**<div class="container py-5"> <div class="row justify-content-center"> <div class="col-lg-8"> <div class="card border-0 shadow-sm"> <div class="card-body p-4"> <!-- Header del perfil --> <div class="d-flex flex-column align-items-center text-center mb-4"> <img [src]="user.avatar" alt="Avatar" class="rounded-circle mb-3" width="120" height="120"> <h2 class="mb-1">{{ user.name }}</h2> <p class="text-muted mb-3">{{ user.email }}</p> <span class="badge bg-primary rounded-pill px-3">Miembro desde {{ user.joinDate }}</span> </div> <!-- Sección de información --> <div class="row g-3 mb-4"> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-person-lines-fill me-2 text-primary"></i> Información Personal </h3> <ul class="list-unstyled"> <li class="mb-2"> <strong>Nombre:</strong> {{ user.name }} </li> <li class="mb-2"> <strong>Email:</strong> {{ user.email }} </li> <li class="mb-2"> <strong>Bio:</strong> {{ user.bio }} </li> </ul> </div> </div> </div> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-bar-chart-line-fill me-2 text-primary"></i> Estadísticas </h3> <div class="d-flex flex-wrap gap-3"> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksCompleted }}</div> <small class="text-muted">Tareas completadas</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.projects }}</div> <small class="text-muted">Proyectos</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.lastActive }}</div> <small class="text-muted">Última actividad</small> </div> </div> </div> </div> </div> </div> <!-- Acciones --> <div class="d-flex justify-content-center gap-3"> <button class="btn btn-primary px-4"> <i class="bi bi-pencil-square me-2"></i> Editar Perfil </button> <button class="btn btn-outline-danger px-4" (click)="logout()"> <i class="bi bi-box-arrow-right me-2"></i> Cerrar Sesión </button> </div> </div> </div> </div> </div> </div> import { Component } from '@angular/core'; import { AuthService } from '../../shared/services/auth.service'; @Component({ selector: 'app-profile', standalone: false, templateUrl: './profile.component.html', styleUrls: ['./profile.component.scss'] }) export class ProfileComponent { user = { name: 'Alex García', email: 'alex@ejemplo.com', joinDate: '15 Enero 2023', avatar: 'assets/images/avatar-placeholder.png', // Puedes usar una imagen por defecto bio: 'Desarrollador frontend con experiencia en Angular y diseño de interfaces.', stats: { tasksCompleted: 42, projects: 5, lastActive: 'Hace 2 horas' } }; constructor( private authService: AuthService ) {} logout(): void { this.authService.logout(); // Llama al método logout de tu AuthService } } @use 'bootstrap/scss/bootstrap' as bs; .profile-card { .avatar-img { object-fit: cover; border: 3px solid bs.$primary; } .stat-card { transition: transform 0.3s ease; &:hover { transform: translateY(-3px); box-shadow: bs.$box-shadow-sm; } } .btn-edit { padding: 0.5rem 1.5rem; font-weight: 500; &:hover { box-shadow: 0 0.5rem 1rem rgba(bs.$primary, 0.15); } } } Actualmente esta es mi página de perfil. Es una página estática, sin problema. Bueno, el tema es que en mi BBDD la única información que hay es la ID, el nombre y el email. Luego hay categorías y tareas de las que se puede llevar la cuenta (tanto pendientes como completadas en el caso de las tareas) con segun que consultas. Pues bien, lo que quiero es eso, justamente. Cambiame el esquema para que siga siendo bonito pero se adapte a la información de que dispongo. Avatar no hay porque no hay campo ni he configurado nada para un avatar. Y así. Tampoco tengo timestamps... Otra cosa usa la ID para decir "eres el usuario número tal" siendo tal la id. Simplemente por decorar y ocupar un poco más.**

Aquí tienes una versión modificada de tu página de perfil que se adapta a los datos que realmente tienes disponibles (ID, nombre, email) y aprovecha las estadísticas de tareas y categorías:

html

<div class="container py-5">

<div class="row justify-content-center">

<div class="col-lg-8">

<div class="card border-0 shadow-sm">

<div class="card-body p-4">

<!-- Header del perfil -->

<div class="d-flex flex-column align-items-center text-center mb-4">

<div class="rounded-circle bg-primary bg-opacity-10 d-flex align-items-center justify-content-center mb-3"

