Manual: Desarrollo de Aplicaciones con Angular - Ejemplo con Pokemon App

1. Introducción a Angular

1.1 ¿Qué es Angular?

Angular es un **framework de desarrollo frontend** creado por Google, diseñado para construir aplicaciones web dinámicas y de una sola página (SPA, Single Page Applications). Utiliza TypeScript (un superconjunto de JavaScript) y sigue un enfoque basado en componentes, lo que permite dividir la interfaz de usuario en partes reutilizables y modulares.

Características principales de Angular:

- **Arquitectura basada en componentes**: La UI se divide en componentes reutilizables.

- **Inyección de dependencias**: Permite inyectar servicios y otros recursos en los componentes.

- **Enrutamiento**: Soporta navegación entre vistas sin recargar la página.

- **Data Binding**: Sincroniza automáticamente los datos entre el modelo y la vista.

- **Directivas**: Extiende la funcionalidad del HTML con lógica personalizada.

- **Servicios**: Proporciona una forma de manejar lógica de negocio y datos de manera centralizada.

Ejemplo en el proyecto:

En tu proyecto `Pokemon App`, usamos Angular para crear una aplicación que permite listar Pokémon, agregarlos a una colección, y gestionarlos, todo

mientras se mantiene una interfaz consistente y un flujo de navegación fluido.

---

2. Estructura General de una Aplicación Angular

Una aplicación Angular sigue una estructura modular y basada en componentes. Los elementos clave incluyen:

- **Componentes**: Bloques de construcción de la UI.
- **Módulos**: Agrupan componentes, directivas y servicios (en aplicaciones más antiguas; en este proyecto usamos standalone components).
- **Servicios**: Manejan la lógica de negocio y los datos.
- **Enrutamiento**: Gestiona la navegación entre vistas.
- **Directivas y Pipes**: Añaden funcionalidad al HTML (por ejemplo, `*ngFor`, `| titlecase`).
- **Templates y Estilos**: Definen la apariencia de los componentes.

Ejemplo en el proyecto:

- **Componentes**: `AppComponent`, `LoginComponent`, `MainLayoutComponent`, `MenuLateralComponent`, `ColeccionComponent`, `AltaComponent`, `BajaComponent`.
- **Servicios**: `PokemonService` (para manejar los datos de Pokémon).
- **Enrutamiento**: Definido en `main.ts` para navegar entre `/login`, `/main/coleccion`, `/main/alta`, y `/main/baja`.

---

3. Roles de los Componentes en el Proyecto

A continuación, detallo los componentes utilizados en tu proyecto `Pokemon App`, su propósito, y cómo se relacionan entre sí.

3.1 `AppComponent`

- **Rol**: Es el componente raíz de la aplicación. Actúa como un contenedor para el enrutamiento y no tiene lógica ni interfaz propia.

- **Ubicación**: `src/app/app.component.ts`

- **Propósito**: Renderiza el `<router-outlet>` principal, que permite que Angular renderice los componentes asociados a las rutas definidas.

- **Relaciones**:

  - Es el punto de entrada de la aplicación (definido en `main.ts` con `bootstrapApplication(AppComponent)`).

  - No interactúa directamente con otros componentes, pero sirve como contenedor para que el enrutamiento renderice `LoginComponent` o `MainLayoutComponent`.

Código:

```typescript
import { Component, OnInit } from '@angular/core';
import { RouterModule } from '@angular/router';

@Component({
  selector: 'app-root',
  standalone: true,
```

```
  imports: [RouterModule],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  ngOnInit() {
    console.log('AppComponent initialized');
  }
}
```

Template (`app.component.html`):

```html
<router-outlet></router-outlet>
```

Ejemplo:

- Cuando accedes a `http://localhost:4200`, el enrutamiento redirige a `/login`, y `AppComponent` renderiza `LoginComponent` dentro de `<router-outlet>`.

3.2 `LoginComponent`

- **Rol**: Gestiona la autenticación del usuario, mostrando un formulario de login para ingresar el nombre de usuario y la contraseña.

- **Ubicación**: `src/app/login/login.component.ts`

- **Propósito**: Permite al usuario iniciar sesión con las credenciales `emilio` y `12345`. Si las credenciales son correctas, establece `isLoggedIn` en `localStorage` y redirige a `/main`.

- **Relaciones**:

  - Es renderizado directamente por `AppComponent` cuando la ruta es `/login`.

  - Usa el servicio `Router` para redirigir a `/main` tras un login exitoso.

  - Interactúa con `AuthGuard` (indirectamente) porque `AuthGuard` verifica el estado `isLoggedIn` al intentar acceder a `/main`.

