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 la 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 el módulo de CommonServicesModule

- ng generate module CommonServices

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, warn, info, 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.

Añadir, como atributo, la colección de notificaciones

```
private listado: Array<Notification> = [];
```

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

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

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() { 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(`NOTICATION: ${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.service.ts

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

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

```
export { CommonServicesModule } from './common-services.module';
export * from './notification.service';
```

Importar el módulo recién creado en el módulo principal:

- Editar app.module.ts
- En la tabla de la propiedad imports de @NgModule añadir CommonServicesModule (realizar el import de la clase correspondiente)

Ampliación: 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.

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:

```
const n = new Notification(id, msg, type);
this.listado.push(n);
this.notificacion$.next(n);
```

Módulo MainModule:

Crear el módulo de MainModule

- ng generate module main

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

Editar el fichero y exportar la clase del módulo:

```
export { MainModule } from './main.module';
```

Importar el módulo recién creado en el módulo principal:

- Editar app.module.ts
- En la tabla de la propiedad imports de @NgModule añadir MainModule (realizar el import de la clase correspondiente)

Crear el componente de notificaciones dentro del módulo:

- ng generate component main/notification

Editar el fichero src/app/main/main.module.ts del módulo.

El generador a registrado automáticamente la declaración del componente, pero hace falta exportarlo para poder utilizarlo fuera del módulo:

```
@NgModule({
  declarations: [NotificationComponent],
  exports: [NotificationComponent],
  imports: [
    CommonModule
  ]
})
export class MainModule { }
```

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

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:

```
<div class="notificaciones" *ngIf="VM.HayNotificaciones">
  <div class="notificacion"
    [class.error]="item.Type === VM.NotificationType.error"
    *ngFor="let item of VM.Listado; let i=index">
    {{item.Message}}
    <button type="button" (click)="VM.remove(i)">X</button>
  </div>
```

```
    <div class="center">
      <button class="btn btn-
info" type="button" (click)="VM.clear()">Cerrar</button>
    </div>
</div>
```

Editar el fichero src/styles.css con estilo global de la aplicación y añadir:

```
.notificaciones .notificacion {
  border: solid 1px rgb(255, 208, 0);
  border-radius: 5px;
  color: #856404;
  background: #fff3cd;
  padding: 2pt 5pt 5pt 5pt;
  margin: 5pt;
  margin-top: 1pt;
  -webkit-box-shadow: 5px 7px 5px 0px rgba(136, 100, 4, 0.49);
  -moz-box-shadow: 5px 7px 5px 0px rgba(136, 100, 4, 0.49);
  box-shadow: 5px 7px 5px 0px rgba(136, 100, 4, 0.49);
}
.notificaciones .notificacion button {
  float: right;
  font-size: 0.8em;
  font-weight: bold;
  border-radius: 10px;
  background: #000000;
  color: white;
}
.notificaciones .center {
  text-align: center;
}
.notificaciones .error {
  color: white;
  background: #B00020;
  border-color: red;
  -webkit-box-shadow: 5px 7px 5px 0px rgba(255, 0, 0, 0.49);
  -moz-box-shadow: 5px 7px 5px 0px rgba(255, 0, 0, 0.49);
  box-shadow: 5px 7px 5px 0px rgba(255, 0, 0, 0.49);
}
```

Incorporar al componente principal el nuevo componente creado. Editar el fichero src/app/app.component.html y añadir al principio:

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

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)" >
</p>
```

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

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

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

Ampliación: 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(`Suscripcion: ${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();
  }
}
```