

Exercises Feedback:

1. Se debe configurar CORS en Program.cs para permitir la conexión con Angular en la otra dirección:

```
(...)  
// Add services to the container.  
builder.Services.AddControllers();  
  
//Configure CORS to connect to Angular  
builder.Services.AddCors(options =>  
{  
    options.AddPolicy("AllowAngularApp", policy =>  
    {  
        policy.WithOrigins("http://localhost:62089") // Your Angular app's URL  
            .AllowAnyHeader()  
            .AllowAnyMethod();  
    });  
});  
  
(...)  
app.UseCors("AllowAngularApp");  
  
app.UseHttpsRedirection();  
  
app.UseAuthorization();  
  
app.MapControllers();  
  
app.Run();
```

2. Si necesito inicializar alguna variable que depende por ejemplo de una interfaz que se encuentra en el service.ts (en este caso), debo inicializar esa variable OnInit y no en la declaración de variables, de otra forma tendré un error de redundancia en las dependencias:

```
export class PatientManagementComponent implements OnInit{  
  
    patientsList: IPatient[] = [];  
    newPatient!: IPatient;  
    selectedPatient!: IPatient | null;  
  
    constructor(private _patientService: PatientService) { }  
  
    ngOnInit() {  
        this.newPatient = this.getEmptyPatientObject();  
        this.loadPatients();  
    }  
}
```

3. Después de instalar bootstrap, es necesario incluir las librerías en 'angular.json', de lo contrario no se implementarán los estilos bootstrap, así:

```
"styles": [  
    "src/styles.css",  
    "node_modules/bootstrap/dist/css/bootstrap.min.css"  
],  
"scripts": [  
    "node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"  
]
```

4. Para poder implementar `[formGroup]` y `formControlName` en el formulario, es necesario importar `ReactiveFormsModule` en `app.module.ts`

5. El constructor del componente del formulario requiere instanciar el objeto `FormGroup`, para tomar los valores de los inputs, así:

```
patientForm!: FormGroup;

constructor(private _formBuilder: FormBuilder, private _patientService:
PatientService) {
  this.patientForm = this._formBuilder.group({
    firstName: ['', [Validators.required, Validators.minLength(3)]],
    lastName: ['', [Validators.required, Validators.minLength(3)]],
    email: ['', [Validators.required, Validators.email]],
    healthCardNum: ['', [Validators.required, Validators.minLength(10)]],
    gender: [''],
    birthDate: ['']
  });
}
```

6. Para implementar `routerLink` (SPA) en los links, se debe importar el módulo `RouterLink`

7. La sintaxis para establecer las rutas es la siguiente:

```
const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: "admissions", component: AdmissionsComponent },
  { path: "**", redirectTo: '', pathMatch: 'full' }
];
```

8. Para usar `[(ngModel)]`, como por ejemplo en la siguiente línea de código:

```
<select id="physician" [(ngModel)]="selectedPhysicianId" class="form-select"
(change)="loadAdmissions()" ">
```

Se debe importar **FormsModule** en `app.module.ts`

`[(ngModel)]` se usa para pasar parámetros desde el HTML hacia una variable en el TS (logica). En el caso específico de este ejemplo el elemento `<select>` le pasa al TS el valor seleccionado en la lista desplegable y con ese valor se ejecuta un método para cargar los valores relacionados con ese Physician.

```
<div class="container">
  <h1>Admissions Database</h1>
  <label>Select a Physician:</label>
  <select id="physician" [(ngModel)]="selectedPhysicianId" class="form-select" (change)="loadAdmissions()">
    <option *ngFor="let p of physicians" [value]="p.physicianId">{{p.firstName}} {{p.lastName}}</option>
  </select>
  <app-admissions-table [admissions]="admissions" [loading]="loading"/>
</div>
```

```
import { Component, OnInit } from '@angular/core';
import { ApiServiceService, IAdmissionDetails, IPhysician } from '../services/api-service.service';

@Component({
  selector: 'app-admissions',
  templateUrl: './admissions.component.html',
  styleUrls: ['./admissions.component.css']
})
export class AdmissionsComponent implements OnInit {
  physicians: IPhysician[] = [];
  admissions: IAdmissionDetails[] = [];
  selectedPhysicianId: number | null = null;
  loading: boolean = true;

  constructor(private _apiService: ApiServiceService) {}

  ngOnInit(): void {
    this.loadPhysicians();
  }

  loadPhysicians() {
    this._apiService.getPhysicians().subscribe(data => {
      this.physicians = data;
      this.loading = false;
    });
  }

  loadAdmissions() {
    if (this.selectedPhysicianId) {
      this._apiService.getAdmissionByPhysician(this.selectedPhysicianId).subscribe((data) => {
        this.admissions = data;
        this.loading = false;
      });
    }
  }
}
```