Código:

```typescript
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators, ReactiveFormsModule } from '@angular/forms';
import { Router } from '@angular/router';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatInputModule } from '@angular/material/input';
import { MatButtonModule } from '@angular/material/button';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-login',
  standalone: true,
  imports: [
    CommonModule,
    ReactiveFormsModule,
```

```
    MatFormFieldModule,
    MatInputModule,
    MatButtonModule
  ],
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  loginForm: FormGroup;
  errorMessage: string | null = null;

  private readonly validCredentials = {
    username: 'emilio',
    password: '12345'
  };

  constructor(private fb: FormBuilder, private router: Router) {
    this.loginForm = this.fb.group({
      username: ['', Validators.required],
      password: ['', Validators.required]
    });
  }

  ngOnInit() {
    console.log('LoginComponent initialized');
  }
```

```
  onSubmit() {
    this.errorMessage = null;

    if (this.loginForm.invalid) {
      if (this.loginForm.get('username')?.hasError('required') ||
this.loginForm.get('password')?.hasError('required')) {
        this.errorMessage = 'Por favor, completa todos los campos.';
      }
      return;
    }

    const { username, password } = this.loginForm.value;
    if (username === this.validCredentials.username && password ===
this.validCredentials.password) {
      localStorage.setItem('isLoggedIn', 'true');
      this.router.navigate(['/main']);
    } else {
      this.errorMessage = 'Nombre de usuario o contraseña incorrectos.';
    }
  }
}
```

Template (`login.component.html`):

```html
```

```
<div class="login-container">
  <h2>Iniciar Sesión</h2>
  <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
    <mat-form-field appearance="outline">
      <mat-label>Nombre de usuario</mat-label>
      <input matInput formControlName="username" required>
      <mat-error *ngIf="loginForm.get('username')?.hasError('required') &&
loginForm.get('username')?.touched">
        El nombre de usuario es obligatorio
      </mat-error>
    </mat-form-field>

    <mat-form-field appearance="outline">
      <mat-label>Contraseña</mat-label>
      <input matInput formControlName="password" type="password"
required>
      <mat-error *ngIf="loginForm.get('password')?.hasError('required') &&
loginForm.get('password')?.touched">
        La contraseña es obligatoria
      </mat-error>
    </mat-form-field>

    <button mat-raised-button color="primary" type="submit"
[disabled]="loginForm.invalid">Iniciar Sesión</button>
  </form>

  <div *ngIf="errorMessage" class="error-message">
```

{{ errorMessage }}

```

**Ejemplo**:

- Al cargar `http://localhost:4200`, el enrutamiento redirige a `/login`, y `LoginComponent` muestra un formulario. Al ingresar las credenciales correctas, el usuario es redirigido a `/main`.

---

#### **3.3 `MainLayoutComponent`**

- **Rol**: Es el componente contenedor para las vistas protegidas (`/main/coleccion`, `/main/alta`, `/main/baja`). Proporciona el layout principal con el `mat-toolbar`, el `mat-sidenav`, y el footer.

- **Ubicación**: `src/app/main-layout/main-layout.component.ts`

- **Propósito**: Define la estructura visual de las vistas protegidas, incluyendo la barra de navegación superior (`mat-toolbar`), el menú lateral (`mat-sidenav` con `MenuLateralComponent`), y el footer. También incluye un botón de cerrar sesión.

- **Relaciones**:

  - Es renderizado por `AppComponent` cuando la ruta es `/main`.

  - Contiene un `<router-outlet>` para renderizar las subrutas (`ColeccionComponent`, `AltaComponent`, `BajaComponent`).

  - Usa `MenuLateralComponent` para mostrar el menú lateral.

  - Interactúa con `Router` para manejar el logout y redirigir a `/login`.
```

Código:

```typescript
import { Component, OnInit } from '@angular/core';
import { CommonModule } from '@angular/common';
import { MatToolbarModule } from '@angular/material/toolbar';
import { MatSidenavModule } from '@angular/material/sidenav';
import { MatIconModule } from '@angular/material/icon';
import { Router, RouterModule } from '@angular/router';
import { MenuLateralComponent } from '../menu-lateral/menu-lateral.component';

@Component({
  selector: 'app-main-layout',
  standalone: true,
  imports: [
    CommonModule,
    MatToolbarModule,
    MatSidenavModule,
    MatIconModule,
    RouterModule,
    MenuLateralComponent
  ],
  templateUrl: './main-layout.component.html',
  styleUrls: ['./main-layout.component.css']
})
export class MainLayoutComponent implements OnInit {
```

```
  nombreAutor: string = 'Nombre del alumno';

  constructor(private router: Router) {
    console.log('MainLayoutComponent constructor');
  }

  ngOnInit() {
    console.log('MainLayoutComponent initialized');
  }

  logout() {
    localStorage.removeItem('isLoggedIn');
    this.router.navigate(['/login']);
  }
}
```

Template (`main-layout.component.html`):

```html
<header>
  <mat-toolbar>
    <mat-icon>catching_pokemon</mat-icon>
    <span class="toolbar-title">Pokemon App</span>
    <span class="spacer"></span>
    <button mat-icon-button (click)="logout()">
      <mat-icon>logout</mat-icon>