style="width: 120px; height: 120px;">

<i class="bi bi-person-fill text-primary" style="font-size: 3rem;"></i>

</div>

<h2 class="mb-1">{{ user.name }}</h2>

<p class="text-muted mb-3">{{ user.email }}</p>

<span class="badge bg-primary rounded-pill px-3">Usuario #{{ user.id }}</span>

</div>

<!-- Sección de información -->

<div class="row g-3 mb-4">

<div class="col-md-6">

<div class="card h-100 border-0 bg-light">

<div class="card-body">

<h3 class="h5 mb-3">

<i class="bi bi-person-lines-fill me-2 text-primary"></i>

Información Básica

</h3>

<ul class="list-unstyled">

<li class="mb-2">

<strong>ID de usuario:</strong> #{{ user.id }}

</li>

<li class="mb-2">

<strong>Nombre:</strong> {{ user.name }}

</li>

<li class="mb-2">

<strong>Email:</strong> {{ user.email }}

</li>

</ul>

</div>

</div>

</div>

<div class="col-md-6">

<div class="card h-100 border-0 bg-light">

<div class="card-body">

<h3 class="h5 mb-3">

<i class="bi bi-bar-chart-line-fill me-2 text-primary"></i>

Tu Productividad

</h3>

<div class="d-flex flex-wrap gap-3">

<div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1">

<div class="h4 text-primary mb-0">{{ user.stats.tasksCompleted }}</div>

<small class="text-muted">Tareas completadas</small>

</div>

<div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1">

<div class="h4 text-primary mb-0">{{ user.stats.tasksPending }}</div>

<small class="text-muted">Tareas pendientes</small>

</div>

<div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1">

<div class="h4 text-primary mb-0">{{ user.stats.categories }}</div>

<small class="text-muted">Categorías</small>

</div>

</div>

</div>

</div>

</div>

</div>

<!-- Acciones -->

<div class="d-flex justify-content-center gap-3">

<button class="btn btn-primary px-4">

<i class="bi bi-pencil-square me-2"></i>

Editar Perfil

</button>

<button class="btn btn-outline-danger px-4" (click)="logout()">

<i class="bi bi-box-arrow-right me-2"></i>

Cerrar Sesión

</button>

</div>

</div>

</div>

</div>

</div>

</div>

Y el componente TypeScript actualizado:

typescript

import { Component } from '@angular/core';

import { AuthService } from '../../shared/services/auth.service';

import { TaskService } from '../../shared/services/task.service';

import { CategoryService } from '../../shared/services/category.service';

@Component({

selector: 'app-profile',

templateUrl: './profile.component.html',

styleUrls: ['./profile.component.scss']

})

export class ProfileComponent {

user = {

id: 0,

name: '',

email: '',

stats: {

tasksCompleted: 0,

tasksPending: 0,

categories: 0

}

};

constructor(

private authService: AuthService,

private taskService: TaskService,

private categoryService: CategoryService

) {}

ngOnInit(): void {

// Aquí deberías cargar los datos reales del usuario

this.loadUserData();

this.loadStatistics();

}

loadUserData(): void {

// Suponiendo que tienes un método para obtener los datos del usuario logueado

const currentUser = this.authService.getCurrentUser();

if (currentUser) {

this.user.id = currentUser.id;

this.user.name = currentUser.name;

this.user.email = currentUser.email;

}

}

loadStatistics(): void {

// Obtener estadísticas de tareas

this.taskService.getTasksCount().subscribe(counts => {

this.user.stats.tasksCompleted = counts.completed;

this.user.stats.tasksPending = counts.pending;

});

// Obtener estadísticas de categorías

this.categoryService.getCategoriesCount().subscribe(count => {

this.user.stats.categories = count;

});

}

logout(): void {

this.authService.logout();

}

}

Cambios realizados:

1. Eliminé el avatar y lo reemplacé con un icono de persona sobre fondo claro
2. Añadí el ID de usuario como identificador único ("Usuario #X")
3. Simplifiqué la información personal mostrando solo ID, nombre y email
4. Actualicé las estadísticas para mostrar:
   * Tareas completadas
   * Tareas pendientes
   * Número de categorías
5. Eliminé los campos que no tienes (bio, fecha de registro, última actividad)
6. El diseño sigue siendo limpio y atractivo, pero ahora muestra solo información real

Para que esto funcione necesitarás implementar los servicios correspondientes para obtener:

* El conteo de tareas completadas/pendientes (TaskService)
* El número de categorías (CategoryService)
* Los datos del usuario actual (AuthService)

El estilo SCSS puede mantenerse igual ya que las clases CSS no han cambiado significativamente.

**const currentUser = this.authService.getToken(); if (currentUser) { this.user.id = currentUser.id; this.user.name = currentUser.name; this.user.email = currentUser.email; } Esto permite acceder al payload del token?**

Depende de cómo hayas implementado tu AuthService, pero generalmente **NO**, acceder directamente al token no te da automáticamente acceso al payload decodificado. Necesitas decodificar el token JWT para extraer la información del usuario. Aquí te explico cómo hacerlo correctamente:

**Solución completa:**

1. **Primero, instala la librería para decodificar JWT**:

bash

npm install jwt-decode

1. **Modifica tu método en el componente**:

typescript

import { jwtDecode } from 'jwt-decode'; // Importa la función de decodificación

// ...

loadUserData(): void {

const token = this.authService.getToken();

if (token) {

try {

const decoded: any = jwtDecode(token); // Decodifica el token

this.user.id = decoded.userId || decoded.sub; // Depende de tu backend

this.user.name = decoded.name;

this.user.email = decoded.email;

} catch (error) {

console.error('Error al decodificar el token:', error);

}

}

}

**Alternativa mejor tipada (recomendada):**

Crea una interfaz para el payload:

typescript

interface JwtPayload {

userId: number;

name: string;

email: string;

// otras propiedades que incluya tu token

}

// Luego úsala al decodificar:

const decoded = jwtDecode<JwtPayload>(token);

this.user.id = decoded.userId;

// ...

