Laboratorio Angular:
Mantenimiento de Contactos

Vamos a crear un sistema de mantenimiento de la entidad contactos (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 Contactos

- ng generate module modulo

Renombrar la carpeta “modulo” a “contactos”

Editar contactos/modulo.module.ts:

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

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

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

Crear el fichero contactos/index.ts del módulo

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

```
export * 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 ContactosModule

Servicios:

Crear el fichero de los servicios:

- ng generate service contactos/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 contactos/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 ContactosDAOService extends RESTDAOService<any, any> {
  constructor(http: HttpClient) {
    super(http, 'contactos', { withCredentials: true });
  }
}
```

Renombrar la clase ServiciosService por ContactosViewModelService

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 ContactosViewModelService:

```
  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: ContactosDAOService) { }
```

Crear propiedades que expongan los atributos enlazables en las plantillas:

```
  public get Modo(): ModoCRUD { return this.modo; }
  public get Listado(): Array<any> { return this.listado; }
  public get Elemento(): any { return this.elemento; }
```

Comando para obtener el listado a mostrar:

```
  public list(): void {

    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(): void {
  this.elemento = {};
  this.modo = 'add';
}
public edit(key: any): void {
  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): void {
  this.dao.get(key).subscribe(
    data => {
      this.elemento = data;
      this.modo = 'view';
    },
    err => this.notify.add(err.message)
  );
}
public delete(key: any): void {
  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(): void {
  this.elemento = {};
  this.idOriginal = null;
  this.list();
}
```

```
  public send(): void {
    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 contactos/componente --module contactos

Mover ficheros de la carpeta contactos/componente a la carpeta contactos/ y borrar la carpeta contactos/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 toda la clase ComponenteComponent generada por:

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

Crear resto de componentes:

```
@Component({
  selector: 'app-contactos-list',
  templateUrl: './tmpl-list.component.html',
  styleUrls: ['./componente.component.css']
})
export class ContactosListComponent implements OnInit {
  constructor(protected vm: ContactosViewModelService) { }
  public get VM(): ContactosViewModelService { return this.vm; }
  ngOnInit(): void { }
}
@Component({
  selector: 'app-contactos-add',
  templateUrl: './tmpl-form.component.html',
  styleUrls: ['./componente.component.css']
})
export class ContactosAddComponent implements OnInit {
  constructor(protected vm: ContactosViewModelService) { }
  public get VM(): ContactosViewModelService { return this.vm; }
  ngOnInit(): void { }
}
@Component({
  selector: 'app-contactos-edit',
  templateUrl: './tmpl-form.component.html',
  styleUrls: ['./componente.component.css']
})
export class ContactosEditComponent implements OnInit, OnDestroy {
  constructor(protected vm: ContactosViewModelService) { }
  public get VM(): ContactosViewModelService { return this.vm; }
  ngOnInit(): void { }
  ngOnDestroy(): void { }
 }
@Component({
  selector: 'app-contactos-view',
  templateUrl: './tmpl-view.component.html',
  styleUrls: ['./componente.component.css']
})
export class ContactosViewComponent implements OnInit, OnDestroy {
  constructor(protected vm: ContactosViewModelService) { }
  public get VM(): ContactosViewModelService { return this.vm; }
  ngOnInit(): void { }
  ngOnDestroy(): void { }
 }
```

Preparar registro de componentes:

```
export const CONTACTOS_COMPONENTES = [
  ContactosComponent, ContactosListComponent, ContactosAddComponent,
  ContactosEditComponent, ContactosViewComponent,
];
```

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

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

Añadir al menú (según corresponda).

Plantillas:

Componente anfitrión:

Editar tmpl-anfitrion.component.html:

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

Componente listado:

Editar tmpl-list.component.html:

```
<table class="table table-striped table-hover">
 <thead>
  <tr class="table-info">
   <th class="display-4">Lista de contactos</th>
   <th class="text-right"><input type="button" class="btn btn-success"
     value="Añadir" (click)="VM.add()"></th>
  </tr>
 </thead>
 <tbody>
  <tr *ngFor="let item of VM.Listado">
   <td>
    <div class="container">
     <div class="row">
      <div class="col-md-1">
       <img class="rounded-circle float-left" src="{{item.avatar}}"
        alt="Foto de {{item.nombre}} {{item.apellidos}}" width="75" height="75">
      </div>
      <div class="col-md-11">
       <button type="button" class="btn btn-link btn-lg" (click)="VM.view(item.id)">{{item.tratamiento}}
        {{item.nombre}} {{item.apellidos}}</button>
       <br>
       <b>Tlfn.:</b> {{item.telefono}} <b>Correo:</b> {{item.email}}
      </div>
     </div>
    </div>
   </td>
   <td class="btn-group float-right">
    <button class="btn btn-info" (click)="VM.view(item.id)"><i class="fas fa-eye"></i></button>
    <button class="btn btn-success" (click)="VM.edit(item.id)"><i class="fas fa-pen"></i></button>
    <button class="btn btn-danger" (click)="VM.delete(item.id)"><i class="far fa-trash-alt"></i></button>
   </td>
  </tr>
 </tbody>
</table>
```

Componente de detalle:

Editar tmpl-view.component.html:

```
<h1>Contacto</h1>
<div class="row">
 <div class="col-xs-12 col-sm-6 col-md-6">
   <div class="well well-sm">
     <div class="row">
       <div class="col-sm-6 col-md-4">
         <img src="{{VM.Elemento.avatar}}"
             alt="Foto de {{VM.Elemento.nombre}} {{VM.Elemento.apellidos}}" class="rounded" />
       </div>
       <div class="col-sm-6 col-md-8">
         <h4>{{VM.Elemento.tratamiento}} {{VM.Elemento.nombre}} {{VM.Elemento.apellidos}}</h4>
         <p [hidden]="!VM.Elemento.conflictivo"><small class="text-danger">
             <i class="fas fa-skull-crossbones mr-2"></i>Persona conflictiva</small></p>
         <p>
           <i class="fas fa-phone-alt mr-2"></i>{{VM.Elemento.telefono}}
           <br />
           <i class="fas fa-envelope mr-2"></i>
            <a href="mailto:{{VM.Elemento.email}}">{{VM.Elemento.email}}</a>
           <br />
           <i class="fas fa-gifts mr-2"></i>{{VM.Elemento.nacimiento | date:'dd/MM/yyyy'}}
         </p>
         <div class="btn-group">
           <button type="button" class="btn btn-primary">
             Social</button>
           <button type="button" class="btn btn-primary dropdown-toggle" data-toggle="dropdown">
             <span class="caret"></span><span class="sr-only">Social</span>
           </button>
           <ul class="dropdown-menu" role="menu">
             <li><a href="#">Twitter</a></li>
             <li><a href="https://plus.google.com/+Jquery2dotnet/posts">Google +</a></li>
             <li><a href="https://www.facebook.com/jquery2dotnet">Facebook</a></li>
             <li class="divider"></li>
             <li><a href="#">Github</a></li>
           </ul>
           <input class="btn btn-secondary" type="button" value="Volver" (click)="VM.cancel()">
         </div>
       </div>
     </div>
   </div>
 </div>
</div>
```

Componentes con formularios:

Editar tmpl-form.component.html:

```
<h1>Contacto</h1>
<form #miForm="ngForm">
 <div class="form-row">
  <div class="form-group col-md-2">
   <label class="form-label" for="id">Código:</label>
   <input class="form-control" [class.is-invalid]="id.invalid && miForm.dirty" type="number" name="id"
    id="id" [(ngModel)]="VM.Elemento.id" #id="ngModel" required min="0">
   <div class="invalid-feedback" [hidden]="!id.hasError('required') || miForm.pristine">Es obligatorio</div>
  </div>
  <div class="form-group col-md-2">
   <label class="form-label" for="tratamiento">Tratamiento:</label>
   <select class="form-control form-select" name="tratamiento" id="tratamiento"
    [(ngModel)]="VM.Elemento.tratamiento" #tratamiento="ngModel">
    <option>Sr.</option>
    <option>Sra.</option>
    <option>Srta.</option>
    <option>Dr.</option>
    <option>Dra.</option>
    <option>Ilmo.</option>
    <option>Ilma.</option>
    <option>Excmo.</option>
    <option>Excma.</option>
   </select>
  </div>
  <div class="form-group col-md-4">
   <label class="form-label" for="nombre">Nombre:</label>
   <input class="form-control" [class.is-invalid]="nombre.invalid" type="text" name="nombre" id="nombre"
    [(ngModel)]="VM.Elemento.nombre" #nombre="ngModel" required minlength="2" maxlength="50">
   <div class="invalid-feedback" [hidden]="!nombre?.errors?.required">Es obligatorio</div>
   <div class="invalid-feedback" [hidden]="!nombre?.errors?.minlength && !nombre?.errors?.maxlength">
    Debe tener entre 2 y 10 letras</div>
  </div>
  <div class="form-group col-md-4">
   <label class="form-label" for="apellidos">Apellidos:</label>
   <input class="form-control" [class.is-invalid]="apellidos.invalid" type="text" name="apellidos"
    id="apellidos"  [(ngModel)]="VM.Elemento.apellidos" #apellidos="ngModel" minlength="2"
    maxlength="50">
   <div class="invalid-feedback"
    [hidden]="!apellidos?.errors?.minlength && !apellidos?.errors?.maxlength">
    Debe tener entre 2 y 10 letras</div>
  </div>
 </div>
 <div class="form-row">
  <div class="form-group col-md-2">
   <label class="form-label" for="telefono">Telefono:</label>
   <input class="form-control" [class.is-invalid]="telefono.invalid" type="tel" name="telefono" id="telefono"
```

