UNIVERSIDAD TECNOLOGICA 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:

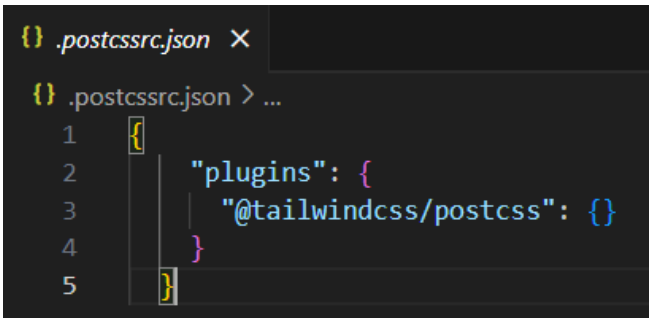
MAYO – AGOSTO 2025

Introducción

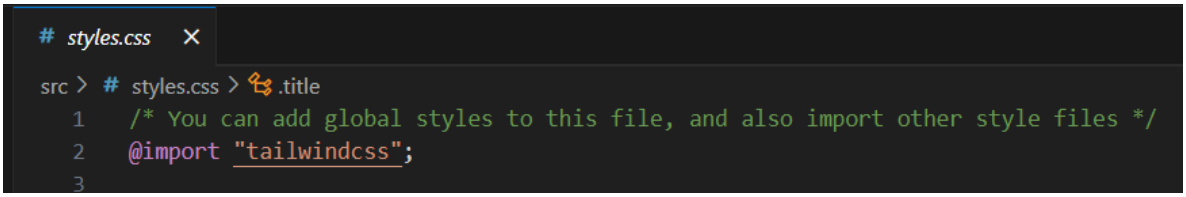
En esta práctica se realizo con Angular junto con la herramienta de diseño Tailwind CSS para crear una aplicación web sencilla, es un tipo lista de tareas “To do App”. Aprendimos a configurar Tailwind en Angular, crear componentes, trabajar con señales (signals) para manejar datos de forma reactiva y 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 installar Talwind CSS “npm install -D tailwindcss postcss autoprefixer”

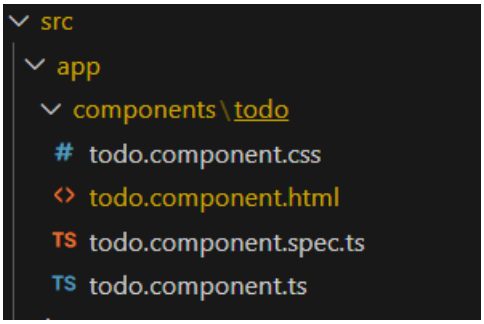
Abrimos el siguinte link https://tailwindcss.com/docs/guides/angular y creamos un archivo en raíz con el nombre de “.postcssrc.json” y pegamos el código que viene alado del archivo



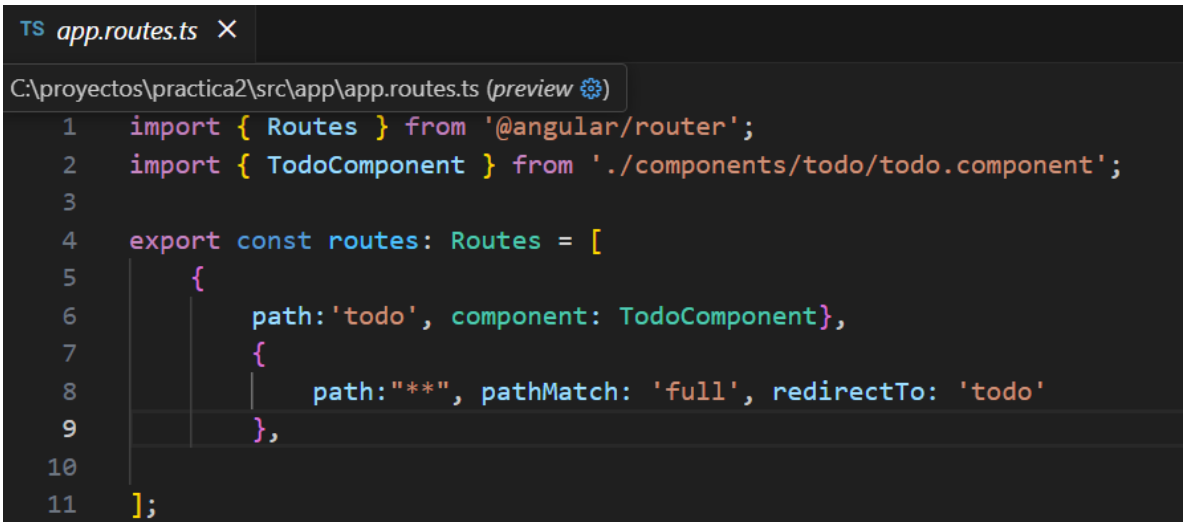
En el styles.css de raíz ponemor el import de Tailwind CSS



