Escuela: *"Instituto Tecnológico de Oaxaca"*

Nombre del alumno: Andrea Quetzali Ordoñez Pérez

Nombre del maestro: Adelina Martínez Nieto

Materia: Programación web

Tarea "Práctica de Consumo de APIs"

2 Unidad

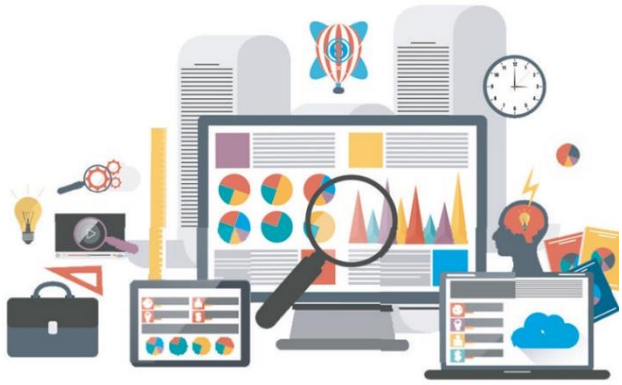
7° Semestre

Grupo: 7SA

Horario: 11:00 – 12:00 pm

Carrera: Ingeniería en Sistemas Computacionales

Fecha: 14/10/2024



Contenido

Preguntas

¿Qué hace el método getUsers en este servicio?

Se encarga de realizar una solicitud a HTTP GET, a la URL https://api.escuelajs.co/api/v1/users, por lo que se obtendrá una lista de usuarios de esa API.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class UserService {
  private apiUrl = 'https://api.escuelajs.co/api/v1/users';
  constructor(private http: HttpClient) { }
  getUsers(): Observable<any[]> {
    return this.http.get<any[]>(this.apiUrl);
  }
}
```

¿Por qué es necesario importar HttClientModule?

En base al archivo app.module.ts, debe de importar a HttpClientModule, ya que UserService usa HttpClient que es una clase proporcionada que permite realizar solicitudes HTTP (GET, POST, PUT, DELETE, etc.) a servidores remotos desde la aplicación. Esto es especialmente útil para interactuar con APIs o servidores backend.

```
import { Component, OnInit, Input } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HttpClientModule } from '@angular/common/http';
import { UserService } from '../../services/user.service';

@Component({
  selector: 'app-user-list',
  standalone: true,
  imports: [CommonModule, HttpClientModule],
  providers: [UserService],
  templateUrl: './user-list.component.html',
  styleUrls: ['./user-list.component.css']
})
export class UserListComponent implements OnInit {
  users: any[] = [];
```

```
  constructor(private userService: UserService) { }
  ngOnInit(): void {
    this.userService.getUsers().subscribe(data => {
      this.users = data;
    });
  }
}
```

¿Qué función cumple el método ngOnInit en el componente UserListComponent?
Usa para:

- ✓ **Cargar los usuarios al iniciar el componente**: Ya que este se inicializa y ngOnInit llama al método getUsers del servicio UserService.
- ✓ **Suscribirse a la respuesta del servicio**: getUsers devuelve un Observable que representa la respuesta asincrónica de la solicitud HTTP. Dentro de ngOnInit, se realiza una suscripción (subscribe) para recibir los datos cuando estén disponibles.
- ✓ **Asignar los datos a la propiedad users**: Una vez que los datos se reciben en la respuesta, se asignan a la propiedad users del componente. Esto permite que users esté disponible para el uso en la plantilla (user-list.component.html), donde se muestren los usuarios en una lista o tabla.

```
✓ import { Component, OnInit, Input } from '@angular/core';
✓ import { CommonModule } from '@angular/common';
✓ import { HttpClientModule } from '@angular/common/http';
✓ import { UserService } from '../../services/user.service';
✓
✓ @Component({
✓   selector: 'app-user-list',
✓   standalone: true,
✓   imports: [CommonModule, HttpClientModule],
✓   providers: [UserService],
✓   templateUrl: './user-list.component.html',
✓   styleUrls: ['./user-list.component.css']
✓ })
✓ export class UserListComponent implements OnInit {
✓   users: any[] = [];
✓   constructor(private userService: UserService) { }
✓   ngOnInit(): void {
✓     this.userService.getUsers().subscribe(data => {
✓       this.users = data;
✓     });
✓   }
✓ }
✓
```

¿Para qué sirve el bucle *ngFor en Angular? Explica cómo se utiliza en este ejemplo

El *ngFor, sirve para tomar una lista de datos como array, y por cada elemento de esa lista, genera una copia del elemento HTML donde está definido. Cada copia recibe acceso a un elemento del array, lo que permite acceder y mostrar sus propiedades en la plantilla, por lo que, en este ejemplo, el *ngFor se utiliza en el componente HTML de UserListComponent para mostrar una lista de usuarios en una tabla, asi mismo permite crear dinámicamente una fila en la tabla para cada usuario en la lista users, de manera que todos los usuarios se muestran en una tabla sin necesidad de escribir manualmente cada fila.