**Si usas Angular 15+ con ESModules:**

typescript

import { jwtDecode } from 'jwt-decode';

// ... en tu método

const decoded = jwtDecode<JwtPayload>(token);

**¿Por qué no funciona directamente con getToken()?**

* getToken() normalmente solo devuelve el token como string ("eyJhbGciOiJIUzI1NiIsIn..."
* El token tiene 3 partes (header, payload, signature) codificadas en Base64
* Necesitas decodificar el payload para acceder a los datos del usuario

**Ejemplo de token decodificado:**

json

{

"userId": 123,

"name": "Alex García",

"email": "alex@ejemplo.com",

"iat": 1710000000,

"exp": 1710086400

}

**Consejo adicional:**

Si usas mucho esta funcionalidad, considera añadir un método getCurrentUser() en tu AuthService que haga la decodificación internamente y devuelva ya el objeto con los datos del usuario.

**Entonces cada vez que ejecute la función getToken voy a tener que ejecutar todo esto para decodificar el token? No sería mejor decodificarlo en otro sitio y no en cada funciójn? Por ejemplo crear una función en el propio authservice?**

¡Exactamente! Tienes toda la razón. **No es eficiente** decodificar el token cada vez que necesites acceder a los datos del usuario. La mejor práctica es **centralizar la lógica de decodificación en tu AuthService** y exponer métodos para acceder a los datos ya decodificados.

**Implementación óptima en AuthService:**

1. **Primero, añade métodos para manejar el token decodificado**:

typescript

// auth.service.ts

import { Injectable } from '@angular/core';

import { jwtDecode } from 'jwt-decode';

@Injectable({

providedIn: 'root'

})

export class AuthService {

private decodedToken: any = null;

// Almacena el token y decodifícalo

setToken(token: string): void {

localStorage.setItem('auth\_token', token);

this.decodedToken = jwtDecode(token);

}

// Obtiene el token crudo

getToken(): string | null {

return localStorage.getItem('auth\_token');

}

// Devuelve el token ya decodificado

getDecodedToken(): any {

if (!this.decodedToken && this.getToken()) {

this.decodedToken = jwtDecode(this.getToken()!);

}

return this.decodedToken;

}

// Métodos específicos para datos comunes

getCurrentUserId(): number | null {

const token = this.getDecodedToken();

return token?.userId || token?.sub || null;

}

getCurrentUserName(): string | null {

const token = this.getDecodedToken();

return token?.name || null;

}

getCurrentUserEmail(): string | null {

const token = this.getDecodedToken();

return token?.email || null;

}

// Limpia todo al cerrar sesión

logout(): void {

localStorage.removeItem('auth\_token');

this.decodedToken = null;

}

}

1. **Luego en tu componente, úsalo así**:

typescript

// profile.component.ts

loadUserData(): void {

this.user.id = this.authService.getCurrentUserId() || 0;

this.user.name = this.authService.getCurrentUserName() || 'Nombre no disponible';

this.user.email = this.authService.getCurrentUserEmail() || 'Email no disponible';

}

**Ventajas de este enfoque:**

1. **Single Source of Truth**: La decodificación ocurre solo cuando es necesario y se almacena en memoria.
2. **Mejor performance**: Evitas decodificar el token múltiples veces.
3. **Código más limpio**: Los componentes no necesitan saber cómo se decodifica el token.
4. **Tipado seguro**: Puedes añadir interfaces para el token:

typescript

interface JwtPayload {

userId: number;

name: string;

email: string;

iat: number;

exp: number;

}

// Luego en el AuthService:

getDecodedToken(): JwtPayload | null {

// ... misma implementación pero con tipo

}

**Caso de uso típico:**

1. **Al iniciar sesión**:

typescript

login(credentials: any): Observable<void> {

return this.http.post('/api/login', credentials).pipe(

tap((response: any) => {

this.setToken(response.token); // Decodifica y almacena

})

);

}

1. **Al refrescar la página**:

typescript

// En tu app.component.ts o similar

ngOnInit() {

if (this.authService.getToken()) {

// El AuthService automáticamente decodificará el token al primer acceso

const userId = this.authService.getCurrentUserId();

// ...

}

}

**¿Por qué no usar un Subject/BehaviorSubject?**

Si quieres ir un paso más allá, puedes usar un BehaviorSubject para notificar cambios:

typescript

// En AuthService

private userData = new BehaviorSubject<JwtPayload | null>(null);

// Modifica setToken:

setToken(token: string): void {

localStorage.setItem('auth\_token', token);

this.decodedToken = jwtDecode(token);

this.userData.next(this.decodedToken);

}

// Expón como Observable

getUserData(): Observable<JwtPayload | null> {

return this.userData.asObservable();

}

Así los componentes pueden suscribirse a los cambios del usuario automáticamente. 🚀

**loadUserData(): void { // Suponiendo que tienes un método para obtener los datos del usuario logueado const cut = this.authService.getDecodedToken(); if (cut) { this.user.id = cut.sub; this.user.name = cut.name; this.user.email = cut.email; } } getToken(): string | null { const token = localStorage.getItem(this.TOKEN\_KEY); if (!token || this.isTokenExpired(token)) { this.logout(); return null; } return token } getDecodedToken(): any { const token = this.getToken() if (token == null) { return null } return jwtDecode(token); } Esta lógica va bien, no?**