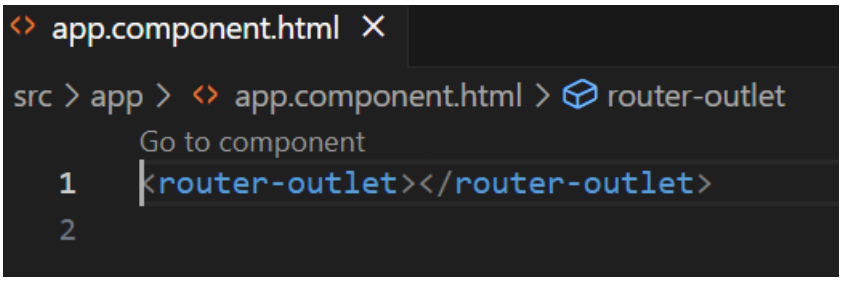
Creamos el componente “todo”



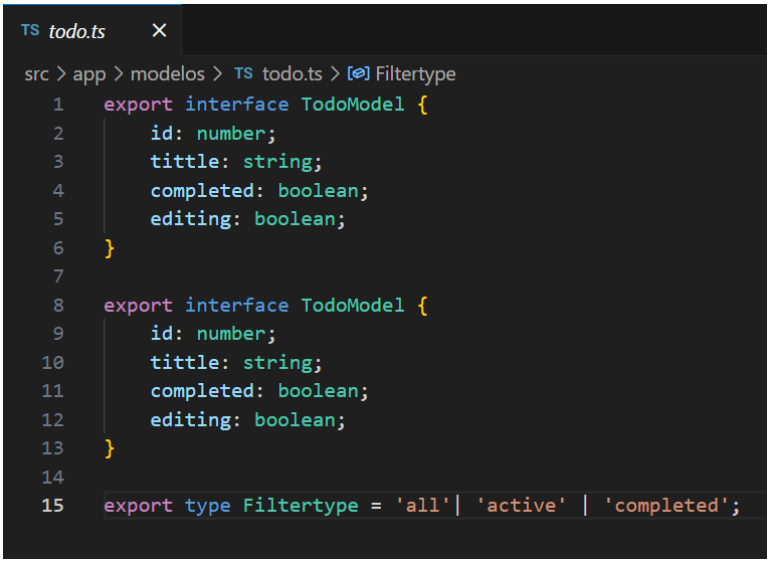
En “app.rotes.ts” ponemos la ruta de todo, aquí se va a ver nuestro cuadro.



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”

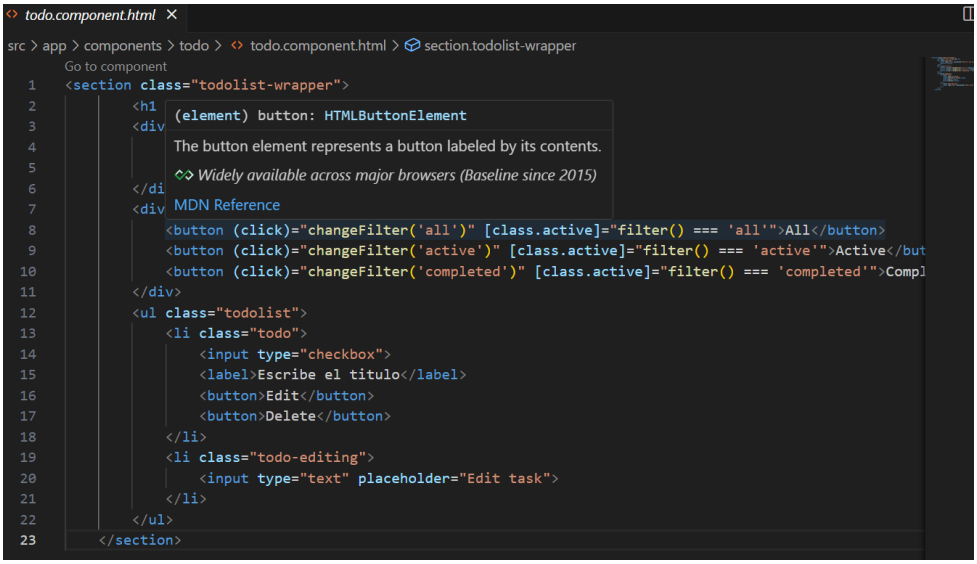


Creamos el modelo de “todo” para generar los campos que ocuparemos.

