Laboratorio Angular:
Sistema de Notificaciones

En una aplicación es común realizar notificaciones al usuario que se le deben mostrar, la forma de presentar las notificaciones puede variar de una aplicación a otra, incluso dentro de la misma aplicación, pero la necesidad persiste.

Vamos a crear un servicio que será el encargado de la gestión de las notificaciones (ViewModel) y será inyectado a cualquier otro artefacto que requiera realizar notificaciones al usuario. Para mejorar su reutilización se creará como contenedor de múltiples notificaciones y que avise cuando reciba una nueva notificación. El servicio se apoyará en un modelo, con la estructura de la notificación, y en una enumeración, que fijará los tipos de notificación.

Para crear la vista, presentación de las notificaciones, crearemos un componente que permitirá la interacción del usuario con múltiples notificaciones.

Siguiendo los principios de modularidad, crearemos un módulo con los servicios generales de la aplicación (CommonServicesModule) y otro módulo para la capa principal de presentación de la aplicación (MainModule).

Módulo CommonServicesModule:

Crear la carpeta del módulo de CommonServicesModule

- md src\app\common-services

Crear el servicio de notificaciones

- ng generate service common-services/Notification

Editar el fichero src/app/common-services/notification.service.ts

Añadir, después de los imports, la enumeración de tipo de notificación

```
export enum NotificationType { error = 'error', warn = 'warn', info = 'info', log
  = 'log' }
```

A continuación, el modelo de la notificación:

```
export class Notification {
        constructor(private id: number, private message: string,
                    private type: NotificationType) {}
        public get Id() { return this.id; }
        public get Message() { return this.message; }
        public get Type() { return this.type; }
      }
```


Completar clase NotificationService.

```
@Injectable({ providedIn: 'root'})
export class NotificationService {
```

Exponer la enumeración como atributo de solo lectura para evitar problemas con las plantillas

```
  public readonly NotificationType = NotificationType;
```

Añadir, como atributo, la colección de notificaciones

```
private listado: Notification[] = [];
```

Inyectar en el constructor el LoggerService para realizar las notificaciones de depuración (realizar el import correspondiente)

```
  constructor(private out: LoggerService) { }
```

Exponer como propiedades de solo lectura los elementos vinculables para realizar el interfaz de usuario (para no romper la encapsulación es necesario clonar las referencias):

```
  public get Listado(): Notification[]
    { return Object.assign([], this.listado); }
  public get HayNotificaciones() { return this.listado.length > 0; }
```

Crear el método que permite añadir nuevas notificaciones:

```
  public add(msg: string, type: NotificationType = NotificationType.error) {
    if (!msg || msg === '') {
      this.out.error('Falta el mensaje de notificación.');
      return;
    }
    const id = this.HayNotificaciones ?
      (this.listado[this.listado.length - 1].Id + 1) : 1;
    const n = new Notification(id, msg, type);
    this.listado.push(n);
    // Redundancia: Los errores también se muestran en consola
    if (type === NotificationType.error) {
      this.out.error(`NOTIFICATION: ${msg}`);
    }
  }
```

Crear el método que permite eliminar una notificación indicando su posición en la colección:

```
  public remove(index: number) {
    if (index < 0 || index >= this.listado.length) {
      this.out.error('Index out of range.');
      return;
    }
    this.listado.splice(index, 1);
  }
```

Borrar todas las notificaciones:

```
  public clear() {
    if (this.HayNotificaciones)
      this.listado.splice(0);
  }
```

Guardar el fichero src/app/common-services/notification.service.ts

Crear el fichero src/app/common-services/index.ts del módulo

Editar el fichero src/app/common-services/index.ts y exportar la clase del módulo y de las notificaciones:

```
export * from './notification.service';
```

Módulo MainModule:

Crear el módulo de MainModule

- md src\app\main

Crear el componente de notificaciones dentro del módulo:

- ng generate component main/notification

Editar el fichero src/app/main/notification/notification.component.ts con la clase del componente.

