8-Reactive

Formularios reactivos con Angular

1. Desacople entre vista y modelo

2. Validación y estados

3. Un gestor de credenciales

1. Desacople entre vista y modelo

Form builder

Form control

Form view

El módulo de seguridad

```
ng g m security --routing true
ng g c security/register
```

app-routing.module.ts

```
{
  path: 'security',
  loadChildren: './security/security.module#SecurityModule'
},
```

security-routing.module.ts

```
const routes: Routes = [
  {
    path: 'register',
    component: RegisterComponent
  },
```

```
    {
      path: '**',
      redirectTo: 'register'
    }
  ];
```

header.component.html

```
<a routerLink="security/register" class="button">Register</a>
```

1.1 Form builder

ReactiveFormModule

```
import { ReactiveFormsModule } from '@angular/forms';
@NgModule({
  declarations: [RegisterComponent],
  imports: [
    CommonModule,
    SecurityRoutingModule,
*   ReactiveFormsModule
  ]
})
export class SecurityModule { }
```

---

register.component.ts

```
export class RegisterComponent implements OnInit {
  public formGroup: FormGroup;

* constructor( private formBuilder: FormBuilder ) { }

  public ngOnInit() {
    this.buildForm();
  }
  private buildForm(){
    this.formGroup = this.formBuilder.group({});
  }
}
```

1.2 Form control

```
  private buildForm() {
    const dateLenght = 10;
    const today = new Date().toISOString().substring(0, dateLength);
    const name = 'JOHN DOE';
  * this.formGroup = this.formBuilder.group({
      registeredOn: today,
      name: name.toLowerCase(),
      email: 'john@angular.io',
      password: ''
    });
  }
```

1.3 Form view

```
  *<form [formGroup]="formGroup">
    <label for="registeredOn">Registered On</label>
    <input name="registeredOn"
  *       formControlName="registeredOn"
          type="date" />
    <label for="name">Name</label>
    <input name="name"
          formControlName="name"
          type="text" />
    <label for="email">E-mail</label>
    <input name="email"
          formControlName="email"
          type="email" />
    <label for="password">Password</label>
    <input name="password"
          formControlName="password"
          type="password" />
  </form>
```

> Recap:

1. Desacople entre vista y modelo

Form builder

Form control

Form view

---

2. Validación y estados

Validadores predefinidos y personalizados

Estados de cambio y validación

2.1 Validadores predefinidos y personalizados

Validators

```
private buildForm() {
  const dateLength = 10;
  const today = new Date().toISOString().substring(0, dateLength);
  const name = 'JOHN DOE';
  const minPassLength = 4;
* this.formGroup = this.formBuilder.group({
    registeredOn: today,
    name: [name.toLowerCase(), Validators.required],
    email: ['john@angular.io', [
      Validators.required, Validators.email
    ]],
    password: ['', [
      Validators.required, Validators.minLength(minPassLength)
    ]]
  });
}
```

Validaciones personalizadas

```
password: ['', [
  Validators.required,
  Validators.minLength(minPassLength),
* this.validatePassword
]]
```

--

```
private validatePassword(control: AbstractControl) {
  const password = control.value;
* let error = null;
  if (!password.includes('$')) {
    error = { ...error, dollar: 'needs a dollar symbol' };
  }
  if (!parseFloat(password[0])) {
    error = { ...error, number: 'must start with a number' };
  }
```

```
  return error;
}
```

2.2 Estados de cambio y validación

Validación general del formulario

```
<button (click)="register()"
    [disabled]="formGroup.invalid">Register me!</button>
```

```
public register() {
* const user = this.formGroup.value;
  console.log(user);
}
```

Validación particular por control

```
public getError(controlName: string): string {
  let error = '';
  const control = this.formGroup.get(controlName);
* if (control.touched && control.errors != null) {
    error = JSON.stringify(control.errors);
  }
  return error;
}
```

```
<span>{{ getError('name')}}</span>
<span>{{ getError('email')}}</span>
<span>{{ getError('password')}}</span>
```

> Recap:

2. Validación y estados

Validadores predefinidos y personalizados

Estados de cambio y validación

---

3. Un gestor de credenciales

Detección y redirección de intrusos

Almacenamiento y uso del token

3.1 Detección y redirección de intrusos

Servicios

```
ng g s security/auth-interceptor
ng g c security/secret
```

Rutas

```
  <a routerLink="security/register" class="button">Register</a>
```

```
const routes: Routes = [
  {
    path: 'register',
    component: RegisterComponent
  },
  {
    path: 'secret',
    component: SecretComponent
  },
  {
    path: '**',
    redirectTo: 'secret'
  }
];
```

```
@NgModule({
  declarations: [RegisterComponent, SecretComponent],
  imports: [CommonModule, SecurityRoutingModule, ReactiveFormsModule,
HttpClientModule],
  providers: [
    {
      provide: HTTP_INTERCEPTORS,
*     useClass: AuthInterceptorService,
      multi: true
```

```
    }
  ]
})
export class SecurityModule {}
```

Interceptor

```
export class AuthInterceptorService implements HttpInterceptor {
  constructor(private router: Router) {}

  public intercept(req: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
    return next.handle(req).pipe(catchError(this.handleError.bind(this)));
  }
  private handleError(err) {
    const unauthorized_code = 401;
    if (err instanceof HttpErrorResponse) {
      if (err.status === unauthorized_code) {
*       this.router.navigate(['security/register']);
      }
    }
    return throwError(err);
  }
}
```

3.2 Almacenamiento y uso del token

Token Store

```
ng g s security/token_store
```

```
export class TokenStoreService {
  private token = '';
* private token$ = new BehaviorSubject<string>('');

  constructor() {}

  public select$ = () => this.token$.asObservable();
  public dispatch(token) {
    this.token = token;
    this.token$.next(this.token);
  }
}
```

Send credentials and dispatch token

```
public register() {
  const url = 'https://api-base.herokuapp.com/api/pub/credentials/registration';
  const user = this.formGroup.value;
  this.httpClient.post<any>(url, user)
*   .subscribe(res => this.tokenStore.dispatch(res.token));
}
```

Get Token en AuthInterceptorService

```
private token = '';
constructor(private router: Router, private tokenStore: TokenStoreService) {
* this.tokenStore.select$()
    .subscribe(token => (this.token = token));
}
```

Use Token

```
public intercept(req: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
  const authHeader = { Authorization: 'bearer ' + this.token };
* const authReq = req.clone({ setHeaders: authHeader });
  return next.handle(authReq)
    .pipe(catchError(this.handleError.bind(this)));
}
```

> Recap:

3. Un gestor de credenciales

Detección y redirección de intrusos

Almacenamiento y uso del token

> Next:

Material design y Angular

Instalación y comandos

Layout básico

Componentes básicos

> **Blog de apoyo:** Formularios reactivos con Angular
>
> > By Alberto Basalo