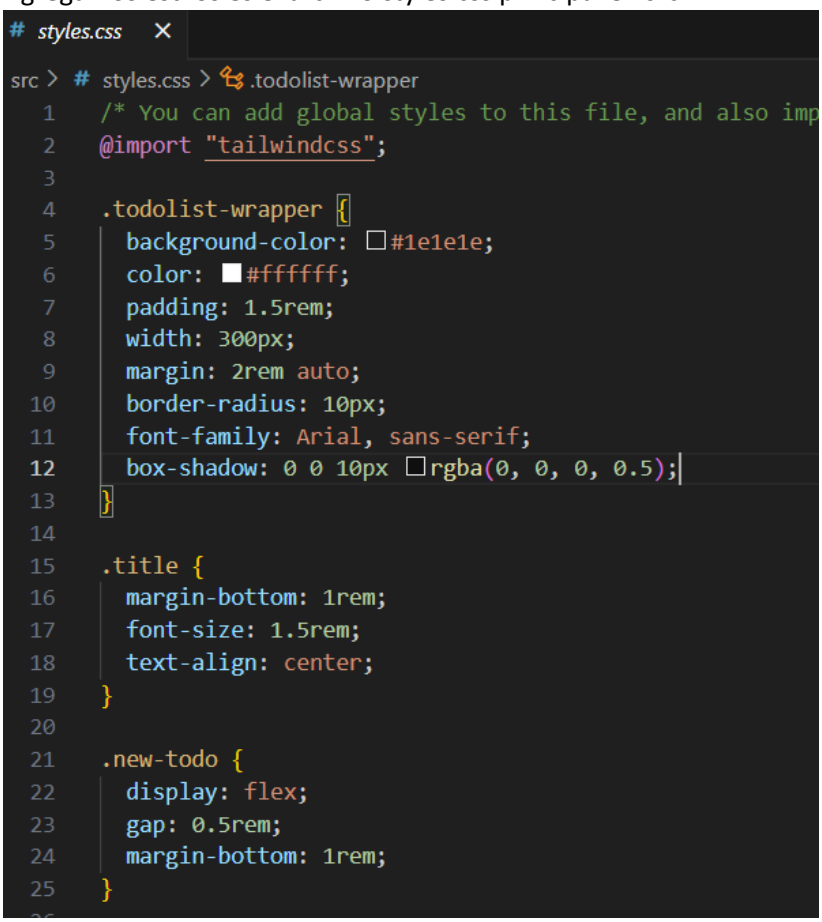


En el todo.component.ts ponemos el siguiente codigo dentro de un <section></section> tenemos un sección donde vamos a poder crear una nuva tarea y filtras las tareas que tengamos. Ocupamos el para enlistar los datos que tenemos dentro del check box.

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



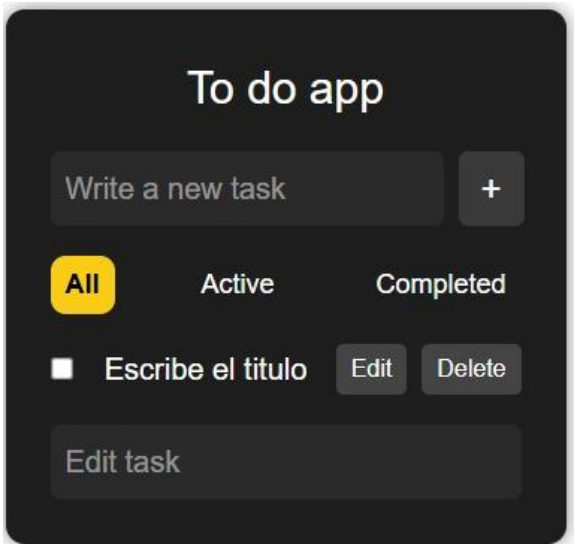
Agregamos estilos es el archivo styles.css principal en src



