Práctica #5

Objetivos

- Definir un template driven form para editar los datos del alumno
- Agregar la función de cancelar la edición y no propagar los cambios hasta confirmar los mismos
- Agregar validaciones a nuestro formulario
- Utilizar Angular Material para capturar la edición, mostrar ayudas y errores de validación.

1) Agregar los módulos faltantes a la aplicación

Para poder utilizar la infraestructura de **template driven forms** de Angular es necesario utilizar el módulo **FormsModule** e importarlo a nuestro módulo. Esto ya lo hemos realizado en una práctica anterior. Verifique que esto sea así en nuestro **app.modules.ts**.

- Verifique que existe el **import** de **FormModule** así como que esté en el array de **imports** en el decorador del módulo.

También, en las prácticas anteriores hemos agregado los módulos de **MatInputModule y MatFormFieldModule** que ahora utilizaremos más extensamente, pero nos falta agregar los módulos de Angular Material para otros controles:

- Agregue estos 4 **imports** adicionales:

```
import { MatSelectModule } from '@angular/material/select';
import { MatCheckboxModule } from '@angular/material/checkbox';
import { MatRadioModule } from '@angular/material/radio';
import { MatButtonModule } from '@angular/material/button';
```

De esta forma podremos utilizar los controles **mat-select, mat-radio, mat-checkbox y mat-button**

- Agregue estos cuatro módulos en el array de imports del módulo.
- Guarde los cambios

3) Utilizar el mat-radio-button para seleccionar el sexo en la edición de un alumno

- Edite el archivo **alumno-edicion.component.html**
- Reemplace esta línea de html

```
Sexo: {{itemList.Sexos(alumnoSeleccionado.sexo)}} <br />
```

Por

```
  Sexo

  <mat-radio-group [(ngModel)] = "alumnoSeleccionado.sexo" name="sexo">
    <mat-radio-button class="all-width"
      *ngFor="let sexo of sexos"
      [value] = "sexo.index">
      {{sexo.descripcion}}<br />
    </mat-radio-button>
  </mat-radio-group>

  <br /><br />
```

Este código html agregado hace lo siguiente:

- Utiliza el **mat-radio-group** para agrupar todos los controles de radio buttons.
- A este **mat-radio-group** lo vinculamos con el ngModel
- Por cada opción en el array de sexos generamos un control **mat-radio-button**
- A cada control le asignamos el valor y la descripción

Pruebe la aplicación. Verá ahora los radio button reflejando las opciones de sexo. Vamos ahora a darle un mejor formato

- Edite el archivo **alumno-edicion.component.css** y agregue la siguiente clase CSS

```
.all-width {
  width: 100%;
  margin-bottom: 0.5em;
}
```

- En el template HTML, antes del cierre de **</mat-card>** , agregue el siguiente código:

```
  <br /><br />
  {{ alumnoSeleccionado | json }}
</mat-card>
```

- Este código nos mostrará en formato json todo el objeto alumnoSeleccionado utilizando el **pipe json**. Los pipes son funciones que modifican la salida. Esto nos servirá de debug.
- Pruebe la aplicación. Vea que pasa si borra el pipe.

4) Utilizar el mat-select para seleccionar el perfil en la edición de un alumno

- Reemplace esta línea de html

```
Perfil: {{itemList.Perfiles(alumnoSeleccionado.perfil)}} <br>
```

- por el siguiente código html justo debajo de los radio buttons

```
  <mat-form-field  class="all-width">
    <mat-select placeholder="Perfil"
[(ngModel)]="alumnoSeleccionado.perfil" name="perfil">
        <mat-option *ngFor="let perfil of perfiles"
[value]="perfil.index">
          {{ perfil.descripcion }}
        </mat-option>
      </mat-select>
  </mat-form-field>

  <br/><br/>
```

- Vea cómo está conformado el select.
- Pruebe la aplicación

5) Utilizar el mat-checkbox para activar o desactivar un alumno

- Reemplace esta línea de html

```
Activo: {{alumnoSeleccionado.activo}}
```

- Por el siguiente código

```
<mat-checkbox [(ngModel)]="alumnoSeleccionado.activo" name="activo">
  Activo
</mat-checkbox>
```

- Guarde los cambios y pruebe la aplicación
- Vea de agregar los estilos y clases css necesarios para emprolijar la edición

6) Agregar el formulario y validaciones

En esta parte cambiaremos el formulario utilizando binding de 2 vías con el modelo que se encuentra en la propiedad **AlumnoSeleccionado**.

- Necesitamos un elemento **form** relacionado con un ngForm, para ello agregue la siguiente línea luego del **<h2>Alumno Seleccionado</h2>**

```
<form novalidate #f="ngForm">
```

Con esto declaramos una variable local del template llamada **f**.

- Agregue las validaciones **required** a la edición del nombre y del apellido

```
    <mat-form-field class="all-width">
        <input matInput required name="nombre" placeholder="Nombre"
[(ngModel)]="alumnoSeleccionado.nombre">
    </mat-form-field>
    <mat-form-field  class="all-width">
      <input matInput required name="apellido" placeholder="Apellido"
[(ngModel)]="alumnoSeleccionado.apellido">
    </mat-form-field>
```

Utilizaremos la propiedad **name** para vincular el valor del control con el modelo. Esto es un requisito para poder utilizar **[(ngModel)]**. Además agregaremos la validación control obligatorio (**required**).