9. Para realizar la inclusión de un componente hijo en un padre, los "props" se deben enviar de la siguiente forma desde el HTML padre:

```
<div class="container">
  <h1>Admissions Database</h1>
  <label>Select a Physician:</label>
  <select id="physician" [(ngModel)]="selectedPhysicianId" class="form-select" (change)="loadAdmissions()">
    <option *ngFor="let p of physicians" [value]="p.physicianId">{{p.firstName}} {{p.lastName}}</option>
  </select>
  <app-admissions-table [admissions]="admissions" [loading]="loading"/>
</div>
```

Child component (pointing to `<app-admissions-table>`)

Props (pointing to `[admissions]` and `[loading]`)

Adicionalmente en el archivo .ts del componente hijo se deben usar los módulos @Input() para recibir los props:

```
import { Component, Input } from '@angular/core';
import { IAdmissionDetails } from '../../services/api-service.service';

@Component({
  selector: 'app-admissions-table',
  templateUrl: './admissions-table.component.html',
  styleUrls: ['./admissions-table.component.css']
})
export class AdmissionsTableComponent {
  @Input() admissions: IAdmissionDetails[] = [];
  @Input() loading: boolean = true;
}
```

Importar el modulo

Modificar el nombre a plural

Declarar las entradas

Finalmente, por alguna razón, también debe cambiar el nombre de la propiedad *styleUrl* que por defecto esta en singular, a plural *styleUrls*, tanto en el componente hijo como en el componente padre y hacerlo array, por ese se pone entre llaves cuadradas.

10. Sin embargo, cuando los parámetros se quieren pasar anidando paginas (pages) en lugar del componente hijo explicito dentro del componente padre, esto se debe hacer mediante **params**, así:

componente HTML del padre:

```
<div class="container">
  <h1>List of Patients</h1>
  <table class="table table-striped">
    <thead>
      <tr>
        <th scope="col">Patient Id</th>
        <th scope="col">Name</th>
        <th scope="col">Gender</th>
        <th scope="col">Birth Date</th>
        <th scope="col">View Details</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let p of patients">
        <th scope="row">{{p.patientId}}</th>
        <td>{{p.firstName}} {{p.lastName}}</td>
        <td>{{p.gender}}</td>
        <td>{{p.birthDate | date: 'dd/MM/yyyy'}}</td>
        <td>
          <button type="button" class="btn btn-outline-primary" [routerLink]="['/patients', p.patientId]">View Detail</button>
        </td>
      </tr>
    </tbody>
  </table>
</div>
```

se usa el descriptor [routerLink] y se pasa la ruta junto con el parametro esperado por la ruta

Componente TS de la nueva ruta a la que voy ("hijo"):

```
import { Component, OnInit } from '@angular/core';
import { ApiPatientsService, IPatient } from '../services/api-patients.service';
import { ActivatedRoute, Params } from '@angular/router';

@Component({
  selector: 'app-patient-details',
  templateUrl: './patient-details.component.html',
  styleUrls: ['./patient-details.component.css']
})
export class PatientDetailsComponent implements OnInit {

  patientsList?: IPatient[];
  patient!: IPatient;

  constructor(private _route: ActivatedRoute, private _patientService: ApiPatientsService) { }

  ngOnInit(): void {
    this.patient = this.initializePatientObject();
    this.loadPatients()
  }

  loadPatients() {
    this._route.params.subscribe({
      next: (params: Params) => {
        console.log(params['id']);
        this.setPatientData(params['id']);
      }
    })
  }

  setPatientData(id:string) {
    this._patientService.getPatientById(Number(id)).subscribe(data => {
      console.log(data);
      this.patient = data;
    })
  }
}
```

Usa params para tomar la informacion del componente padre.

Una vez se tiene el parametro que en este caso es patientid, ahi si puedo llamar la API

10. Establecer parámetros de la base de datos en appsettings.json:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "chddb":
"Server=(LocalDB)\\MSSQLLocalDB;Database=chddb;Trusted_Connection=True"
  }
}
```

11. Scaffold-DbContext

1. Install-Package Microsoft.EntityFrameworkCore.Tools -Version 7
2. Install-Package Microsoft.EntityFrameworkCore.SqlServer -Version 7
3. Scaffold-DbContext name=chddb Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Tables physicians

12. Editar Program.cs

```
// Add services to the container.
builder.Services.AddControllers();
builder.Services.AddDbContext<ChddbContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("chddb")));
```

13. Para eliminar registros existentes desde un componente hijo hacia un componente padre que esta manejando la logica, se deben usar propiedades @Input, @Output y EventEmitter:

Componente –hijo–