```

```
    </button>
  </mat-toolbar>
</header>

<mat-sidenav-container>
  <mat-sidenav mode="side" opened>
    <app-menu-lateral></app-menu-lateral>
  </mat-sidenav>
  <mat-sidenav-content>
    <router-outlet></router-outlet>
  </mat-sidenav-content>
</mat-sidenav-container>
<footer>
  <p>© 2025 - Emilio Garruta</p>
</footer>
```

Estilos (`main-layout.component.css`):

```css
.toolbar-title {
  margin-left: 10px;
  font-size: 24px;
  font-weight: bold;
}

footer {
```

```
  background-color: rgb(202, 249, 158);
}

header {
  background-color: rgb(202, 249, 158);
}

mat-toolbar {
  background-color: transparent;
  display: flex;
  align-items: center;
}

.spacer {
  flex: 1 1 auto;
}
```

Ejemplo:

- Cuando accedes a `/main` después de iniciar sesión, `MainLayoutComponent` renderiza el `mat-toolbar` con el título "Pokemon App", el `mat-sidenav` con el menú lateral, y el footer. El `<router-outlet>` renderiza `ColeccionComponent` (por defecto), `AltaComponent`, o `BajaComponent` según la subruta.

3.4 `MenuLateralComponent`

- **Rol**: Proporciona el menú lateral con enlaces de navegación a las vistas protegidas.

- **Ubicación**: `src/app/menu-lateral/menu-lateral.component.ts`

- **Propósito**: Muestra un menú de navegación con enlaces a `/main/coleccion`, `/main/alta`, y `/main/baja`.

- **Relaciones**:

  - Es usado por `MainLayoutComponent` dentro del `mat-sidenav`.

  - Usa `routerLink` para permitir la navegación entre las rutas.

Código:

```typescript
import { Component, OnInit } from '@angular/core';
import { MatListModule } from '@angular/material/list';
import { RouterModule } from '@angular/router';

@Component({
  selector: 'app-menu-lateral',
  standalone: true,
  imports: [MatListModule, RouterModule],
  templateUrl: './menu-lateral.component.html',
  styleUrls: ['./menu-lateral.component.css']
})
export class MenuLateralComponent implements OnInit {
  ngOnInit() {
    console.log('MenuLateralComponent initialized');
```

```
  }
}
```

Template (`menu-lateral.component.html`):

```html
<mat-nav-list>
  <a mat-list-item routerLink="/main/coleccion">Lista de Pokémon</a>
  <a mat-list-item routerLink="/main/alta">Agregar Pokémon</a>
  <a mat-list-item routerLink="/main/baja">Mi lista de Pokémon</a>
</mat-nav-list>
```

Ejemplo:

- En la vista `/main`, `MenuLateralComponent` se muestra dentro del `mat-sidenav` y permite al usuario navegar entre las vistas "Lista de Pokémon", "Agregar Pokémon", y "Mi lista de Pokémon".

3.5 `ColeccionComponent`

- **Rol**: Muestra una lista de Pokémon obtenidos desde PokéAPI, con la opción de agregarlos a la colección personalizada.

- **Ubicación**: `src/app/coleccion/coleccion.component.ts`

- **Propósito**: Obtiene una lista de Pokémon desde PokéAPI usando `PokemonService` y los muestra en acordeones (`mat-expansion-panel`). Permite al usuario agregar Pokémon a su colección.

- **Relaciones**:

  - Es renderizado por `MainLayoutComponent` dentro del `<router-outlet>` cuando la ruta es `/main/coleccion`.

  - Usa `PokemonService` para obtener los Pokémon y agregar Pokémon a la colección.

  - Interactúa con `MainLayoutComponent` (indirectamente) porque está dentro de su `<router-outlet>`.

Código:

```typescript
import { Component, OnInit } from '@angular/core';
import { CommonModule } from '@angular/common';
import { MatExpansionModule } from '@angular/material/expansion';
import { MatButtonModule } from '@angular/material/button';
import { PokemonService } from '../services/pokemon.service';
import { Pokemon } from '../models/pokemon.model';

@Component({
  selector: 'app-coleccion',
  standalone: true,
  imports: [CommonModule, MatExpansionModule, MatButtonModule],
  templateUrl: './coleccion.component.html',
  styleUrls: ['./coleccion.component.css']
})
export class ColeccionComponent implements OnInit {
  pokemons: Pokemon[] = [];
```

```
constructor(private pokemonService: PokemonService) {
  console.log('ColeccionComponent constructor');
}

ngOnInit() {
  console.log('ColeccionComponent initialized');
  this.loadPokemons();
}

loadPokemons() {
  this.pokemonService.getPokemons().subscribe(pokemons => {
    this.pokemons = pokemons;
    console.log('Pokemons loaded:', this.pokemons);
  });
}

addToMyCollection(pokemon: Pokemon) {
  this.pokemonService.addToCollection(pokemon).subscribe({
    next: () => {
      console.log(`${pokemon.name} agregado a la colección`);
    },
    error: (err) => {
      console.error('Error al agregar a la colección:', err);
    }
  });
}
```

