Laboratorio Angular: Mantenimiento de Personas

Vamos a crear un sistema de mantenimiento de la entidad personas (CRUD: Create, Read, Update, Delete).

Para ello crearemos un módulo que contenga los diferentes artefactos necesarios para la implementación.

Al módulo añadiremos un par de servicios: el servicio DAO (Data Access Object) que centralizará el acceso a los datos en el servidor así como su persistencia, y un servicio ViewModel que gestionará la lógica de presentación.

Así mismo, añadiremos al módulo una serie de componentes que actuaran de vistas.

Modulo:

Crear el módulo de Personas

- ng generate module modulo

Renombrar la carpeta “modulo” a “personas”

Editar personas/modulo.module.ts:

- Renombrar la clase ModuloModule con PersonasModule.
- Añadir la propiedad exports: [] a @NgModule
- Importar:

```
imports: [
    CommonModule, FormsModule, RouterModule.forChild([]),
      MyCoreModule, CommonAppModule,
  ]
```

***Nota**: Cambiar MyCoreModule según corresponda.*

Crear el fichero personas/index.ts del módulo

Editar el fichero personas/index.ts y exportar la clase del módulo:

```
export { PersonasModule } from './modulo.module';
```

Importar el módulo recién creado en el módulo principal:

- Editar app.module.ts
- En la tabla de la propiedad imports de @NgModule añadir PersonasModule

Servicios:

Crear el fichero de los servicios:

- ng generate service personas/servicios

Editar los ficheros de entorno:

- En src\environments\environment.ts, añadir al objeto environment:

```
apiURL: 'http://localhost:4321/api/',
```

- En src\environments\environment.prod.ts, añadir al objeto environment:

```
apiURL: '/api/',
```

Editar el fichero personas/servicios.service.ts

Crear tipo de datos para controlar la transición de estados:

```
export type ModoCRUD = 'list' | 'add' | 'edit' | 'view' | 'delete';
```

Crear la clase base de los servicios DAO:

```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { environment } from 'src/environments/environment';

export abstract class RESTDAOService<T, K> {
  protected baseUrl = environment.apiURL;
  constructor(protected http: HttpClient, entidad: string, protected option = {})
        {
    this.baseUrl += entidad;
  }
  query(): Observable<T> {
    return this.http.get<T>(this.baseUrl, this.option);
  }
  get(id: K): Observable<T> {
    return this.http.get<T>(this.baseUrl + '/' + id, this.option);
  }
  add(item: T): Observable<T> {
    return this.http.post<T>(this.baseUrl, item, this.option);
  }
  change(id: K, item: T): Observable<T> {
    return this.http.put<T>(this.baseUrl + '/' + id, item, this.option);
  }
  remove(id: K): Observable<T> {
    return this.http.delete<T>(this.baseUrl + '/' + id, this.option);
  }
}
```

Añadir el servicio DAO:

```
@Injectable({
  providedIn: 'root'
})
export class PersonasDAOService extends RESTDAOService<any, any> {
  constructor(http: HttpClient) {
    super(http, 'personas', { withCredentials: true });
  }
}
```

Renombrar la clase ServiciosService por PersonasViewModelService

La clase necesita los siguientes atributos:

- Un modo para saber el estado de la operación
- Una colección con el listado de los elementos que se están visualizando.
- Una referencia al elemento que se está visualizando o editando.
- Una cache del identificador del elemento que originalmente se solicitó para su edición.