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

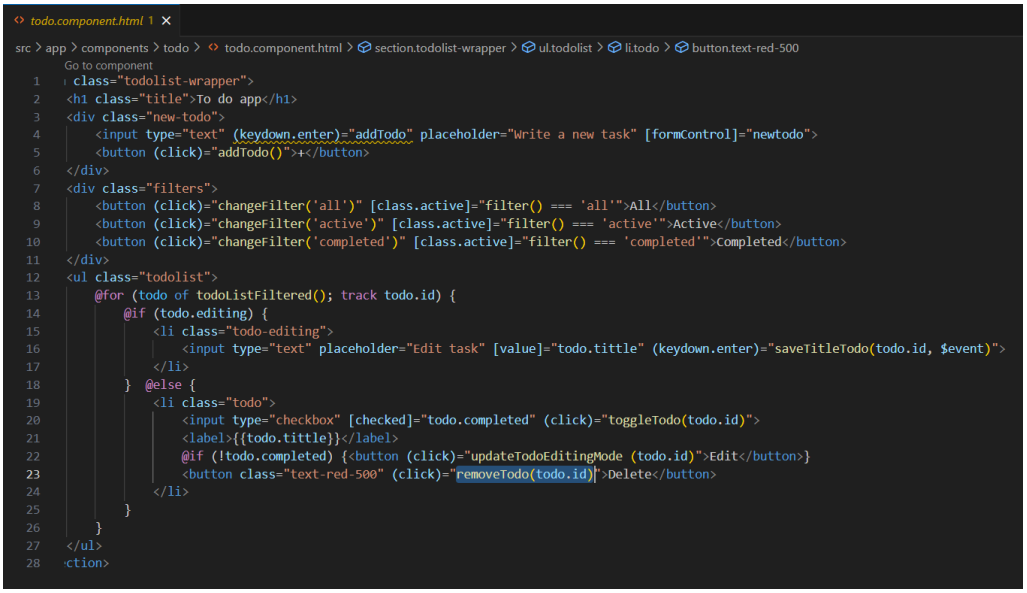
```ts
export class TodoComponent {

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

  filter = signal<Filtertype>('all');

  changeFilter(filterString: Filtertype) {
    this.filter.set(filterString);
  }
```

Así queda la pirmera parte de la practica

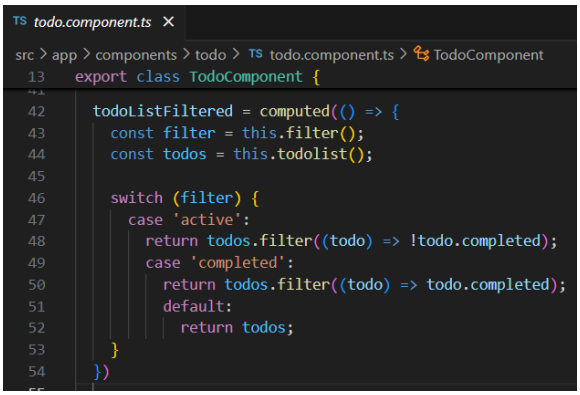


Segunda parte de la practica

Agregamos lo que falta al “todo.component.html” para que tenga todas las funcioalidades 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.



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.



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

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 ×
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(), tittle: 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 ×
src > app > components > todo > TS todo.component.ts > TodoComponer
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.

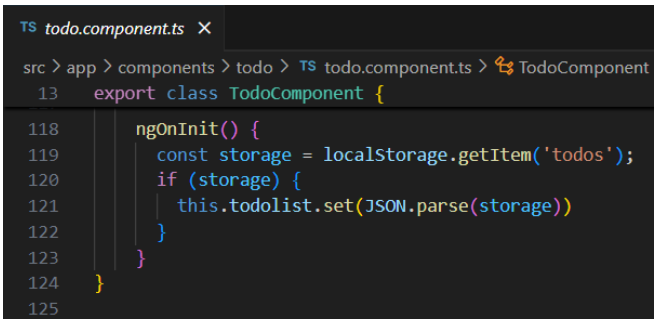
```
TS todo.component.ts ×
src > app > components > todo > TS todo.component.ts > TodoComponent > toggleTodo
13   export class TodoComponent {
90       removeTodo (todoiD: number) {
91         this.todolist.update((prev_todos) => prev_todos.filter((todo) => todo.id !==todoiD))
92       }
```

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

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

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.