}

```

**Template** (`coleccion.component.html`):

```html
<h2>Lista de Pokémon</h2>
<div class="accordion-container">
  <mat-accordion>
    <mat-expansion-panel *ngFor="let pokemon of pokemons">
      <mat-expansion-panel-header>
        <mat-panel-title>{{ pokemon.name | titlecase }}</mat-panel-title>
      </mat-expansion-panel-header>
      <div class="panel-content">
        <img [src]="pokemon.sprite" alt="{{ pokemon.name }}"
class="pokemon-sprite">
        <div class="pokemon-details">
          <p>ID: {{ pokemon.id }}</p>
          <p>Tipos: {{ pokemon.types.join(', ') }}</p>
          <button mat-raised-button color="primary"
(click)="addToMyCollection(pokemon)">Agregar a mi colección</button>
        </div>
      </div>
    </mat-expansion-panel>
  </mat-accordion>
</div>
```
```

Estilos (`coleccion.component.css`):

```css
.accordion-container {
  max-width: 600px;
  margin: 0 auto;
  padding: 0 16px;
}

.pokemon-sprite {
  max-width: 100px;
  margin: 10px 0;
}

mat-expansion-panel {
  margin-bottom: 8px;
  box-shadow: 0 2px 4px rgb(218, 224, 215);
  border-radius: 4px;
}

.panel-content {
  display: flex;
  align-items: center;
  gap: 16px;
}
```

```
.pokemon-details {
  flex: 1;
}

@media (max-width: 768px) {
  .accordion-container {
    max-width: 100%;
    padding: 0 8px;
  }

  .pokemon-sprite {
    max-width: 80px;
  }

  .panel-content {
    flex-direction: column;
    align-items: flex-start;
  }
}
```

Ejemplo:

- Al acceder a `/main/coleccion`, `ColeccionComponent` usa `PokemonService` para obtener una lista de Pokémon desde PokéAPI y los muestra en acordeones. El usuario puede expandir un acordeón para ver los detalles de un Pokémon y agregar un Pokémon a su colección haciendo clic en "Agregar a mi colección".

3.6 `AltaComponent`

- **Rol**: Permite al usuario agregar un Pokémon a su colección ingresando el ID del Pokémon.

- **Ubicación**: `src/app/alta/alta.component.ts`

- **Propósito**: Proporciona un formulario donde el usuario puede ingresar el ID de un Pokémon. Busca el Pokémon con ese ID usando `PokemonService` y lo agrega a la colección.

- **Relaciones**:

  - Es renderizado por `MainLayoutComponent` dentro del `<router-outlet>` cuando la ruta es `/main/alta`.

  - Usa `PokemonService` para buscar un Pokémon por ID y agregar Pokémon a la colección.

  - Usa `ChangeDetectorRef` para forzar la actualización de la interfaz después de restablecer el formulario.

Código:

```typescript
import { Component, OnInit, ChangeDetectorRef } from '@angular/core';
import { FormBuilder, FormGroup, Validators, ReactiveFormsModule } from '@angular/forms';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatInputModule } from '@angular/material/input';
import { MatButtonModule } from '@angular/material/button';
import { CommonModule } from '@angular/common';
import { PokemonService } from '../services/pokemon.service';
```

```
import { Pokemon } from '../models/pokemon.model';

@Component({
  selector: 'app-alta',
  standalone: true,
  imports: [
    ReactiveFormsModule,
    MatFormFieldModule,
    MatInputModule,
    MatButtonModule,
    CommonModule
  ],
  templateUrl: './alta.component.html',
  styleUrls: ['./alta.component.css']
})
export class AltaComponent implements OnInit {
  pokemonForm: FormGroup;
  feedbackMessage: string | null = null;

  constructor(
    private fb: FormBuilder,
    private pokemonService: PokemonService,
    private cdr: ChangeDetectorRef
  ) {
    this.pokemonForm = this.fb.group({
      id: ['', [Validators.required, Validators.min(1)]]
```

```
  });
 }

 ngOnInit() {
  console.log('AltaComponent initialized');
 }

 onSubmit() {
  this.feedbackMessage = null;

  if (this.pokemonForm.invalid) {
   if (this.pokemonForm.get('id')?.hasError('required')) {
    this.feedbackMessage = 'El ID es obligatorio';
   }
   return;
  }

  const id = this.pokemonForm.get('id')?.value;
  this.pokemonService.getPokemonDetails(id).subscribe({
   next: (pokemon: Pokemon) => {
    this.pokemonService.addToCollection(pokemon).subscribe({
     next: () => {
      this.feedbackMessage = `${pokemon.name.charAt(0).toUpperCase()
+ pokemon.name.slice(1)} agregado a la lista`;
      this.pokemonForm.reset();
      this.pokemonForm.get('id')?.setValue('');
```

```
          Object.keys(this.pokemonForm.controls).forEach(key => {
            this.pokemonForm.get(key)?.markAsUntouched();
            this.pokemonForm.get(key)?.markAsPristine();
          });
          this.cdr.detectChanges();
        },
        error: (err) => {
          console.error('Error al agregar a la colección:', err);
          this.feedbackMessage = 'Error al agregar el Pokémon a la colección.';
        }
      });
    },
    error: (err) => {
      console.error('Error al buscar Pokémon:', err);
      this.feedbackMessage = 'No se encontró un Pokémon con ese ID.';
    }
  });
 }
}
```

