7-watch

Vigilancia y seguridad en Angular

1. Observables para monitorizar datos

2. Interceptores para gestionar errores

3. Un notificador de problemas

1. Observables para monitorizar datos

Productores de observables

Un Store de notificaciones

Desacoplados pero conectados

El módulo de notificaciones

```
ng g m notifications --routing true
ng g c notifications/sender
ng g c notifications/receiver
```

app-routing.module.ts

```
{
  path: 'notifications',
  loadChildren: './notifications/notifications.module#NotificationsModule'
},
```

notifications-routing.module.ts

```
const routes: Routes = [
  {
    path: 'sender',
    component: SenderComponent
```

```
  },
  {
    path: 'receiver',
    component: ReceiverComponent
  },
  {
    path: '**',
    redirectTo: 'sender'
  }
];
```

header.component.html

```
<a routerLink="notifications" class="button">Notifications</a>
```

1.1 Productores de observables

Of y from

```
value$ = of(new Date().getMilliseconds());
value$.subscribe(r=> console.log(r));
stream$ = from([1, 'two', '***']);
stream$.subscribe(r=> console.log(r));
list$ = of(['N', 'S', 'E', 'W']);
list$.subscribe(r=> console.log(r));
```

Subject y BehaviorSubject

```
const data = {name:'', value:0};

const need_sync$ = new Subject<any>();
// on time
need_sync.subscribe(r=> console.log(r));
need_sync.next(data);
// too late
need_sync.subscribe(r=> console.log(r));

const no_hurry$ = new BehaviorSubject<any>(this.data);
// its ok
no_hurry.subscribe(r=> console.log(r));
no_hurry.next(data);
// its also ok
no_hurry.subscribe(r=> console.log(r));
```

1.2 Un Store de notificaciones

```
ng g s notifications/notificationsStore
```

```
export class NotificationsStoreService {
  private notifications = [];
  private notifications$ = new BehaviorSubject<any[]>([]);

  constructor() {}

  public select$ = () => this.notifications$.asObservable();
  public dispatch(notification) {
    this.notifications.push(notification);
    this.notifications$.next([...this.notifications]);
  }
}
```

1.3 Desacoplados pero conectados

Emisión

Vista con un formulario para enviar mensajes

```
<h2>
  Notes sender
</h2>
<form>
  <fieldset>
    <section>
      <label for="note">Note</label>
      <input name="note"
             [(ngModel)]="note" />
    </section>
  </fieldset>
  <button (click)="send()">Send</button>
</form>
<a [routerLink]="['../receiver']">Go to receiver</a>
```

Dependencia y uso del servicio del almacén de notificaciones

```
export class SenderComponent implements OnInit {
  public note = '';
```

```
  constructor(private notificationsStore: NotificationsStoreService) {}

  ngOnInit() {}

  public send() {
    this.notificationsStore.dispatch(this.note);
  }
}
```

Recepción

Listado de notificaciones, no importa el orden de subscripción

```
<h2>
  Notes receiver
</h2>
<ul>
  <li *ngFor="let note of notes$ | async">{{ note | json }}</li>
</ul>
<a [routerLink]="['../sender']">Go to sender</a>
```

Dependencia y uso del servicio del almacén de notificaciones

```
export class ReceiverComponent implements OnInit {
  public notes$;

  constructor(private notificationsStore: NotificationsStoreService) {}

  ngOnInit() {
    this.notes$ = this.notificationsStore.select$();
  }
}
```

> Recap:

1. Observables para monitorizar datos

Productores de observables

Un Store de notificaciones

Desacoplados pero conectados

---

2. Interceptores para gestionar errores

El operador catchError

Gestión centralizada de errores

Creamos un servicio como base del interceptor

```
ng g s notifications/errorInterceptor
```

--

le hacemos implementar la interface HttpInterceptor

```
export class ErrorInterceptorService implements HttpInterceptor {
  constructor() {}

  public intercept(req: HttpRequest<any>, next: HttpHandler)
    : Observable<HttpEvent<any>> {
    return next.handle(req);
  }
}
```

y lo proveemos invirtiendo el control

```
@NgModule({
  declarations: [SenderComponent, ReceiverComponent],
  imports: [
    CommonModule,
    NotificationsRoutingModule,
    HttpClientModule,
    FormsModule
  ],
  providers: [
    {
      provide: HTTP_INTERCEPTORS,
      useClass: ErrorInterceptorService,
      multi: true
    }
  ]
})
export class NotificationsModule {}
```

2.1 El operador catchError

```
public intercept(req, next) {
   return next.handle(req).pipe(tap(null, err=>console.log(err)));
   // return next.handle(req).pipe(catchError(err => of(null)));
   // return next.handle(req).pipe(catchError(err => throwError(err)));
}
```

2.2 Gestión centralizada de errores

```
public intercept(req, next) {
  return next.handle(req).pipe(catchError(this.handleError));
}

private handleError(err) {
  const unauthorized_code = 401;
  let userMessage = 'Fatal error';
  if (err instanceof HttpErrorResponse) {
    if (err.status === unauthorized_code) {
      userMessage = 'Authorization needed';
    } else {
      userMessage = 'Comunications error';
    }
  }
  console.log(userMessage);
  return throwError(err);
}
```

> Recap:

2. Interceptores para gestionar errores

El operador catchError

Gestión centralizada de errores

---

3. Un notificador de problemas

Emisión mediante el Store

Recepción desacoplada del interceptor

3.1 Emisión mediante el Store

```
// dependencia en el constructor
constructor(private notificationsStore: NotificationsStoreService) {}

// En el interceptor.
// Ojo al bind(this), necesario para no perder el contexto
return next.handle(req).pipe(catchError(this.handleError.bind(this)));

private handleError(err) {
  let userMessage = 'Fatal error';
  // emisión de la notificación
  this.notificationsStore.dispatch(userMessage);
}
```

3.2 Recepción desacoplada del interceptor

Por ejemplo desde el ReceiverComponent

```
<button (click)="forceError()">Force http Error</button>
```

```
public forceError() {
  const privateUrl = 'https://api-base.herokuapp.com/api/priv/secrets';
  this.httpClient.get(privateUrl).subscribe();
  const notFoundUrl = 'https://api-base.herokuapp.com/api/pub/items/9';
  this.httpClient.get(notFoundUrl).subscribe();
}
```

> Recap:

3. Un notificador de problemas

Emisión mediante el Store

Recepción desacoplada del interceptor

> Next:

Formularios reactivos con Angular

Desacoplar vista y modelo

El Form Group

Validaciones

> **Blog de apoyo:** Vigilancia y seguridad en Angular
>
> > By Alberto Basalo