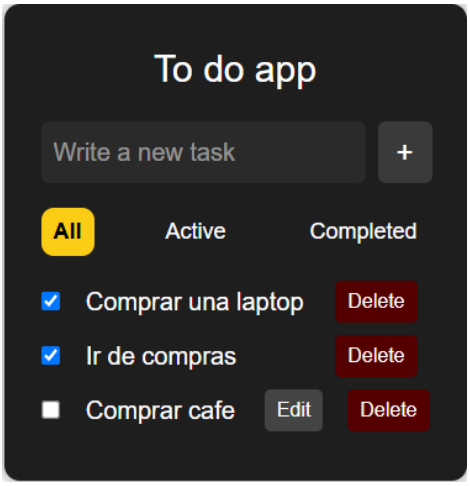
Agregamos “ngOnInit()” revisa si ya existen tareas guardadas y se muestran.

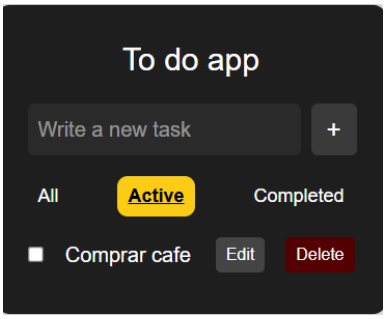


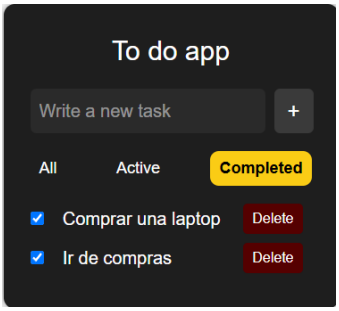
Y así queda

Tengo dos tareas completas y una incompleta

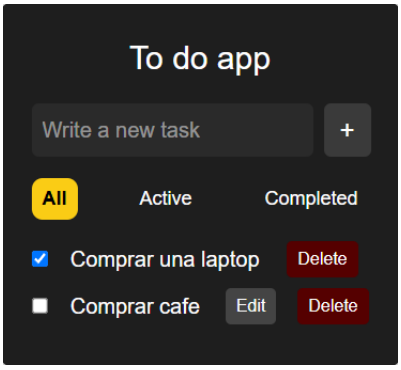
```
todolist = signal<TodoModel[]>([
  {
    id: 1,
    tittle: 'Comprar una laptop',
    completed: true,
    editing: false
  },
  {
    id: 2,
    tittle: 'Ir de compras',
    completed: true,
    editing: false
  },
  {
    id: 3,
    tittle: 'Comprar cafe',
    completed: false,
    editing: false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



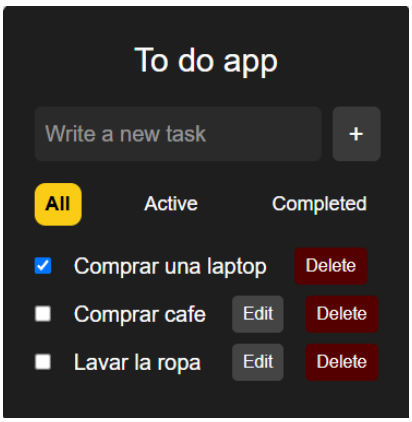




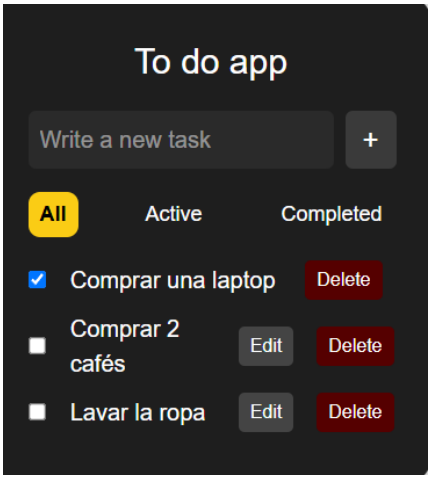
Eliminamos la tarea “ir de compras”



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 esten. También se configuró el almacenamiento local para que las tareas no se pierdan al recargar la página. Obtuvimos una aplicación funcionañ. Esta experiencia fue útil para reforzar conocimientos sobre componentes, eventos, formularios y estilos con Tailwind en proyectos con Angular.