Template (`alta.component.html`):

```html
<h2>Agregar Pokémon a mi colección</h2>
<form [formGroup]="pokemonForm" (ngSubmit)="onSubmit()">
 <mat-form-field appearance="outline">
```

```
    <mat-label>ID del Pokémon</mat-label>
    <input matInput formControlName="id" type="number" required>
    <mat-error *ngIf="pokemonForm.get('id')?.hasError('required') &&
pokemonForm.get('id')?.touched && pokemonForm.get('id')?.dirty">
      El ID es obligatorio
    </mat-error>
  </mat-form-field>
  <button mat-raised-button color="primary" type="submit">Buscar y
Agregar</button>
</form>
<div *ngIf="feedbackMessage" class="feedback-message">
  {{ feedbackMessage }}
</div>
```

Estilos (`alta.component.css`):

```css
mat-form-field {
  width: 200px;
}

form {
  display: flex;
  flex-direction: column;
  align-items: flex-start;
  gap: 10px;
```

```
}

button {
  width: 200px;
}

.feedback-message {
  margin-top: 10px;
  padding: 10px;
  border-radius: 4px;
  font-size: 14px;
  color: #1B5E20;
  background-color: #C8E6C9;
  border: 1px solid #4CAF50;
  width: 200px;
}
```

Ejemplo:

- Al acceder a `/main/alta`, `AltaComponent` muestra un formulario donde el usuario puede ingresar un ID (por ejemplo, `6` para Charizard). Al enviar el formulario, `PokemonService` busca el Pokémon con ese ID y lo agrega a la colección, mostrando un mensaje de retroalimentación.

3.7 `BajaComponent`

- **Rol**: Muestra la colección personalizada de Pokémon del usuario y permite eliminar Pokémon de la colección.

- **Ubicación**: `src/app/baja/baja.component.ts`

- **Propósito**: Obtiene la colección personalizada del usuario desde `PokemonService` y la muestra en una lista. Permite eliminar Pokémon de la colección.

- **Relaciones**:

  - Es renderizado por `MainLayoutComponent` dentro del `<router-outlet>` cuando la ruta es `/main/baja`.

  - Usa `PokemonService` para obtener la colección (`getMyCollection`) y eliminar Pokémon (`removeFromCollection`).

Código:

```typescript
import { Component, OnInit } from '@angular/core';
import { CommonModule } from '@angular/common';
import { MatListModule } from '@angular/material/list';
import { MatIconModule } from '@angular/material/icon';
import { MatButtonModule } from '@angular/material/button';
import { PokemonService } from '../services/pokemon.service';
import { Pokemon } from '../models/pokemon.model';

@Component({
  selector: 'app-baja',
  standalone: true,
  imports: [CommonModule, MatListModule, MatIconModule, MatButtonModule],
```

```
  templateUrl: './baja.component.html',
  styleUrls: ['./baja.component.css']
})
export class BajaComponent implements OnInit {
  myCollection: Pokemon[] = [];

  constructor(private pokemonService: PokemonService) {
    console.log('BajaComponent constructor');
  }

  ngOnInit() {
    console.log('BajaComponent initialized');
    this.loadCollection();
  }

  loadCollection() {
    this.pokemonService.getMyCollection().subscribe({
      next: (collection) => {
        this.myCollection = collection;
        console.log('My collection loaded:', this.myCollection);
      },
      error: (err) => {
        console.error('Error al cargar la colección:', err);
      }
    });
  }
```

```
  removeFromCollection(id: number) {
    if (confirm(`¿Estás seguro de que quieres eliminar este Pokémon (ID: ${id})?`)) {
      this.pokemonService.removeFromCollection(id).subscribe({
        next: () => {
          this.loadCollection();
        },
        error: (err) => {
          console.error('Error al eliminar el Pokémon:', err);
        }
      });
    }
  }
}
```

Template (`baja.component.html`):

```html
<h2>Mi Lista de Pokémon</h2>
<div class="list-container">
  <mat-list>
    <mat-list-item *ngFor="let pokemon of myCollection">
      <img [src]="pokemon.sprite" alt="{{ pokemon.name }}" class="pokemon-sprite">
      <span>{{ pokemon.name | titlecase }} (ID: {{ pokemon.id }})</span>
```

```
      <button mat-icon-button (click)="removeFromCollection(pokemon.id)">
        <mat-icon>delete</mat-icon>
      </button>
    </mat-list-item>
  </mat-list>
</div>
```