Realizar las importaciones de las declaraciones utilizadas en la plantilla:

```
@Component({
    :
    imports: [NgIf, NgFor, I18nSelectPipe]
})
export class NotificationComponent {
```

Inyectar en el constructor el NotificationService que actuará como ViewModel de la vista (realizar el import correspondiente)

```
  constructor(private vm: NotificationService) { }
```

Exponer el ViewModel, como propiedad de solo lectura, para permitir la vinculación desde la plantilla:

```
public get VM() { return this.vm; }
```

Editar el fichero src/app/main/notification/notification.component.html, con la plantilla del componente, y sustituir el código por defecto por:

```
@if(VM.HayNotificaciones) {
  <div class="notificaciones">
    @for(item of VM.Listado; track item.Id) {
      <div class="alert {{item.Type | i18nSelect:{
          'error':'alert-danger',
          'warn':'alert-warning', 'info':'alert-primary',
          'other':'alert-success' } }} alert-dismissible fade show"
          role="alert">
        {{item.Message}}
        <button type="button" class="btn-close" data-bs-dismiss="alert"
          aria-label="Close" (click)="VM.remove($index)"></button>
      </div>
    }
    <div class=" text-center">
      <button class="btn btn-info" type="button"
        (click)="VM.clear()">Cerrar</button>
    </div>
  </div>
}
```

Incorporar al componente principal el nuevo componente creado. Editar el fichero src/app/app.component.ts y añadir las importaciones de los componentes (realizar el import de la clase correspondiente):

```
@Component({
  :
  imports: [CommonModule, RouterOutlet, NotificationComponent,],
```

Editar el fichero src/app/app.component.html y añadir al principio:

```
<app-notification />
```

Crear el fichero src/app/main/index.ts del módulo

Editar el fichero y exportar la clase del componente:

```
export * from './notification/notification.component'
```

Verificación

Para probar el nuevo sistema de notificaciones vamos a crear un componente de Demos que posteriormente eliminaremos.

Crear el componente de notificaciones dentro del módulo:

- ng generate component demos

Editar el fichero src/app/demos/demos.component.ts para inyectar en el constructor el NotificationService y tener acceso al sistema de notificaciones (realizar el import correspondiente)

```
  constructor(public vm: NotificationService) { }
```

Editar el fichero src/app/demos/demos.component.html, con la plantilla del componente, y sustituir el código por defecto por:

```
<p>
  <input type="button" value="Error" (click)="vm.add('Esto es una notificación de error')" >
  <input type="button" value="Warn" (click)="vm.add('Esta notificación es un aviso',
vm.NotificationType.warn)" >
  <input type="button" value="Info" (click)="vm.add('Solo una notificación informativa',
vm.NotificationType.info)" >
  <input type="button" value="Log" (click)="vm.add('Para trazar con notificaciones',
vm.NotificationType.log)" >
</p>
```

Incorporar al componente principal el nuevo componente creado. Editar el fichero src/app/app.component.ts y añadir las importaciones de los componentes:

```
@Component({
  :
  imports: [CommonModule, RouterOutlet, NotificationComponent, DemosComponent, ],
```

Editar el fichero src/app/app.component.html y añadir al principio, después del componente de notificaciones:

```
<app-notification />
<app-demos />
```

Ejecutar y probar. Abrir el inspector de código para verificar las salidas a consola.

Versión modal del componente (Ampliación)

Crear el componente de notificaciones modales dentro del módulo:

- ng generate component main/notification-modal

Editar el fichero src/app/main/notification/notification-modal.component.ts con la clase del componente.

Realizar las importaciones de las declaraciones utilizadas en la plantilla:

```
@Component({
```

```
    :
    imports: [NgIf, NgClass, NgFor, NgSwitch, NgSwitchCase, NgSwitchDefault]
})
export class NotificationModalComponent {
```

Inyectar en el constructor el NotificationService que actuará como ViewModel de la vista (realizar el import correspondiente)

```
  constructor(private vm: NotificationService) { }
```

Exponer el ViewModel, como propiedad de solo lectura, para permitir la vinculación desde la plantilla:

```
public get VM() { return this.vm; }
```