Sí, la lógica que has implementado es correcta y sigue buenas prácticas. Vamos a analizarla y sugerir algunas pequeñas mejoras:

**Análisis de tu implementación actual:**

1. **getToken()**:
   * Verifica si el token existe y no está expirado
   * Hace logout automático si el token no es válido ✅
   * Retorna null en casos de error ✅
2. **getDecodedToken()**:
   * Reutiliza getToken() para obtener el token crudo ✅
   * Maneja el caso null adecuadamente ✅
   * Decodifica solo cuando es necesario ✅
3. **loadUserData()**:
   * Usa el método getDecodedToken() del servicio ✅
   * Asigna correctamente las propiedades del usuario ✅

**Mejoras sugeridas:**

1. **Tipado fuerte** (Recomendado):

typescript

interface DecodedToken {

sub: string | number; // ID del usuario

name: string;

email: string;

exp: number;

// ... otras propiedades que incluya tu token

}

getDecodedToken(): DecodedToken | null {

const token = this.getToken();

return token ? jwtDecode<DecodedToken>(token) : null;

}

1. **Manejo de propiedades opcionales**:

typescript

loadUserData(): void {

const token = this.authService.getDecodedToken();

this.user = {

id: token?.sub ?? 0,

name: token?.name ?? 'Invitado',

email: token?.email ?? '',

stats: this.user.stats // Mantiene las estadísticas existentes

};

}

1. **Cache del token decodificado** (Optimización):

typescript

private decodedToken: DecodedToken | null = null;

getDecodedToken(): DecodedToken | null {

if (!this.decodedToken) {

const token = this.getToken();

this.decodedToken = token ? jwtDecode<DecodedToken>(token) : null;

}

return this.decodedToken;

}

// Al hacer logout:

logout(): void {

this.decodedToken = null;

// ... resto de la lógica

}

1. **Validación adicional** (Opcional):

typescript

getDecodedToken(): DecodedToken | null {

const token = this.getToken();

if (!token) return null;

try {

const decoded = jwtDecode<DecodedToken>(token);

// Verifica que tenga las propiedades mínimas

if (!decoded.sub || !decoded.email) {

throw new Error('Token inválido');

}

return decoded;

} catch (error) {

console.error('Error decodificando token:', error);

this.logout();

return null;

}

}

**¿Por qué tu implementación actual es buena?**

* **Separación de preocupaciones**: El servicio maneja la lógica del token
* **Seguridad**: Hace logout automático con tokens inválidos
* **Reusabilidad**: Los componentes no necesitan saber cómo se decodifica el token
* **Mantenibilidad**: Fácil de modificar si cambia la estructura del token

**Caso a tener en cuenta:**

Si tu backend usa sub para el ID pero es string y necesitas número:

typescript

this.user.id = Number(cut.sub); // Conversión explícita

Tu implementación es sólida, y con estas pequeñas mejoras quedaría aún más robusta y type-safe. El enfoque que has tomado es exactamente el que usaría en un proyecto profesional.

**loadUserData(): void { // Suponiendo que tienes un método para obtener los datos del usuario logueado const cut = this.authService.getDecodedToken(); if (cut) { this.user.id = cut.sub; this.user.name = cut.name; this.user.email = cut.email; } this.apiService.get(`users/id/${this.user.id}`) } Como capturo la respuesta**

Para capturar la respuesta de la llamada API que obtiene los detalles del usuario, necesitas suscribirte al Observable que retorna el método get(). Aquí te muestro cómo hacerlo correctamente:

**Implementación completa con manejo de respuesta:**

typescript

loadUserData(): void {

// 1. Obtener datos del token

const decodedToken = this.authService.getDecodedToken();

if (decodedToken) {

// 2. Asignar datos básicos del token

this.user.id = decodedToken.sub;

this.user.name = decodedToken.name;

this.user.email = decodedToken.email;

// 3. Obtener detalles adicionales del API

this.apiService.get(`users/id/${this.user.id}`).subscribe({

next: (userDetails) => {

// 4. Actualizar datos con la respuesta del API

this.user = {

...this.user,

...userDetails, // Sobreescribe o añade propiedades

stats: {

tasksCompleted: userDetails.completedTasks || 0,

tasksPending: userDetails.pendingTasks || 0,

categories: userDetails.categoriesCount || 0

}

};

},

error: (err) => {

console.error('Error al cargar detalles del usuario:', err);

// Puedes manejar el error como prefieras:

// - Mostrar un mensaje al usuario

// - Usar valores por defecto

}

});

}

}

**Versión con async/await (si usas Angular 16+):**

typescript

async loadUserData(): Promise<void> {

try {

const decodedToken = this.authService.getDecodedToken();

if (!decodedToken) return;

this.user.id = decodedToken.sub;

this.user.name = decodedToken.name;

this.user.email = decodedToken.email;

const userDetails = await firstValueFrom(

this.apiService.get(`users/id/${this.user.id}`)

);

this.user = {

...this.user,

...userDetails,

stats: {

tasksCompleted: userDetails.completedTasks || 0,

tasksPending: userDetails.pendingTasks || 0,

categories: userDetails.categoriesCount || 0

}

};