Estilos (`baja.component.css`):

```css
.pokemon-sprite {
  width: 40px;
  height: 40px;
  margin-right: 10px;
  vertical-align: middle;
}

mat-list-item {
  display: flex;
  align-items: center;
}

span {
  flex-grow: 1;
}
```

Ejemplo:

- Al acceder a `/main/baja`, `BajaComponent` muestra la lista de Pokémon que el usuario ha agregado a su colección. El usuario puede eliminar un Pokémon haciendo clic en el ícono de basura, lo que actualiza la colección en `localStorage` mediante `PokemonService`.

3.8 `AuthGuard`

- **Rol**: Es un guardia de ruta que protege las rutas bajo `/main` para que solo los usuarios autenticados puedan acceder.

- **Ubicación**: `src/app/auth.guard.ts`

- **Propósito**: Verifica si el usuario está autenticado (basado en `localStorage.getItem('isLoggedIn') === 'true'`) y permite o deniega el acceso a las rutas protegidas.

- **Relaciones**:

  - Es usado por el enrutamiento en `main.ts` para proteger la ruta `/main`.

  - Interactúa con `Router` para redirigir a `/login` si el usuario no está autenticado.

Código:

```typescript
import { Injectable } from '@angular/core';
import { CanActivate, Router } from '@angular/router';
import { Observable, of } from 'rxjs';

@Injectable({
```

```
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor(private router: Router) {}

  canActivate(): Observable<boolean> | Promise<boolean> | boolean {
    console.log('AuthGuard: Checking if user is logged in');
    const isLoggedIn = localStorage.getItem('isLoggedIn') === 'true';
    if (isLoggedIn) {
      console.log('AuthGuard: User is logged in, allowing access');
      return true;
    } else {
      console.log('AuthGuard: User is not logged in, redirecting to /login');
      this.router.navigate(['/login']);
      return false;
    }
  }
}
```

Ejemplo:

- Si intentas acceder a `/main/coleccion` sin iniciar sesión, `AuthGuard` verifica `isLoggedIn`. Como no está definido en `localStorage`, redirige a `/login`. Después de iniciar sesión, `isLoggedIn` se establece en `true`, y `AuthGuard` permite el acceso.

3.9 `PokemonService`

- **Rol**: Es un servicio que maneja la lógica de negocio relacionada con los Pokémon, incluyendo la obtención de datos desde PokéAPI y la gestión de la colección personalizada.

- **Ubicación**: `src/app/services/pokemon.service.ts`

- **Propósito**: Proporciona métodos para obtener Pokémon, obtener detalles de un Pokémon, agregar Pokémon a la colección, eliminar Pokémon de la colección, y obtener la colección personalizada.

- **Relaciones**:

  - Es usado por `ColeccionComponent`, `AltaComponent`, y `BajaComponent` para manejar datos.

  - Usa `HttpClient` para hacer solicitudes HTTP a PokéAPI.

  - Usa `localStorage` para persistir la colección personalizada (`myCollection`).

Código:

```typescript
import { Injectable, PLATFORM_ID, Inject } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, map, of } from 'rxjs';
import { isPlatformBrowser } from '@angular/common';
import { Pokemon } from '../models/pokemon.model';

@Injectable({
  providedIn: 'root'
})
```

```
export class PokemonService {
  private apiUrl = 'https://pokeapi.co/api/v2/pokemon';
  private myCollection: Pokemon[] = [];

  constructor(
    private http: HttpClient,
    @Inject(PLATFORM_ID) private platformId: Object
  ) {
    if (isPlatformBrowser(this.platformId)) {
      this.loadFromLocalStorage();
    }
  }

  private loadFromLocalStorage(): void {
    if (!isPlatformBrowser(this.platformId)) {
      this.myCollection = [];
      return;
    }

    const storedCollection = localStorage.getItem('myPokemonCollection');
    if (storedCollection) {
      try {
        this.myCollection = JSON.parse(storedCollection);
      } catch (e) {
        console.error('Error al cargar la colección desde localStorage:', e);
        this.myCollection = [];
```

```
    }
   } else {
    this.myCollection = [];
   }
  }

  private saveToLocalStorage(): void {
   if (!isPlatformBrowser(this.platformId)) {
    return;
   }

   try {
    localStorage.setItem('myPokemonCollection',
JSON.stringify(this.myCollection));
   } catch (e) {
    console.error('Error al guardar la colección en localStorage:', e);
   }
  }

  getPokemons(limit: number = 20, offset: number = 0):
Observable<Pokemon[]> {
   console.log('PokemonService: getPokemons called with limit:', limit, 'offset:',
offset);
   return
this.http.get<any>(`${this.apiUrl}?limit=${limit}&offset=${offset}`).pipe(
    map(response => {
     console.log('PokemonService: getPokemons response:', response);
```

```
      return response.results.map((result: any, index: number) => ({
        id: offset + index + 1,
        name: result.name,
        types: [],
        sprite:
`https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/${offset + index + 1}.png`
      }));
    })
  );
 }

 getPokemonDetails(id: number): Observable<Pokemon> {
  console.log('PokemonService: getPokemonDetails called with id:', id);
  return this.http.get<any>(`${this.apiUrl}/${id}`).pipe(
   map(response => {
    console.log('PokemonService: getPokemonDetails response:', response);
    return {
     id: response.id,
     name: response.name,
     types: response.types.map((type: any) => type.type.name),
     sprite: response.sprites.front_default
    };
   })
  );
 }
```

```
  addToCollection(pokemon: Pokemon): Observable<void> {
    console.log('PokemonService: addToCollection called with pokemon:', pokemon);
    if (!this.myCollection.some(p => p.id === pokemon.id)) {
      this.myCollection.push(pokemon);
      this.saveToLocalStorage();
    }
    return of();
  }

  removeFromCollection(id: number): Observable<void> {
    console.log('PokemonService: removeFromCollection called with id:', id);
    this.myCollection = this.myCollection.filter(p => p.id !== id);
    this.saveToLocalStorage();
    return of();
  }

  getMyCollection(): Observable<Pokemon[]> {
    console.log('PokemonService: getMyCollection called');
    return of(this.myCollection);
  }
}
```