Crear atributos de la clase PersonasViewModelService:

```
  protected modo: ModoCRUD = 'list';
  protected listado: Array<any> = [];
  protected elemento: any = {};
  protected idOriginal: any = null;
```

Crear constructor e inyectar dependencias:

```
  constructor(protected notify: NotificationService,
              protected out: LoggerService,
              protected dao: PersonasDAOService, ) { }
```

Crear propiedades que expongan los atributos enlazables en las plantillas:

```
  public get Modo() { return this.modo; }
  public get Listado() { return this.listado; }
  public get Elemento() { return this.elemento; }
```

Comando para obtener el listado a mostrar:

```
  public list() {
    this.dao.query().subscribe(
      data => {
        this.listado = data;
        this.modo = 'list';
      },
      err => this.notify.add(err.message)
    );
  }
```

Comandos para preparar las operaciones con la entidad:

```
public add() {
  this.elemento = {};
  this.modo = 'add';
}
public edit(key: any) {
  this.dao.get(key).subscribe(
    data => {
      this.elemento = data;
      this.idOriginal = key;
      this.modo = 'edit';
    },
    err => this.notify.add(err.message)
  );
}
public view(key: any) {
  this.dao.get(key).subscribe(
    data => {
      this.elemento = data;
      this.modo = 'view';
    },
    err => this.notify.add(err.message)
  );
}
public delete(key: any) {
  if (!window.confirm('¿Seguro?')) { return; }

  this.dao.remove(key).subscribe(
    data => this.list(),
    err => this.notify.add(err.message)
  );
}
```

Comandos para cerrar la vista de detalle o el formulario:

```
public cancel() {
  this.elemento = {};
  this.idOriginal = null;
  this.list();
}
public send() {
  switch (this.modo) {
    case 'add':
      this.dao.add(this.elemento).subscribe(
        data => this.cancel(),
        err => this.notify.add(err.message)
      );
      break;
    case 'edit':
      this.dao.change(this.idOriginal, this.elemento).subscribe(
        data => this.cancel(),
        err => this.notify.add(err.message)
      );
      break;
    case 'view':
      break;
  }
}
```

Componente:

Crear ficheros del componente anfitrión:

- ng generate component personas/componente --module personas

Mover ficheros de la carpeta personas/componente a la carpeta personas/ y borrar la carpeta personas/componente.

Renombrar fichero componente.component.html por tmpl-anfitrion.component.html

Crear plantillas del resto de componente (crear ficheros en blanco) en el directorio del módulo:

- tmpl-list.component.html
- tmpl-form.component.html
- tmpl-view.component.html

Sustituir en el componente todo el ComponenteComponent generado por:

```
@Component({
  selector: 'app-personas',
  templateUrl: './tmpl-anfitrion.component.html',
  styleUrls: ['./componente.component.css']
})
export class PersonasComponent implements OnInit {
  constructor(protected vm: PersonasViewModelService) { }
  public get VM() { return this.vm; }
  ngOnInit() {
    this.vm.list();
  }
}
```

Crear resto de componentes:

```
@Component({
  selector: 'app-personas-list',
  templateUrl: './tmpl-list.component.html',
  styleUrls: ['./componente.component.css']
})
export class PersonasListComponent implements OnInit {
  constructor(protected vm: PersonasViewModelService) { }
  public get VM() { return this.vm; }
  ngOnInit() { }
}
@Component({
  selector: 'app-personas-add',
  templateUrl: './tmpl-form.component.html',
  styleUrls: ['./componente.component.css']
})
export class PersonasAddComponent implements OnInit {
  constructor(protected vm: PersonasViewModelService) { }
  public get VM() { return this.vm; }
  ngOnInit() { }
}
@Component({
  selector: 'app-personas-edit',
  templateUrl: './tmpl-form.component.html',
  styleUrls: ['./componente.component.css']
})
export class PersonasEditComponent implements OnInit, OnDestroy {
  constructor(protected vm: PersonasViewModelService) { }
  public get VM() { return this.vm; }
  ngOnInit() { }
  ngOnDestroy() { }
 }
@Component({
  selector: 'app-personas-view',
  templateUrl: './tmpl-view.component.html',
  styleUrls: ['./componente.component.css']
})
export class PersonasViewComponent implements OnInit, OnDestroy {
  constructor(protected vm: PersonasViewModelService) { }
  public get VM() { return this.vm; }
  ngOnInit() { }
  ngOnDestroy() { }
 }
```

Preparar registro de componentes:

```
export const PERSONAS_COMPONENTES = [
  PersonasComponent, PersonasListComponent, PersonasAddComponent, PersonasEditCom
ponent,
  PersonasViewComponent,
];
```

Editar personas/modulo.module.ts, en @NgModule:

- Sustituir declarations: [ComponenteComponent] por declarations: [PERSONAS_COMPONENTES];
- Añadir exports: [PERSONAS_COMPONENTES]

Añadir al menú (según corresponda).