```
<div class="container mt-5">
    <h2>User List</h2>
    <table class="table table-bordered table-striped">
        <thead>
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Email</th>
                <th>Role</th>
            </tr>
        </thead>
        <tbody>
            <tr *ngFor="let user of users">
                <td>{{ user.id }}</td>
                <td>{{ user.name }}</td>
                <td>{{ user.email }}</td>
                <td>{{ user.role }}</td>
            </tr>
        </tbody>
    </table>
</div>
```

Ejecución de proyecto

Lista de usuarios

IDENTIFICACIÓN	Nombre	Correo electrónico	Rol
1	Jhon	john@mail.com	cliente
2	Maria	maria@mail.com	cliente
3	Admin	admin@mail.com	Admin
5	Phillip	phillip1234@gmail.com	cliente
6	Phillip	phillip1234@gmail.com	cliente
7	Phillip	phillip1234@gmail.com	cliente
9	Phillip	phillip1234@gmail.com	cliente
11	Allan	allan123@gmail.com	cliente
12	Phillip	phillip1234@gmail.com	cliente
13	Johnathan Walker	argentum@gmail.com	cliente
14	Nicolas	nico@gmail.com	cliente
15	Soleh Sunedi	solehsunedi@gmail.com	cliente
16	a.C	kieuduytung3@gmail.com	cliente
17	John	john@mail.com	cliente
18	actualizar	update@gmail.com	cliente
19	Nicolas	nico@gmail.com	cliente
20	Jhon	john@mail.com	cliente
21	Nicolas	nico@gmail.com	cliente
22	actualizar	update@gmail.com	cliente
23	Nicolas	nico@gmail.com	cliente
24	MELIKA	john@mail.com	cliente
25	Depi	depipermana87@gmail.com	cliente
26	NewName	ddsada@mail.com	cliente
27	olrosaZz	olrosa@mail.com	cliente
28	wqew	we@mail.ru	cliente
29	Pammi	balor@gmail.com	cliente
30	Pammi	balor@gmail.com	cliente
31	Pammi	balor2@gmail.com	cliente
32	Nosotros	wq@mail.ru	cliente
33	wqewq	qw@mail.ru	cliente
34	sadsad	asd@mail.ru	cliente
35	wwqeq	wq@mail.ru	cliente
36	wwqeq	wq@mail.ru	cliente
37	Nicolas	nico@gmail.com	cliente
39	qwewq	qwe@mail.ru	cliente
40	Arif Shumon	AS@gmail.com	cliente
41	Tsharp	test@gmail.com	cliente
42	Información de la prueba	test@gmail.com	cliente
43	Información de la prueba	test@gmail.com	cliente
44	Datos de Tsharp	data@gmail.com	cliente
45	Ibrahim	ibrahim111@gmail.com	Admin
46	Ibrahim	ibrahim@gmail.com	cliente
47	Mahesh	mahesh@gmail.com	cliente
48	asdasdasd	wqe@mail.ru	cliente
49	Jhon	john@mail.com	cliente
50	wqeqwe	wqe@mail.ru	cliente
51	wqewqe	asd@mail.ru	cliente
52	wqewqe	asd@mail.ru	cliente
53	Goutham	goutham@gmail.com	cliente
54	wqeqw	qwe@mail.ru	cliente
55	wqe	asd@mail.ru	cliente
56	Антон	asd@mail.ru	cliente
57	Nicolas	nico@gmail.com	cliente
58	Manoj Kumar	manoj@gmail.com	cliente

¿Qué ventajas tiene el uso de servicios en Angular para el consumo de APIs?

En que conlleva a una estructura de organización, reutilización y mantenimiento de codigo, por ejemplo, al usar servicios se separa la lógica de negocio y la lógica de presentación, ya que esto permite que los componentes se enfoquen únicamente en manejar la interfaz de usuario y los datos que reciben del servicio. Otra ventaja pordria ser que al usar servicios para consumir APIs, ya que la lógica de negocio queda centralizada en un solo lugar. Incluso Angular permite la inyección de dependencias en servicios, lo cual facilita el manejo de dependencias como HttpClient para realizar las peticiones HTTP. Esto hace que los servicios sean más modulares y fáciles de probar. Incluso los servicios permiten implementar estrategias de almacenamiento en caché para minimizar las llamadas a la API. Por ejemplo, puedes almacenar temporalmente los datos obtenidos de una API y reutilizarlos si los mismos datos son necesarios en otras partes de la aplicación. Y en la parte de servicios se puede implementar estrategias de seguridad, como la inclusión de tokens de autenticación en los encabezados de las solicitudes, sin exponer estos detalles en los componentes.

¿Por qué es importante separar la lógica de negocio de la lógica de presentación?

Permite que el codigo sea más escalable, limpio, escalable y fácil de probar, ya que ambas lógicas cumplen con propósitos diferentes, como: la lógica de negocio se encarga de las reglas y operaciones fundamentales, mientras que la lógica de presentación se centra en cómo se muestran esos datos al usuario. Al separarlas, el código de cada una se vuelve más fácil de entender y mantener, sin entrelazar tareas distintas en un solo lugar. Además, al tener la lógica de negocio separada, se puede probar de manera aislada sin que preocuparse de la interfaz de usuario. Esto permite realizar pruebas unitarias más precisas y facilita detectar y corregir errores en las reglas de negocio.

¿Qué otros tipos de datos o APIs podrías integrar en un proyecto como este?

- ✓ Auth0 o Firebase Authentication para autenticación segura y autorización de usuarios.
- ✓ API de gestión de usuarios si se necesita manejar roles, permisos o perfiles de usuarios personalizados.
- ✓ APIs de productos como Amazon Product Advertising o eBay para obtener información de productos, precios y reseñas.
- ✓ YouTube Data API o Vimeo API para integrar videos en la aplicación.
- ✓ Spotify o Apple Music API para incluir música o listas de reproducción.
- ✓ Firebase Cloud Messaging (FCM) para enviar notificaciones push a los usuarios.
- ✓ Twilio para funcionalidades de mensajería SMS o llamadas de voz si necesitas un canal de comunicación adicional.