Ejemplo:

- En `ColeccionComponent`, `PokemonService.getPokemons()` obtiene una lista de Pokémon desde PokéAPI y la muestra.

- En `AltaComponent`, `PokemonService.getPokemonDetails(id)` busca un Pokémon por ID y lo agrega a la colección con `PokemonService.addToCollection(pokemon)`.

- En `BajaComponent`, `PokemonService.getMyCollection()` obtiene la colección personalizada, y `PokemonService.removeFromCollection(id)` elimina un Pokémon.

---

4. Relación entre los Componentes

Los componentes y servicios de tu aplicación están interconectados mediante el enrutamiento, la inyección de dependencias, y el flujo de datos. Aquí está un resumen de cómo se relacionan:

- **Enrutamiento**:

  - `AppComponent` es el componente raíz y contiene el `<router-outlet>` principal.

  - `LoginComponent` se renderiza en la ruta `/login`.

  - `MainLayoutComponent` se renderiza en la ruta `/main` (protegida por `AuthGuard`) y contiene un `<router-outlet>` secundario para las subrutas `/main/coleccion`, `/main/alta`, y `/main/baja`.

  - `ColeccionComponent`, `AltaComponent`, y `BajaComponent` son renderizados dentro del `<router-outlet>` de `MainLayoutComponent`.

- **Navegación**:

  - `MenuLateralComponent` (dentro de `MainLayoutComponent`) usa `routerLink` para navegar entre `/main/coleccion`, `/main/alta`, y `/main/baja`.

  - `LoginComponent` usa `router.navigate(['/main'])` para redirigir después de un login exitoso.

  - `AuthGuard` usa `router.navigate(['/login'])` para redirigir si el usuario no está autenticado.

  - `MainLayoutComponent` usa `router.navigate(['/login'])` para redirigir al cerrar sesión.

- **Flujo de datos**:

  - `PokemonService` es el encargado de manejar los datos de los Pokémon y la colección personalizada.

  - `ColeccionComponent` usa `PokemonService.getPokemons()` para obtener la lista de Pokémon y `PokemonService.addToCollection()` para agregar Pokémon a la colección.

  - `AltaComponent` usa `PokemonService.getPokemonDetails(id)` para buscar un Pokémon por ID y `PokemonService.addToCollection()` para agregarlo.

  - `BajaComponent` usa `PokemonService.getMyCollection()` para obtener la colección y `PokemonService.removeFromCollection(id)` para eliminar un Pokémon.

- **Autenticación**:

  - `LoginComponent` establece `isLoggedIn` en `localStorage` al iniciar sesión.

  - `AuthGuard` verifica `isLoggedIn` para permitir o denegar acceso a las rutas protegidas.

  - `MainLayoutComponent` usa `localStorage.removeItem('isLoggedIn')` para cerrar sesión.