Plantillas:

Componente anfitrión:

Editar tmpl-anfitrion.component.html:

```
<ng-container [ngSwitch]="VM.Modo" >
  <app-personas-add *ngSwitchCase="'add'"></app-personas-add>
  <app-personas-edit *ngSwitchCase="'edit'"></app-personas-edit>
  <app-personas-view *ngSwitchCase="'view'"></app-personas-view>
  <app-personas-list *ngSwitchDefault></app-personas-list>
</ng-container>
```

Componente listado:

Editar tmpl-list.component.html:

```
<table>
  <thead>
    <tr>
      <th>Nombre</th>
      <td><input type="button" value="Añadir" (click)="VM.add()"></td>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let item of VM.Listado">
      <td>{{item.nombre}} {{item.apellidos}}</td>
      <td>
        <input type="button" value="Ver" (click)="VM.view(item.id)">
        <input type="button" value="Editar" (click)="VM.edit(item.id)">
        <input type="button" value="Borrar" (click)="VM.delete(item.id)">
      </td>
    </tr>
  </tbody>
</table>
```

Componente de detalle:

Editar tmpl-view.component.html:

```
<p>
  <b>Código:</b> {{VM.Elemento.id}}<br>
  <b>Nombre:</b> {{VM.Elemento.nombre}}<br>
  <b>Apellidos:</b> {{VM.Elemento.apellidos}}<br>
  <b>Edad:</b> {{VM.Elemento.edad}}
</p>
<p>
    <input type="button" value="Volver" (click)="VM.cancel()">
</p>
```

Componentes con formularios:

Editar tmpl-form.component.html:

```
<form #miForm="ngForm">
  <p>
    <b>Código:</b>
    <ng-container *ngIf="VM.Modo === 'add'">
      <input type="number" id="id" name="id" [(ngModel)]="VM.Elemento.id"
#id="ngModel" required>
      <span class="error" [hidden]="!id.hasError('required') || miForm.pristine">
Es obligatorio</span>
    </ng-container>
    <ng-container *ngIf="VM.Modo !== 'add'">
      {{VM.Elemento.id}}
    </ng-container>
    <br>
    <b>Nombre:</b>
    <input type="text" id="nombre" name="nombre" [(ngModel)]="VM.Elemento.nombre"
 #nombre="ngModel" required
      maxlength="10" minlength="2">
    <span class="error" [hidden]="!nombre?.errors?.required">Es obligatorio
      </span>
    <span class="error"
      [hidden]="!nombre?.errors?.minlength && !nombre?.errors?.maxlength">
      Debe tener entre 2 y 10 letras</span>
    <br>
    <b>Apellidos:</b>
    <input type="text" id="apellidos" name="apellidos"
      [(ngModel)]="VM.Elemento.apellidos" #apellidos="ngModel"
      maxlength="10" minlength="2" >
    <span class="error"
      [hidden]="!apellidos?.errors?.minlength && !apellidos?.errors?.maxlength">
      Debe tener entre 2 y 10 letras</span>
    <br>
    <b>Edad:</b>
    <input type="number" id="edad" name="edad" [(ngModel)]="VM.Elemento.edad"
      #edad="ngModel" min="16" max="67">
    <span class="error" [hidden]="!edad?.errors?.min && !edad?.errors?.max">
        Debe estar entre los 16 y 67 años</span>
  </p>
  <p>
    <input type="button" value="Enviar" (click)="VM.send()"
      [disabled]="miForm.invalid">
    <input type="button" value="Volver" (click)="VM.cancel()">
  </p>
</form>
```