} catch (err) {

console.error('Error:', err);

// Manejo de errores

}

}

**Recomendaciones adicionales:**

1. **Tipado de respuestas**:

typescript

interface UserDetails {

completedTasks: number;

pendingTasks: number;

categoriesCount: number;

// ... otras propiedades

}

// En tu API Service

get<T>(endpoint: string): Observable<T> {

return this.http.get<T>(`${this.baseUrl}/${endpoint}`);

}

// Luego en el componente

this.apiService.get<UserDetails>(`users/id/${this.user.id}`)

.subscribe((details) => { ... });

1. **Loader/Estado de carga**:

typescript

isLoading = false;

loadUserData(): void {

this.isLoading = true;

this.apiService.get(...).subscribe({

next: (response) => { ... },

error: (err) => { ... },

complete: () => this.isLoading = false

});

}

1. **Cancelar suscripciones** (para evitar memory leaks):

typescript

private destroy$ = new Subject<void>();

ngOnInit(): void {

this.loadUserData();

}

ngOnDestroy(): void {

this.destroy$.next();

this.destroy$.complete();

}

// En la suscripción:

this.apiService.get(...).pipe(

takeUntil(this.destroy$)

).subscribe(...);

1. **Manejo de errores en UI**:

html

<div \*ngIf="isLoading">Cargando...</div>

<div \*ngIf="errorMessage" class="alert alert-danger">{{ errorMessage }}</div>

Esta implementación te permite:

* Obtener primero los datos básicos del token
* Luego complementarlos con información del backend
* Manejar correctamente los estados de carga y error
* Mantener un código bien tipado y organizado

**por lo que sea el usaurio no se está actualizando import { Component } from '@angular/core'; import { AuthService } from '../../shared/services/auth.service'; import { jwtDecode } from 'jwt-decode'; import { ApiService } from '../../shared/services/api.service'; @Component({ selector: 'app-profile', standalone: false, templateUrl: './profile.component.html', styleUrls: ['./profile.component.scss'] }) export class ProfileComponent { user = { id: 3, name: 'Alex García', email: 'alex@ejemplo.com', stats: { tasksCompleted: 0, tasksPending: 0, categories: 0 }, }; constructor( private authService: AuthService, private apiService: ApiService ) {} ngOnInit(): void { this.loadUserData(); //this.loadStatistics(); } loadUserData(): void { // Suponiendo que tienes un método para obtener los datos del usuario logueado const cut = this.authService.getDecodedToken(); // if (cut) { // this.user.id = cut.sub; // this.user.name = cut.name; // this.user.email = cut.email; // } // <- Actualmente desde el token podría sacarse todo //this.apiService.get(`users/id/${this.user.id}`) // <- Este endpoint sirve pero mejor usar el seguro this.apiService.get(`auth/getProfile`).subscribe({ next: (userDetails) => { this.user.id = userDetails.id this.user.name = userDetails.name this.user.email = userDetails.email } }) } logout(): void { this.authService.logout(); // Llama al método logout de tu AuthService } }**

**Solución para el Problema de Actualización del Usuario**

El problema parece estar en cómo estás manejando la actualización del objeto user. Aquí te presento una solución completa con explicaciones:

**Problemas Detectados**

1. **Inicialización estática del objeto user**: Estás creando un objeto con valores por defecto que podrían estar sobrescribiendo los datos reales.
2. **Manejo de la respuesta API**: La actualización del objeto user no es reactiva.
3. **Falta de tipado**: No hay interfaces definidas para los datos del usuario.

**Solución Optimizada**

typescript

import { Component } from '@angular/core';

import { AuthService } from '../../shared/services/auth.service';

import { ApiService } from '../../shared/services/api.service';

// Interfaces para tipado seguro

interface UserStats {

tasksCompleted: number;

tasksPending: number;

categories: number;

}

interface UserProfile {

id: number;

name: string;

email: string;

stats: UserStats;

}

@Component({

selector: 'app-profile',

templateUrl: './profile.component.html',

styleUrls: ['./profile.component.scss']

})

export class ProfileComponent {

user: UserProfile = {

id: 0,

name: '',

email: '',

stats: {

tasksCompleted: 0,

tasksPending: 0,

categories: 0

}

};

isLoading = true;

errorMessage: string | null = null;

constructor(

private authService: AuthService,

private apiService: ApiService

) {}

ngOnInit(): void {

this.loadUserData();

}

loadUserData(): void {

this.isLoading = true;

this.errorMessage = null;

// Opción 1: Usar el endpoint seguro

this.apiService.get<UserProfile>('auth/getProfile').subscribe({

next: (userDetails) => {

this.user = {

...userDetails,

stats: {

tasksCompleted: userDetails.stats?.tasksCompleted || 0,

tasksPending: userDetails.stats?.tasksPending || 0,

categories: userDetails.stats?.categories || 0

}

};

this.isLoading = false;

},

error: (err) => {

console.error('Error loading profile:', err);

this.errorMessage = 'Error al cargar el perfil';

this.isLoading = false;

// Fallback: intentar con datos del token si el API falla

this.loadFromToken();

}

});