Diagrama de Relaciones:

```
[AppComponent]
    ├── /login → [LoginComponent] → (inicia sesión) → localStorage.setItem('isLoggedIn', 'true')
    |                                   └── (redirección) → /main
    └── /main → [AuthGuard] → (verifica isLoggedIn)
            └── [MainLayoutComponent]
                ├── (contiene) [MenuLateralComponent] → (navegación) → /main/coleccion, /main/alta, /main/baja
                ├── /main/coleccion → [ColeccionComponent] → (usa) PokemonService.getPokemons(), PokemonService.addToCollection()
                ├── /main/alta → [AltaComponent] → (usa) PokemonService.getPokemonDetails(), PokemonService.addToCollection()
                └── /main/baja → [BajaComponent] → (usa) PokemonService.getMyCollection(), PokemonService.removeFromCollection()
```

---

5. Ejemplo Completo del Flujo de la Aplicación

Escenario: El usuario inicia la aplicación, inicia sesión, navega por las vistas, agrega y elimina Pokémon, y cierra sesión.

1. **Inicio de la aplicación**:

- El usuario accede a `http://localhost:4200`.

- El enrutamiento en `main.ts` redirige a `/login` (`{ path: '', redirectTo: '/login', pathMatch: 'full' }`).

- `AppComponent` renderiza `LoginComponent` dentro de `<router-outlet>`.

2. **Inicio de sesión**:

- El usuario ingresa las credenciales `emilio` y `12345` en el formulario de `LoginComponent`.

- `LoginComponent` verifica las credenciales y establece `localStorage.setItem('isLoggedIn', 'true')`.

- `LoginComponent` redirige a `/main` (`this.router.navigate(['/main'])`).

3. **Acceso a `/main`**:

- `AuthGuard` verifica `isLoggedIn` y permite el acceso porque `isLoggedIn` es `true`.

- `MainLayoutComponent` se renderiza, mostrando el `mat-toolbar`, el `mat-sidenav` (con `MenuLateralComponent`), y el footer.

- La subruta `/main` redirige a `/main/coleccion` (`{ path: '', redirectTo: 'coleccion', pathMatch: 'full' }`).

- `ColeccionComponent` se renderiza dentro del `<router-outlet>` de `MainLayoutComponent`.

4. **Listado de Pokémon**:

- `ColeccionComponent` usa `PokemonService.getPokemons()` para obtener una lista de Pokémon desde PokéAPI.

- Los Pokémon se muestran en acordeones (`mat-expansion-panel`), y el usuario puede agregar un Pokémon a su colección haciendo clic en "Agregar a mi colección".

- `PokemonService.addToCollection(pokemon)` agrega el Pokémon a `myCollection` y lo guarda en `localStorage`.

5. **Navegación a `/main/alta`**:

- El usuario hace clic en "Agregar Pokémon" en el menú lateral (`MenuLateralComponent`).

- `AltaComponent` se renderiza, mostrando un formulario para ingresar el ID de un Pokémon.

- El usuario ingresa un ID (por ejemplo, `6` para Charizard) y envía el formulario.

- `PokemonService.getPokemonDetails(6)` busca el Pokémon y `PokemonService.addToCollection()` lo agrega a la colección.

6. **Navegación a `/main/baja`**:

- El usuario hace clic en "Mi lista de Pokémon" en el menú lateral.

- `BajaComponent` se renderiza, mostrando la colección personalizada obtenida con `PokemonService.getMyCollection()`.

- El usuario elimina un Pokémon haciendo clic en el ícono de basura, lo que llama a `PokemonService.removeFromCollection(id)`.

7. **Cerrar sesión**:

- El usuario hace clic en el botón de cerrar sesión en el `mat-toolbar` de `MainLayoutComponent`.

- `MainLayoutComponent` ejecuta `logout()`, que elimina `isLoggedIn` de `localStorage` y redirige a `/login`.

---

6. Conclusión

Este proyecto, `Pokemon App`, es un excelente ejemplo de cómo Angular permite construir aplicaciones SPA modulares y bien estructuradas. A continuación, un resumen de los conceptos clave aplicados:

- **Componentes**: Cada componente (`LoginComponent`, `MainLayoutComponent`, etc.) tiene una responsabilidad específica y es reutilizable.

- **Enrutamiento**: El enrutamiento permite navegar entre vistas sin recargar la página, con rutas protegidas por `AuthGuard`.

- **Servicios**: `PokemonService` centraliza la lógica de negocio y los datos, permitiendo que los componentes compartan datos de manera eficiente.

- **Inyección de dependencias**: Los componentes inyectan servicios (`PokemonService`, `Router`) para acceder a funcionalidades comunes.

- **Data Binding y Directivas**: Los templates usan directivas como `*ngFor` y `| titlecase` para mostrar datos dinámicamente.

Recomendaciones:

- **Seguridad**: Implementar un sistema de autenticación más robusto (por ejemplo, con JWT y un backend real).

- **Reactividad**: Usar `BehaviorSubject` en `PokemonService` para que los cambios en `myCollection` se reflejen automáticamente en todos los componentes.

- **Diseño**: Mejorar el diseño de la página de login con un fondo, un logo, o animaciones.

Este manual debería proporcionarte una base sólida para entender Angular y cómo se aplica en tu proyecto `Pokemon App`. Si deseas profundizar en

algún aspecto o implementar alguna mejora, ¡estaré encantado de ayudarte!