla enter.

```
TS todo.component.ts X
src > app > components > todo > TS todo.component.ts > TodoComponent > toggleTodo
13 export class TodoComponent {
104   saveTitleTodo (todoId:number, event:Event) {
105     const tittle = (event.target as HTMLInputElement).value;
106     return this.todoList.update ((prev_todos) => prev_todos.map((todo) => {
107       return todo.id === todoId ? {...todo, tittle, editing: false}
108       :todo;
109     }))
110   }
```

Agregamos para que se pueda guardar tareas en el almacenamiento local del navegador. En el constructor se usó effect() que va a guardar algún cambio en la lista dentro de localStorage.

```
TS todo.component.ts X
src > app > components > todo > TS todo.component.ts > TodoComponent > toggleTodo
13 export class TodoComponent {
112   constructor () {
113     effect(() => {
114       localStorage.setItem('todos', JSON.stringify(this.todoList))
115     })
116   }
```

Agregamos “ngOnInit()” revisa si ya existen tareas guardadas y se muestran.

```
TS todo.component.ts X
src > app > components > todo > TS todo.component.ts > TodoComponent
13 export class TodoComponent {
118   ngOnInit() {
119     const storage = localStorage.getItem('todos');
120     if (storage) {
121       this.todoList.set(JSON.parse(storage))
122     }
123   }
124 }
125
```

Y así queda

Tengo dos tareas completas y una incompleta

```
todolist = signal<TodoModel[]>([
  {
    id: 1,
    title: 'Comprar una laptop',
    completed: true,
    editing: false
  },
  {
    id: 2,
    title: 'Ir de compras',
    completed: true,
    editing: false
  },
  {
    id: 3,
    title: 'Comprar cafe',
    completed: false,
    editing: false
  }
])
```

### To do app

Write a new task +

All Active Completed

☒ Comprar una laptop Delete

☒ Ir de compras Delete

☐ Comprar cafe Edit Delete

### To do app

Write a new task +

All **Active** Completed

☐ Comprar cafe Edit Delete

### To do app

Write a new task +

All Active **Completed**

☒ Comprar una laptop Delete

☒ Ir de compras Delete

Eliminamos la tarea “ir de compras”

### To do app

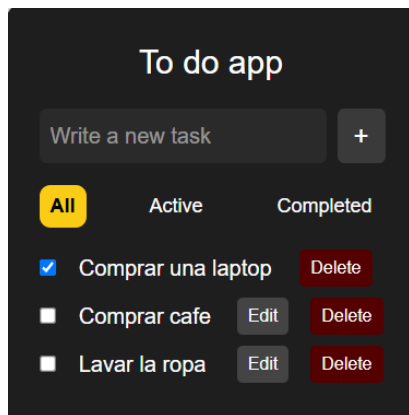
Write a new task +

**All** Active Completed

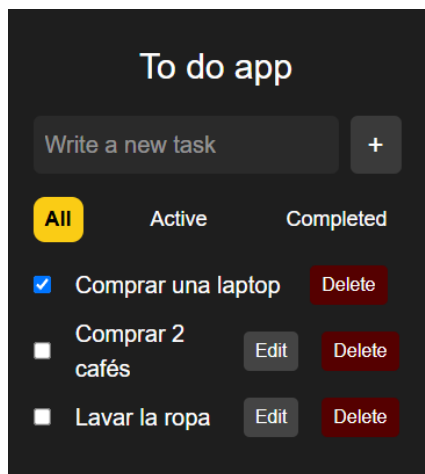
☒ Comprar una laptop Delete

☐ Comprar cafe Edit Delete

Agregamos la tarea “Lavar la ropa”



Editamos comprar un café y agregamos “comprar 2 cafés”



## Conclusión

Gracias a esta práctica aprendimos a integrar Tailwind CSS en Angular y a usar señales para manejar listas dinámicas de tareas. Se creó un componente que permite agregar, editar, eliminar y completar tareas, además de filtrarlas por el estado en el que estén. También se configuró el almacenamiento local para que las tareas no se pierdan al recargar la página. Obtuvimos una aplicación funcional. Esta experiencia fue útil para reforzar conocimientos sobre componentes, eventos, formularios y estilos con Tailwind en proyectos con Angular.