}

private loadFromToken(): void {

const tokenData = this.authService.getDecodedToken();

if (tokenData) {

this.user = {

id: tokenData.sub || 0,

name: tokenData.name || 'Usuario',

email: tokenData.email || '',

stats: this.user.stats // Mantener las estadísticas existentes

};

}

}

logout(): void {

this.authService.logout();

}

}

**Mejoras Implementadas**

1. **Tipado fuerte**:
   * Interfaces UserStats y UserProfile para mayor seguridad en el código.
2. **Manejo de estados**:
   * isLoading para mostrar un spinner durante la carga.
   * errorMessage para mostrar errores al usuario.
3. **Estrategia de fallback**:
   * Si el endpoint API falla, intenta obtener datos básicos del token.
4. **Actualización reactiva**:
   * Reemplazo completo del objeto user para asegurar la detección de cambios.
5. **Operador spread**:
   * Uso de ... para mantener la inmutabilidad.

**HTML Ajustado (opcional)**

html

<div class="container py-5">

<div \*ngIf="isLoading" class="text-center">

<div class="spinner-border text-primary" role="status">

<span class="visually-hidden">Cargando...</span>

</div>

</div>

<div \*ngIf="errorMessage" class="alert alert-danger">

{{ errorMessage }}

</div>

<div \*ngIf="!isLoading && !errorMessage" class="row justify-content-center">

<!-- Tu contenido actual del perfil aquí -->

<div class="col-lg-8">

<div class="card border-0 shadow-sm">

<div class="card-body p-4">

<!-- Mostrar datos de this.user -->

<h2>{{ user.name }}</h2>

<p>{{ user.email }}</p>

<!-- ... resto de tu template ... -->

</div>

</div>

</div>

</div>

</div>

**Posibles Causas del Problema Original**

1. **El endpoint no retorna los datos esperados**:
   * Verifica con Postman/Insomnia que auth/getProfile retorna los datos correctamente.
2. **Problema de detección de cambios**:
   * Angular no detecta cambios cuando modificas propiedades anidadas directamente.
3. **Token no contiene los datos esperados**:
   * Verifica que tu token JWT incluya los campos sub, name y email.
4. **CORS o problemas de autenticación**:
   * Revisa la consola del navegador (F12) para ver errores de red.

Esta solución debería resolver tus problemas de actualización del usuario. Si persisten los problemas, recomiendo verificar:

1. La respuesta real del endpoint auth/getProfile
2. Que el token JWT contenga los datos esperados
3. Que no haya errores en la consola del navegador

**Como ayudan mis cambios exactamente a que sea reactivo? import { Component } from '@angular/core'; import { AuthService } from '../../shared/services/auth.service'; import { jwtDecode } from 'jwt-decode'; import { ApiService } from '../../shared/services/api.service'; @Component({ selector: 'app-profile', standalone: false, templateUrl: './profile.component.html', styleUrls: ['./profile.component.scss'] }) export class ProfileComponent { user = { id: 3, name: 'Alex García', email: 'alex@ejemplo.com', stats: { tasksCompleted: 0, tasksPending: 0, categories: 0 }, }; isLoading = true; errorMessage: string | null = null; constructor( private authService: AuthService, private apiService: ApiService ) {} ngOnInit(): void { this.loadUserData(); //this.loadStatistics(); } loadUserData(): void { this.isLoading = true; this.errorMessage = null; // Suponiendo que tienes un método para obtener los datos del usuario logueado const cut = this.authService.getDecodedToken(); // if (cut) { // this.user.id = cut.sub; // this.user.name = cut.name; // this.user.email = cut.email; // } // <- Actualmente desde el token podría sacarse todo //this.apiService.get(`users/id/${this.user.id}`) // <- Este endpoint sirve pero mejor usar el seguro this.apiService.get(`auth/getProfile`).subscribe({ next: (userDetails) => { this.user.id = userDetails.id this.user.name = userDetails.name this.user.email = userDetails.email this.isLoading = false; }, error: (err) => { console.error('Error loading profile:', err); this.errorMessage = 'Error al cargar el perfil'; this.isLoading = false; } }) } logout(): void { this.authService.logout(); // Llama al método logout de tu AuthService } }**

Tus cambios introducen mejoras importantes para hacer el componente más reactivo, pero hay algunos ajustes clave que podrían optimizarlo aún más. Vamos a analizar cómo ayudan tus cambios y qué más podrías hacer:

**Tus Mejoras Actuales:**

1. **Estados de carga y error**:

typescript

 isLoading = true;

errorMessage: string | null = null;

* Permiten una UI reactiva mostrando spinners o mensajes de error
* Mejoran la experiencia de usuario durante operaciones asíncronas

 **Manejo adecuado de la suscripción**:

typescript

1. this.apiService.get(`auth/getProfile`).subscribe({
2. next: (userDetails) => { ... },
3. error: (err) => { ... }
4. })
   * Manejas correctamente los casos de éxito y error

**Problemas de Reactividad Persistente:**

1. **Actualización por propiedades**:

typescript

1. this.user.id = userDetails.id
2. this.user.name = userDetails.name
3. this.user.email = userDetails.email
   * Angular tiene problemas para detectar cambios cuando modificas propiedades anidadas directamente
   * Especialmente con objetos complejos que tienen nested properties
4. **Falta de inmutabilidad**:
   * Modificar el objeto existente puede no disparar los detectores de cambios

