14 - Formularios reactivos : ReactiveFormsModule, FormControl

Hemos dicho que Angular proporciona dos enfoques diferentes para manejar formularios:

1. Basado en plantillas (visto en el concepto anterior)
2. Reactivos

Ambos capturan los eventos de entrada del usuario desde la vista, crean un modelo de formulario y un modelo de datos para actualizar y proporcionan una forma de rastrear los cambios. Hasta ahora nosotros hemos trabajado con el modelo basado en plantillas.

El enfoque reactivo es más robusto, son más escalables, reutilizables y comprobables. Si los formularios son una parte clave de su aplicación, conviene utilizar el modelo reactivo.

Diferencias entre ambos modelos

- La configuración con el modelo de plantilla se hace mediante directivas y es menos explícito que el modelo reactivo que debe configurarse en la clase mediante código.
- En el formulario basado en plantillas el modelo de datos es más desestructurado.
- En el formulario basado en plantilla la validación se hace mediante directivas, en cambio en el modelo reactivo se hace mediante funciones.
- Las pruebas unitarias son más fácil de implementar en los formularios reactivos.
- Maneja cualquier escenario complejo es más fácil implementarlo con formularios reactivos.

Para trabajar con formularios reactivos necesitamos importar la clase 'ReactiveFormsModule' y crear un objeto de la clase FormControl por cada control visual que contendrá nuestro formulario.

Nuestro primer problema con formularios reactivos será el más elemental, ya que contendrá un solo control 'input' para el ingreso de una cadena.

Problema

Confeccionar una aplicación que permita ingresar actividades pendientes mediante un control 'input' de tipo 'text'.

Se deben listar todas las actividades ingresadas hasta el momento, además de poder borrar una en particular o todas a la vez.

Almacenar en forma local en el navegador las actividades mediante el API localStorage para evitar que se pierdan las actividades cuando se cierre el navegador.

- Crearemos primero el proyecto:

```
ng new proyecto009
```

- Trabajaremos con la componente que generar automáticamente Angular CLI cuando creamos el proyecto, pasamos a modificar la vista (app.component.html):

```
<p> Actividad:
  <input type="text" [formControl]="actividad">
</p>
<p><button (click)="agregar()">Agregar</button></p>
<ol>
  @for(acti of lista;track $index) {
  <li>
    {{acti}} <a (click)="borrar($index)" href="#">Borra?</a>
  </li>
  }
</ol>
<p><button (click)="borrarTodas()">Borrar todas las actividades</b
utton></p>

<router-outlet />
```

Analizaremos este archivo en conjunto luego de presentar 'app.component.ts'

- La clase asociada a la vista 'app.component.html' es el archivo 'app.component.ts' donde implementamos la siguiente lógica:

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { ReactiveFormsModule, FormControl } from '@angular/forms';

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

  actividad = new FormControl();

  lista: string[];

  constructor() {
    this.lista = [];
    let datos = localStorage.getItem("actividades");
    if (datos != null) {
      let arreglo = JSON.parse(datos);
      if (arreglo != null)
        for (let actividad of arreglo)
          this.lista.push(actividad);
    }
  }
```

```
  agregar() {
    this.lista.push(this.actividad.value);
    localStorage.setItem('actividades', JSON.stringify(this.lista)
);
    this.actividad.setValue('');
  }

  borrar(pos: number) {
    this.lista.splice(pos, 1);
    localStorage.clear();
    localStorage.setItem('actividades', JSON.stringify(this.lista)
);
  }

  borrarTodas() {
    localStorage.clear();
    this.lista = [];
  }
}
```

Lo primero que hacemos es importar la clase 'ReactiveFormsModule' y la interface 'FormControl':

```
import { ReactiveFormsModule, FormControl } from '@angular/forms';
```

Debemos importar en la componente la clase 'ReactiveFormsModule':

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

Dentro de la clase 'AppComponent' creamos un objeto de tipo 'FormControl' llamado 'actividad' (uno solo ya que solo tenemos un control de entrada de datos):

```
  actividad = new FormControl();
```

Este atributo 'actividad' se asocia con la vista mediante la sintaxis:

```
  <input type="text" [formControl]="actividad">
```

La cadena que carga el operador en el control visual luego se la puede acceder mediante la propiedad 'value' y si queremos cambiar su contenido disponemos del método 'setValue'.

El problema requiere que se almacenen todas las actividades que ingresa el usuario por teclado, esto sucede en el arreglo 'lista'. La definimos dentro de la clase:

```
  lista: string[];
```

Cuando el operador presiona el botón de 'Agregar' se llama el método 'agregar' y procedemos a recuperar el dato del control visual mediante la propiedad value y la agregamos al arreglo:

```
    this.lista.push(this.actividad.value);
```

Inmediatamente procedemos a borrar el contenido del control visual llamando al método 'setValue':

```
    this.actividad.setValue('');
```

Nuestro problema requiere almacenar cada una de las actividades en un arreglo que se pasa a mostrar en la vista mediante la estructura @for:

```
<ol>
  @for(acti of lista;track $index) {
  <li>
    {{acti}} <a (click)="borrar($index)" href="#">Borra?</a>
  </li>
  }
</ol>
```

Debemos especificar la variable especial $index que se numera de 0 en adelante. Dicho valor se pasa al método borrar: (click)="borrar($index)"

- Vimos hasta ahora todo lo relacionado con el control FormControl, pero nuestro problema requiere que la lista de actividades no se borre al cerrar el navegador, para ello utilizamos el API localStorage que proporciona HTML5.

  Cuando se presiona el botón 'agregar', luego de guardar la actividad del formulario procedemos a almacenar mediante el método 'setItem' del objeto 'localStorage' los datos de la lista pero en formato JSON:

  ```
  agregar() {
    this.lista.push(this.actividad.value);
    localStorage.setItem('actividades', JSON.stringify(this.lista));
    this.actividad.setValue('');
  }
  ```

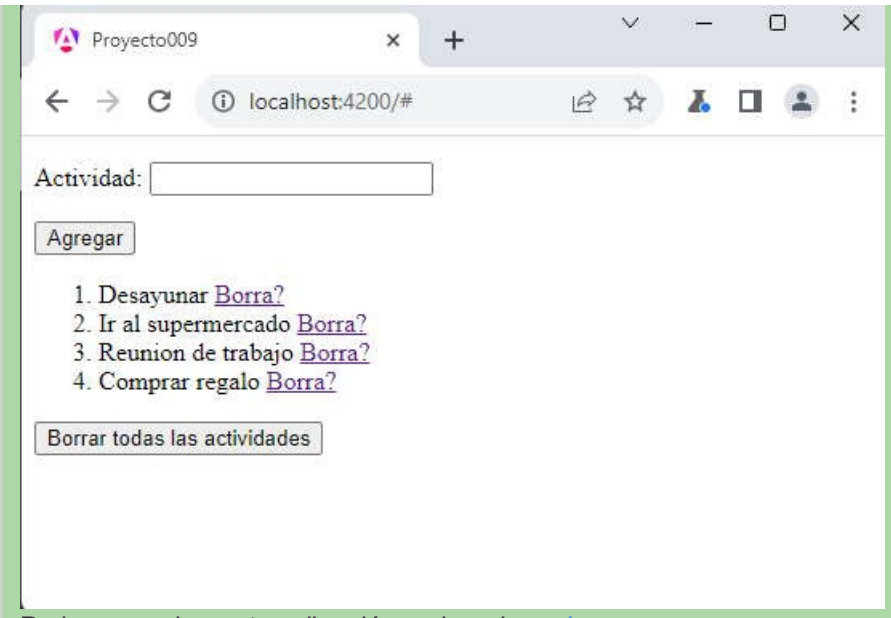
  Si se presiona el botón de borrar todas las actividades procedemos por un lado a borrar los elementos del arreglo y además a borrar los datos almacenados en 'localStorage':

  ```
  borrarTodas() {
    localStorage.clear();
    this.lista=[];
  }
  ```

  Otro punto importante es que cuando se ejecuta el método constructor al cargar la componente, verificamos si hay datos almacenados en el 'localStorage' para leerlos y cargar el arreglo 'lista' con los datos previos de ejecuciones anteriores de la aplicación Angular:

  ```
  constructor() {
    this.lista = [];
    let datos = localStorage.getItem("actividades");
    if (datos != null) {
      let arreglo = JSON.parse(datos);
      if (arreglo != null)
        for (let actividad of arreglo)
          this.lista.push(actividad);
    }
  }
  ```

Si ejecutamos la aplicación tenemos una interfaz similar a:



Podemos probar esta aplicación en la web aquí.

Retornar