```
import { Component, Input, Output, EventEmitter } from '@angular/core';
import { IPatient } from '../services/api-patients.service';

@Component({
  selector: 'app-patients-table',
  templateUrl: './patients-table.component.html',
  styleUrls: ['./patients-table.component.css']
})
0 referencias
export class PatientsTableComponent {
  @Input() patients?: IPatient[];
  @Output() deletePatientEvent = new EventEmitter<number>();

  deletePatient(patientId: number) {
    this.deletePatientEvent.emit(patientId);
  }
}
```

Adicionalmente la conexión entro los componentes se finaliza en los archivos HTML.

HTML –hijo–

```
<table class="table table-striped">
  <thead>
    <tr>
      <th scope="col">Patient Id</th>
      <th scope="col">Name</th>
      <th scope="col">Gender</th>
      <th scope="col">Birth Date</th>
      <th scope="col">View Details</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let p of patients">
      <th scope="row">{{p.patientId}}</th>
      <td>{{p.firstName}} {{p.lastName}}</td>
      <td>{{p.gender}}</td>
      <td>{{p.birthDate | date: 'dd/MM/yyyy'}}</td>
      <td>
        <button type="button" class="btn btn-outline-primary" [routerLink]="['/patients', p.patientId]">View Detail</button>
        <button type="button" class="btn btn-outline-danger" (click)="deletePatient(p.patientId)">Delete</button>
      </td>
    </tr>
  </tbody>
</table>
```

Componente –Padre

```
import { Component, OnInit } from '@angular/core';
import { ApiPatientsService, IPatient } from '../services/api-patients.service';

@Component({
  selector: 'app-patients',
  templateUrl: './patients.component.html',
  styleUrls: ['./patients.component.css']
})
9 referencias
export class PatientsComponent implements OnInit {

  patients: IPatient[] = [];

  0 referencias
  constructor(private _apiService: ApiPatientsService) { }

  ngOnInit(): void {
    this.loadPatients();
  }

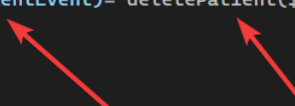
  loadPatients() {
    this._apiService.getPatients().subscribe(data => {
      this.patients = data;
    })
  }

  deletePatient(patientId: number) {
    this._apiService.deletePatient(patientId).subscribe(() => {
      this.loadPatients();
    })
  }
}
```

HTML –padre-

```
<div class="container">
  <h1>List of Patients</h1>
  <ng-container *ngIf="patients.length > 0; else noPatients">
    <app-patients-table [patients]="patients" (deletePatientEvent)="deletePatient($event)"/>
  </ng-container>

  <ng-template #noPatients>
    <h3>Fetching patients information...</h3>
  </ng-template>
</div>
```



Contact Form

1. Para poner a funcionar el formulario tomado de Bootstrap, es necesario importar los módulos:
 - `ReactiveFormsModule`,
 - `FormsModule`

```
export class NewPatientComponent implements OnInit {
  patientForm!: FormGroup;
  newPatient!: IPatient;
  patientsList?: IPatient[] = [];

  constructor(private _formBuilder: FormBuilder, private _patientService: ApiPatientsService) {
    this.patientForm = this._formBuilder.group({
      firstName: [''],
      lastName: [''],
      gender: [''],
      birthDate: [''],
      streetAddress: [''],
      city: [''],
      provinceId: [''],
      postalCode: [''],
      email: [''],
      healthCardNum: [''],
      allergies: [''],
      patientHeight: [''],
      patientWeight: ['']
    });
  }
}
```

Estos nombres deben coincidir con los nombre en `formControlName` en cada elemento `input` de `HTML` y además con los nombres de las propiedades del Objeto

2. La API debe inyectarse en el constructor para generar el método `POST` en el backend y una vez hecho eso, se recarga la base de datos:

```
submit(event: Event) {
  event.preventDefault();
  this.newPatient = this.patientForm.value;
  console.log(this.newPatient);
  this.createNewPatient(this.newPatient);
}

createNewPatient(newPatient: IPatient) {
  this._patientService.addPatient(newPatient).subscribe(() => {
    this.loadPatients();
  })
}

loadPatients() {
  this._patientService.getPatients().subscribe(data => {
    this.patientsList = data;
  })
}
```

3. En el elemento del formulario `<form>` es necesario llamar la propiedad `[formGroup]` y e iniciar el evento `submit` para prevenir que se accione con el botón antes de terminar el formulario

```
<form [formGroup]="patientForm" class="row g-3" (submit)="submit($event)">
```