```
    [(ngModel)]="VM.Elemento.telefono" #telefono="ngModel" maxlength="11">
   <div class="invalid-feedback" [hidden]="!telefono?.errors?.maxlength">
    No debe tener mas de 9 dígitos</div>
  </div>
  <div class="form-group col-md-4">
   <label class="form-label" for="email">Correo:</label>
   <input class="form-control" [class.is-invalid]="email.invalid" type="email" name="email" id="email"
    [(ngModel)]="VM.Elemento.email" #email="ngModel" email lowercase maxlength="100">
   <div class="invalid-feedback" [hidden]="email.valid"><span [hidden]="!email?.errors?.lowercase">
    Debe ir en minúsculas. </span>
   <span [hidden]="!email?.errors?.maxlength">Debe tener menos 100 caracteres. </span>
   <span [hidden]="!email?.errors?.email">Debe ser un correo electrónico. </span></div>
  </div>
  <div class="form-group col-md-2">
   <label class="form-label" for="nacimiento">F. Nacimiento:</label>
   <input class="form-control" type="date" name="nacimiento" id="nacimiento"
    [(ngModel)]="VM.Elemento.nacimiento" #nacimiento="ngModel">
  </div>
  <fieldset class="form-group col-md-2">
   <legend class="col-form-label col-sm-2 pt-0">Sexo</legend>
   <div class="col-sm-10">
    <div class="form-check">
     <input class="form-check-input" type="radio" name="sexo" id="sexo1" value="H"
     [(ngModel)]="VM.Elemento.sexo">
     <label class="form-check-label" for="sexo1">Hombre</label>
    </div>
    <div class="form-check">
     <input class="form-check-input" type="radio" name="sexo" id="sexo2" value="M"
     [(ngModel)]="VM.Elemento.sexo">
     <label class="form-check-label" for="sexo2">Mujer</label>
    </div>
   </div>
  </fieldset>
  <div class="form-group col-md-2">
   <div>Situación:</div>
   <div>
    <div class="form-check">
     <input class="form-check-input" type="checkbox" id="conflictivo" name="conflictivo"
      [(ngModel)]="VM.Elemento.conflictivo" #conflictivo="ngModel">
     <label class="form-check-label" for="conflictivo">Conflictivo</label>
    </div>
   </div>
  </div>
 </div>
 <div class="form-group row">
  <div class="form-group col-md-12">
   <label class="form-label" for="email">Avatar:</label>
   <input class="form-control" type="url" name="avatar" id="avatar" [(ngModel)]="VM.Elemento.avatar"
    #avatar="ngModel" maxlength="500">
```

```
    <div class="invalid-feedback" [hidden]="!avatar?.errors?.maxlength">
      Debe tener menos 500 caracteres.</div>
   </div>
  </div>
  <div class="form-group">
   <input type="button" class="btn btn-success" value="Enviar" (click)="VM.send()"
       [disabled]="miForm.invalid">
   <input type="button" class="btn btn-info" value="Volver" (click)="VM.cancel()">
  </div>
</form>
```

Ampliación: Actualización a un sistema enrutado

Vamos a modificar el sistema de mantenimiento de la entidad contactos para adaptarlo al uso natural de rutas (URL) del navegador y permitir el acceso directo a los componentes mediante sus rutas.

Para ello dejaremos de usar el componente anfitrión para ir directamente a los componentes específicos, por lo que abra que modificar la inicialización de los mismos. El modo pierde funcionalidad en pro de la las rutas que indican la operación a realizar.

Así mismo, el modo comando mediante invocación de métodos debe ser sustituido por hipervínculos a las rutas que representan las operaciones. Por la misma razón hay que cambiar la vuelta atrás por la invocación de una ruta.

Tabla de rutas:

Editar el fichero app-routing.module.ts (o el que corresponda) para agregar las nuevas rutas:

```
  { path: 'contactos', children: [
    { path: '', component: ContactosListComponent},
    { path: 'add', component: ContactosAddComponent},
    { path: ':id/edit', component: ContactosEditComponent},
    { path: ':id', component: ContactosViewComponent},
    { path: ':id/:kk', component: ContactosViewComponent},
  ]},
```

Componentes:

Dado que ahora los componentes van a ser invocados directamente es necesario modificar su proceso de inicialización para que ejecuten los comandos apropiados que cambien el modo del ViewModel:

Editar el fichero contactos/componente.component.ts

Modificar el método ngOnInit de los componentes ContactosListComponent y ContactosAddComponent:

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

```
@Component({
  selector: 'app-contactos-add',
  templateUrl: './tmpl-form.component.html',
  styleUrls: ['./componente.component.css']
})
export class ContactosAddComponent implements OnInit {
  constructor(protected vm: ContactosViewModelService) { }
  public get VM(): ContactosViewModelService { return this.vm; }
  ngOnInit(): void {
    this.vm.add();
  }
}
```

En los componentes ContactosEditComponent y ContactosViewComponent es necesario inyectar los servicios ActivatedRoute y Router para extraer el identificador de la ruta antes de invocar al ViewModel o redirigir en caso de error. Para ello es necesario crear una suscripción que debe ser eliminada al destruir el componente.