Editar el fichero src/app/main/notification/notification-modal.component.html, con la plantilla del componente, y sustituir el código por defecto por:

```
<style>
  .fondo-sombreado {
    position: fixed;
    background-color: black;
    left: 0;
    top: 0;
    width: 100%;
    height: 100%;
    opacity: 0.3;
  }
</style>
@if(VM.HayNotificaciones){
  <div class="fondo-sombreado"></div>
  <div class="modal fade show" [style.display]="'block'" tabindex="-1"
    [attr.aria-hidden]="!VM.HayNotificaciones">
    <div class="modal-dialog">
      <div class="modal-content">
        <div class="modal-header bg-gradient text-white"
          [ngClass]="{
              'bg-danger': VM.Listado[0].Type === VM.NotificationType.error,
              'bg-warning': VM.Listado[0].Type === VM.NotificationType.warn,
              'bg-primary': VM.Listado[0].Type === VM.NotificationType.info,
              'bg-success': VM.Listado[0].Type === VM.NotificationType.log
            }">
          <h5 class="modal-title" id="exampleModalLabel">Notificaciones</h5>
          <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"
            (click)="VM.clear()"></button>
        </div>
        <div class="modal-body bg-light bg-gradient p-0">
          <svg xmlns="http://www.w3.org/2000/svg" style="display: none;">
            <symbol id="check-circle-fill" fill="currentColor" viewBox="0 0 16 16">
              <path
```

```
              d="M16 8A8 8 0 1 1 0 8a8 8 0 0 1 16 0zm-3.97-3.03a.75.75 0 0 0-1.08.022L7.477
9.417 5.384 7.323a.75.75 0 0 0-1.06 1.06L6.97 11.03a.75.75 0 0 0 1.079-.02l3.992-4.99a.75.75 0
0 0-.01-1.05z" />
            </symbol>
            <symbol id="info-fill" fill="currentColor" viewBox="0 0 16 16">
              <path
                d="M8 16A8 8 0 1 0 8 0a8 8 0 0 0 0 16zm.93-9.412-1 4.705c-
.07.34.029.533.304.533.194 0 .487-.07.686-.246l-.088.416c-.287.346-.92.598-1.465.598-.703 0-
1.002-.422-.808-1.319l.738-3.468c.064-.293.006-.399-.287-.47l-.451-.081.082-.381 2.29-.287zM8
5.5a1 1 0 1 1 0-2 1 1 0 0 1 0 2z" />
            </symbol>
            <symbol id="exclamation-triangle-fill" fill="currentColor" viewBox="0 0 16 16">
              <path
                d="M8.982 1.566a1.13 1.13 0 0 0-1.96 0L.165 13.233c-.457.778.091 1.767.98
1.767h13.713c.889 0 1.438-.99.98-1.767L8.982 1.566zM8 5c.535 0 .954.462.9.995l-.35
3.507a.552.552 0 0 1-1.1 0L7.1 5.995A.905.905 0 0 1 8 5zm.002 6a1 1 0 1 1 0 2 1 1 0 0 1 0-2z"
/>
            </symbol>
            <symbol id="forbiben-circle-fill" fill="currentColor" viewBox="0 0 16 16">
              <path d="M16 8A8 8 0 1 1 0 8a8 8 0 0 1 16 0zM4.5 7.5a.5.5 0 0 0 0 1h7a.5.5 0 0 0
0-1h-7z" />
            </symbol>
          </svg>
          @for(item of VM.Listado; track item.Id) {
            @switch (item.Type) {
              @case (VM.NotificationType.error) {
              <div class="m-0 p-3 alert alert-danger d-flex align-items-center rounded-0"
                role="alert">
                <svg class="bi flex-shrink-0 me-2 text-danger" width="24" height="24"
role="img" aria-label="Danger:">
                  <use xlink:href="#forbiben-circle-fill" />
                </svg>
                <div>{{item.Message}}</div>
              </div>
              }
              @case (VM.NotificationType.warn) {
              <div class="m-0 p-3 alert alert-warning d-flex align-items-center rounded-0"
                role="alert">
                <svg class="bi flex-shrink-0 me-2 text-warning" width="24" height="24"
role="img" aria-label="Danger:">
                  <use xlink:href="#exclamation-triangle-fill" />
                </svg>
                <div>{{item.Message}}</div>
              </div>
              }
              @case (VM.NotificationType.info) {
              <div class="m-0 p-3 alert alert-primary d-flex align-items-center rounded-0"
                role="alert">
```

```
                <svg class="bi flex-shrink-0 me-2 text-primary" width="24" height="24"
role="img" aria-label="Info:">
                  <use xlink:href="#info-fill" />
                </svg>
                <div>{{item.Message}}</div>
              </div>
              }
              @default {
              <div class="m-0 p-3 alert alert-success d-flex align-items-center rounded-0"
role="alert">
                <svg class="bi flex-shrink-0 me-2 text-success" width="24" height="24"
role="img" aria-label="Log:">
                  <use xlink:href="#check-circle-fill" />
                </svg>
                <div>{{item.Message}}</div>
              </div>
              }
            }
          }
        </div>
        <div class="modal-footer bg-light bg-gradient">
          <button type="button" class="btn btn-secondary" (click)="VM.clear()">Cerrar</button>
        </div>
      </div>
    </div>
  </div>
}
```