**Solución Completa para Mejorar Reactividad:**

typescript

import { Component } from '@angular/core';

import { AuthService } from '../../shared/services/auth.service';

import { ApiService } from '../../shared/services/api.service';

interface UserProfile {

id: number;

name: string;

email: string;

stats?: {

tasksCompleted: number;

tasksPending: number;

categories: number;

};

}

@Component({

selector: 'app-profile',

templateUrl: './profile.component.html',

styleUrls: ['./profile.component.scss']

})

export class ProfileComponent {

user: UserProfile = {

id: 0,

name: '',

email: '',

stats: {

tasksCompleted: 0,

tasksPending: 0,

categories: 0

}

};

isLoading = true;

errorMessage: string | null = null;

constructor(

private authService: AuthService,

private apiService: ApiService

) {}

ngOnInit(): void {

this.loadUserData();

}

loadUserData(): void {

this.isLoading = true;

this.errorMessage = null;

this.apiService.get<UserProfile>('auth/getProfile').subscribe({

next: (userDetails) => {

// Método 1: Reemplazo completo del objeto (recomendado)

this.user = {

...userDetails,

stats: {

tasksCompleted: userDetails.stats?.tasksCompleted || 0,

tasksPending: userDetails.stats?.tasksPending || 0,

categories: userDetails.stats?.categories || 0

}

};

// Método alternativo 2: Usar Object.assign

// this.user = Object.assign({}, this.user, userDetails);

this.isLoading = false;

},

error: (err) => {

console.error('Error loading profile:', err);

this.errorMessage = 'Error al cargar el perfil';

this.isLoading = false;

}

});

}

logout(): void {

this.authService.logout();

}

}

**¿Por qué esto mejora la reactividad?**

1. **Reemplazo completo del objeto**:
   * Al crear un nuevo objeto con { ... }, Angular detecta el cambio de referencia
   * Esto dispara el mecanismo de detección de cambios automáticamente
2. **Manejo de datos anidados**:
   * El operador spread (...) copia todas las propiedades al nuevo objeto
   * Aseguras que los cambios en propiedades anidadas sean detectados
3. **Tipado fuerte**:
   * La interfaz UserProfile ayuda a mantener la estructura consistente
   * Previene errores al asignar propiedades incorrectas

**Recomendaciones Adicionales:**

1. **Usar OnPush Change Detection** (para optimización):

typescript

 import { ChangeDetectionStrategy } from '@angular/core';

@Component({

// ...

changeDetection: ChangeDetectionStrategy.OnPush

})

* Requerirá usar async pipe o ChangeDetectorRef para cambios manuales

 **Manejo de observables con async pipe**:

typescript

 user$ = this.apiService.get<UserProfile>('auth/getProfile').pipe(

catchError(err => {

this.errorMessage = 'Error al cargar el perfil';

return of(this.getDefaultUser());

})

);

* Luego en el template: \*ngIf="user$ | async as user"

 **TrackBy para listas** (si muestras listas):

html

1. <div \*ngFor="let item of items; trackBy: trackById">

Estos cambios asegurarán que tu componente sea completamente reactivo y eficiente en la actualización de la UI cuando los datos cambien.

**<div class="container py-5"> <div class="row justify-content-center"> <div class="col-lg-8"> <div class="card border-0 shadow-sm"> <div class="card-body p-4"> <!-- Header del perfil --> <div class="d-flex flex-column align-items-center text-center mb-4"> <div class="rounded-circle bg-primary bg-opacity-10 d-flex align-items-center justify-content-center mb-3" style="width: 120px; height: 120px;"> <i class="bi bi-person-fill text-primary" style="font-size: 3rem;"></i> </div> <h2 class="mb-1">{{ user.name }}</h2> <p class="text-muted mb-3">{{ user.email }}</p> <span class="badge bg-primary rounded-pill px-3">Usuario #{{ user.id }}</span> </div> <!-- Sección de información --> <div class="row g-3 mb-4"> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-person-lines-fill me-2 text-primary"></i> Información Básica </h3> <ul class="list-unstyled"> <li class="mb-2"> <strong>ID de usuario:</strong> #{{ user.id }} </li> <li class="mb-2"> <strong>Nombre:</strong> {{ user.name }} </li> <li class="mb-2"> <strong>Email:</strong> {{ user.email }} </li> </ul> </div> </div> </div> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-bar-chart-line-fill me-2 text-primary"></i> Tu Productividad </h3> <div class="d-flex flex-wrap gap-3"> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksCompleted }}</div> <small class="text-muted">Tareas completadas</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksPending }}</div> <small class="text-muted">Tareas pendientes</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.categories }}</div> <small class="text-muted">Categorías</small> </div> </div> </div> </div> </div> </div> <!-- Acciones --> <div class="d-flex justify-content-center gap-3"> <button class="btn btn-primary px-4"> <i class="bi bi-pencil-square me-2"></i> Editar Perfil </button> <button class="btn btn-outline-danger px-4" (click)="logout()"> <i class="bi bi-box-arrow-right me-2"></i> Cerrar Sesión </button> </div> </div> </div> </div> </div> </div> Tal vez el problema es que hay algo mal**