Agregaremos ahora los botones de aceptar y cancelar. El botón de aceptar deberá estar deshabilitado cuando los campos sean inválidos. Agregue el siguiente código antes del cierre **</mat-card>**

```
    <mat-card-actions>
      <button mat-button
          (click) = "Regresar()">
        Cancelar
      </button>
      <button mat-button color="primary"
          [disabled]="f.invalid"
          (click) = "Guardar()">
        Aceptar
      </button>
    </mat-card-actions>

    </form>
```

Utilizando la variable del template **f**, asociada al formulario vía el **ngForm**, podemos saber si el estado es inválido o no y asignar esto a la propiedad **disabled** del control.

- Para ver que es lo que está pasando mientras editamos agregaremos el siguiente código para mostrarnos los objetos **alumnoEdit** y **ngForm**. Agréguelos al final del código:

```
  <br /><br />
  {{f.value | json }}<br />
  {{ alumnoSeleccionado | json }}
```

- Ahora agregaremos las dos funciones que utilizamos en los botones: **Regresar() y Guardar()**. Agregue el siguiente código en el archivo **alumno-edicion.component.ts**

```
  Regresar() {
    this.alumnoSeleccionado = null;
  }

  Guardar() {

  }
}
```

Por ahora la función **Guardar**() no hace nada. Pruebe ahora la aplicación y vea que al borrar todo el nombre de un alumno el botón de Aceptar queda deshabilitado. También vea que por el efecto del 2 way data binding los cambios se propagan por más que presionemos el botón de cancelar. Este no es el efecto deseado y lo corregiremos en los siguientes pasos.

7) Mostrar los mensajes de errores de validación utilizando el mat-form-field

Edite el código html de la edición del nombre y apellido para agregar las validaciones.

- Agregue lo marcado en amarillo

```
<mat-form-field class="all-width">
    <input matInput required name="nombre"
           placeholder="Nombre"
           [(ngModel)]="alumnoSeleccionado.nombre"
           #nombre="ngModel" />
    <mat-error *ngIf="nombre.invalid">Debe ingresar el nombre
    </mat-error>
</mat-form-field>
<mat-form-field  class="all-width">
  <input matInput required name=" apellido"
         placeholder="Apellido"
         [(ngModel)]="alumnoSeleccionado.apellido"
         #apellido="ngModel" />
  <mat-error *ngIf="apellido.invalid">Debe ingresar el apellido
  </mat-error>
</mat-form-field>
```

Vea como hemos definido dos variables locales al template y luego las utilizamos para checkear si los controles están en estado válido o no.

Pruebe ahora la edición generando errores de validación.

8) Desacoplar el formulario del modelo para no propagar los cambios

Vamos ahora a solucionar el problema de la propagación de cambios que genera el 2 way databinding. Hay varias estrategias para esto y la que usaremos es la de desacoplar el modelo de el formulario. Para ello:

Edite el archivo alumno-edicion.component.html y reemplace todos los **[(ngModel)]** por **[ngModel]**

Pruebe ahora la aplicación y compruebe que los cambios han dejado de propagarse. Vea también las diferencias en la información que mostramos para debug entre **alumnoEdicion y f**

Agregaremos ahora la funcionalidad de guardar los cambios. Vamos a pasar en forma manual los valores del formulario a la función **Guardar()** y en su cuerpo moveremos los valores a alumnoEdición.

- Edite el código html y agregue en la invocación de la función Guardar() lo que está resaltado.

```
    <button mat-button color="primary"
        [disabled]="f.invalid"
        (click) = "Guardar(f.value)">
      Aceptar
    </button>
```

- Edite el código de **alumno-edicion.component.ts** y modifique la función Guardar() de esta forma:

```
Guardar(form: any) {
  this.alumnoSeleccionado.nombre = form.nombre;
  this.alumnoSeleccionado.apellido = form.apellido;
  this.alumnoSeleccionado.sexo = form.sexo;
  this.alumnoSeleccionado.perfil = form.perfil;
  this.alumnoSeleccionado.activo = form.activo;

  this.Regresar();
}
```

Pruebe ahora la aplicación. Verá que la misma funciona casi perfectamente. Nos queda un problema y es: ¿qué pasa si volvemos a seleccionar el mismo alumno luego de presionar cancelar o aceptar? En el siguiente punto resolveremos esto.

Otro tema adicional es el siguiente: nuestra clase alumno sólo tiene 5 propiedades, si tuviera 30 el copiado manual sería no solamente tedioso sino también propenso a errores. La solución a esto está en la posibilidad que nos da Javascript de recorrer las propiedades utilizando la funcionalidad de **Object.keys**. Para probar esta funcionalidad cambie la función guardar de la siguiente forma

```
Guardar(form: any) {
  // this.alumnoSeleccionado.nombre = form.nombre;
  // this.alumnoSeleccionado.apellido = form.apellido;
  // this.alumnoSeleccionado.sexo = form.sexo;
  // this.alumnoSeleccionado.perfil = form.perfil;
  // this.alumnoSeleccionado.activo = form.activo;
```

```
  Object.keys(form).forEach((key, index) =>
   this.alumnoSeleccionado[key] = form[key]
  );

  this.Regresar();
}
```

- Guarde los cambios y pruebe ahora la aplicación.

9) Deselecionar el alumno de la lista cuando se acepta o cancela la edición

Para evitar el bug descrito en el punto anterior, haremos lo siguiente:

1. Exponer la propiedad **alumnoSeleccionado** como **@Input** en el componente **alumnos-lista**
2. Crear un evento llamado FinDeEdicion en **alumno-edicion** que se dispare cuando presionamos los botones Aceptar o Cancelar
3. Atrapar este evento en el componente padre (**app.component**) y modificar con un null la nueva propiedad de Input creada en el punto 1

Los invitamos a que desarrollen estos puntos.