Editar el fichero src/app/main/index.ts y exportar la clase del componente:

```
export * from './notification-modal/notification-modal.component'
```

Sustituir en el componente principal anterior por el nuevo componente creado. Editar el fichero src/app/app.component.ts y añadir las importaciones de los componentes:

```
@Component({
  :
  imports: [CommonModule, RouterOutlet, NotificationModalComponent, DemosComponent],
```

Editar el fichero src/app/app.component.html y añadir el sufijo -modal:

```
<app-notification-modal />
```

Volver a realizar las pruebas.

Ampliación: Servicios reactivos

Convertir el servicio en observable

Editar el fichero src/app/common-services/notification.service.ts

Importar, de la biblioteca RxJS, el tipo base del observable:

```
import { Subject } from 'rxjs';
```

Completar clase NotificationService con la implementación de OnDestroy:

```
@Injectable({ providedIn: 'root'})
export class NotificationService implements OnDestroy {
```

Añadir, como atributo, el observable e inicializarlo (caliente):

```
private notificacion$ = new Subject<Notification>();
```

Exponer el observable, como propiedad de solo lectura, para que acepte suscriptores:

```
public get Notificacion() { return this.notificacion$; }
```

Modificar el método Add para que emita las notificaciones:

```
  public add(msg: string, type: NotificationType = NotificationType.error) {
    if (!msg || msg === '') {
      this.out.error('Falta el mensaje de notificación.');
      return;
    }
    const id = this.HayNotificaciones ?
      (this.listado[this.listado.length - 1].Id + 1) : 1;
    const n = new Notification(id, msg, type);
    this.listado.push(n);
     this.notificacion$.next(n);
    // Redundancia: Los errores también se muestran en consola
    if (type === NotificationType.error) {
      this.out.error(`NOTIFICATION: ${msg}`);
    }
  }
```

Notificar en el destructor que el observable se ha completado:

```
  ngOnDestroy(): void {
    this.notificacion$.complete()
  }
```

Crear un suscriptor

Editar el fichero src/app/demos/demos.component.ts.

Añadir OnDestroy a la lista de interfaces:

```
export class DemosComponent implements OnInit, OnDestroy {
```

Agregar un atributo que almacene al suscriptor para poder cancelar la suscripción al destruir el componente:

```
private suscriptor: Unsubscribable | undefined;
```

Al inicializar el componente, se crea la suscripción y se indica el tratamiento de las nuevas notificaciones:

```
  ngOnInit(): void {
    this.suscriptor = this.vm.Notificacion.subscribe(n => {
      if (n.Type !== NotificationType.error) { return; }
      window.alert(`Suscripción: ${n.Message}`);
      this.vm.remove(this.vm.Listado.length - 1);
    });
  }
```

Al destruir el componente se debe cancelar la suscripción para evitar fugas de memoria y de proceso:

```
  ngOnDestroy(): void {
    if (this.suscriptor) {
      this.suscriptor.unsubscribe();
    }
  }
```