```
@Component({
  selector: 'app-contactos-edit',
  templateUrl: './tmpl-form.component.html',
  styleUrls: ['./componente.component.css']
})
export class ContactosEditComponent implements OnInit, OnDestroy {
  private obs$: any;
  constructor(protected vm: ContactosViewModelService,
    protected route: ActivatedRoute, protected router: Router) { }
  public get VM(): ContactosViewModelService { return this.vm; }
  ngOnInit(): void {
    this.obs$ = this.route.paramMap.subscribe(
      (params: ParamMap) => {
        const id = parseInt(params?.get('id') ?? '');
        if (id) {
          this.vm.edit(id);
        } else {
          this.router.navigate(['/404.html']);
        }
      });
  }
  ngOnDestroy(): void {
    this.obs$.unsubscribe();
  }
}
```

```
@Component({
  selector: 'app-contactos-view',
  templateUrl: './tmpl-view.component.html',
  styleUrls: ['./componente.component.css']
})
export class ContactosViewComponent implements OnInit, OnDestroy {
  private obs$: any;
  constructor(protected vm: ContactosViewModelService,
    protected route: ActivatedRoute, protected router: Router) { }
  public get VM(): ContactosViewModelService { return this.vm; }
  ngOnInit(): void {
    this.obs$ = this.route.paramMap.subscribe(
      (params: ParamMap) => {
        const id = parseInt(params?.get('id') ?? '');
        if (id) {
          this.vm.view(id);
        } else {
          this.router.navigate(['/404.html']);
        }
      });
  }
  ngOnDestroy(): void {
    this.obs$.unsubscribe();
  }
}
```

Servicios:

Editar el fichero contactos/servicios.service.ts para modificar la clase ContactosViewModelService.

Agrega un atributo con la ruta de vuelta atrás para facilitar el mantenimiento de la clase:

```
  protected listURL = '/contactos';
```

Inyectar el servicio Router para redirigir la vuelta atrás a la ruta indicada por el nuevo atributo:

```
  constructor(protected notify: NotificationService, protected out: LoggerService,
              protected dao: ContactosDAOService, protected router: Router) { }
```

Modificar el método de vuelta atrás para redirigir a la ruta indicada por el nuevo atributo:

```
  public cancel(): void {
    this.elemento = {};
    this.idOriginal = null;
    // this.list();
    this.router.navigateByUrl(this.listURL);
  }
```

Plantillas:

Componente listado:

Editar tmpl-list.component.html:

```
<table class="table table-striped table-hover">
 <thead>
  <tr class="table-info">
   <th class="display-4">Lista de contactos</th>
   <th class="text-right">
       <input type="button" class="btn btn-success" value="Añadir" routerLink="add">
   </th>
  </tr>
 </thead>
 <tbody>
  <tr *ngFor="let item of VM.Listado">
   <td>
    <div class="container">
     <div class="row">
      <div class="col-md-1">
       <img class="rounded-circle float-left" src="{{item.avatar}}"
        alt="Foto de {{item.nombre}} {{item.apellidos}}" width="75" height="75">
      </div>
      <div class="col-md-11">
       <a class="btn btn-link btn-lg" routerLink="{{item.id}}/{{item.nombre}}-{{item.apellidos}}">
        {{item.tratamiento}} {{item.nombre}} {{item.apellidos}}</a>
       <br>
       <b>Tlfn.:</b> {{item.telefono}} <b>Correo:</b> {{item.email}}
      </div>
     </div>
    </div>
   </td>
   <td class="btn-group float-right">
    <button class="btn btn-info" routerLink="{{item.id}}/{{item.nombre}}-{{item.apellidos}}">
      <i class="fas fa-eye"></i></button>
    <button class="btn btn-success" [routerLink]="[item.id, 'edit']"><i class="fas fa-pen"></i></button>
    <button class="btn btn-danger" (click)="VM.delete(item.id)"><i class="far fa-trash-alt"></i></button>
   </td>
  </tr>
 </tbody>
</table>
```

Componente cabecera:

Editar el correspondiente componente con el menú de la aplicación para agregar el enlace al componente listado actualizado como punto de entrada a los contactos:

```
<li class="nav-item">
  <a class="nav-link" [routerLink]="['/contactos']" routerLinkActive="active">
    contactos</a>
</li>
```