Analizando tu código, hay varios aspectos que podrían estar afectando la reactividad y actualización del componente. Veamos los posibles problemas y soluciones:

**Problemas Detectados:**

1. **Actualización parcial del objeto user**:
   * En loadUserData() solo actualizas id, name y email, pero no las stats
   * Esto puede causar inconsistencias si el backend devuelve stats diferentes
2. **Estructura de datos no coincidente**:
   * El template espera user.stats con tres propiedades
   * Pero en la respuesta API podrían venir con diferente estructura
3. **Falta de manejo inicial de datos**:
   * El objeto user tiene valores por defecto que podrían interferir

**Solución Completa:**

typescript

import { Component } from '@angular/core';

import { AuthService } from '../../shared/services/auth.service';

import { ApiService } from '../../shared/services/api.service';

interface UserStats {

tasksCompleted: number;

tasksPending: number;

categories: number;

}

interface UserProfile {

id: number;

name: string;

email: string;

stats: UserStats;

}

@Component({

selector: 'app-profile',

templateUrl: './profile.component.html',

styleUrls: ['./profile.component.scss']

})

export class ProfileComponent {

user: UserProfile = {

id: 0,

name: 'Cargando...',

email: '',

stats: {

tasksCompleted: 0,

tasksPending: 0,

categories: 0

}

};

isLoading = true;

constructor(

private authService: AuthService,

private apiService: ApiService

) {}

ngOnInit(): void {

this.loadUserData();

}

loadUserData(): void {

this.isLoading = true;

this.apiService.get<any>('auth/getProfile').subscribe({

next: (response) => {

// Asegurar que las stats tengan siempre la estructura correcta

const stats = response.stats || {};

this.user = {

id: response.id || 0,

name: response.name || 'Usuario',

email: response.email || '',

stats: {

tasksCompleted: stats.tasksCompleted || 0,

tasksPending: stats.tasksPending || 0,

categories: stats.categories || stats.categoriesCount || 0

}

};

this.isLoading = false;

},

error: (err) => {

console.error('Error:', err);

this.user.name = 'Error al cargar';

this.isLoading = false;

}

});

}

logout(): void {

this.authService.logout();

}

}

**Cambios Clave:**

1. **Interfaces tipadas**:
   * Definí interfaces para asegurar la estructura de datos
2. **Transformación de datos**:
   * Normalizo la respuesta API para que coincida con lo que espera el template
   * Manejo casos donde stats podría venir con diferente nombre (ej: categoriesCount)
3. **Estado inicial claro**:
   * Valores por defecto más descriptivos ("Cargando...")
4. **Actualización completa**:
   * Siempre creo un nuevo objeto user en lugar de modificar propiedades

**Verificación Adicional:**

1. **Revisa la respuesta real del API**:

typescript

 this.apiService.get('auth/getProfile').subscribe(response => {

console.log('Respuesta completa:', response);

});

 **Asegúrate que las rutas de datos coincidan**:

* Si el API devuelve completed\_tasks en lugar de tasksCompleted, necesitarás mapearlo

 **Prueba este template alternativo para debug**:

html

1. <div class="card-body">
2. <pre>{{ user | json }}</pre>
3. </div>

**Posibles Causas del Problema:**

1. **El endpoint no devuelve stats**:
   * Verifica que tu API incluya las estadísticas en la respuesta
2. **Nombres de campos diferentes**:
   * El template espera tasksCompleted pero el API podría devolver completedTasks
3. **Problema de timing**:
   * Asegúrate que no haya llamadas concurrentes que sobreescriban los datos

Si después de estos cambios aún no se actualiza, recomiendo:

1. Verificar la pestaña "Red" en DevTools para ver la respuesta real del API
2. Añadir console.log antes de asignar this.user
3. Verificar que no haya errores en la consola del navegador

Esta solución debería resolver los problemas de reactividad y asegurar que los datos se muestren correctamente.

**La cosa es que no quiero actualizar los stats, por ahora solo el id name y email y no se actualiza. Hay algún fallo en la reactividad <div class="container py-5"> <div class="row justify-content-center"> <div class="col-lg-8"> <div class="card border-0 shadow-sm"> <div class="card-body p-4"> <!-- Header del perfil --> <div class="d-flex flex-column align-items-center text-center mb-4"> <div class="rounded-circle bg-primary bg-opacity-10 d-flex align-items-center justify-content-center mb-3" style="width: 120px; height: 120px;"> <i class="bi bi-person-fill text-primary" style="font-size: 3rem;"></i> </div> <h2 class="mb-1">{{ user.name }}</h2> <p class="text-muted mb-3">{{ user.email }}</p> <span class="badge bg-primary rounded-pill px-3">Usuario #{{ user.id }}</span> </div> <!-- Sección de información --> <div class="row g-3 mb-4"> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-person-lines-fill me-2 text-primary"></i> Información Básica </h3> <ul class="list-unstyled"> <li class="mb-2"> <strong>ID de usuario:</strong> #{{ user.id }} </li> <li class="mb-2"> <strong>Nombre:</strong> {{ user.name }} </li> <li class="mb-2"> <strong>Email:</strong> {{ user.email }} </li> </ul> </div> </div> </div> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-bar-chart-line-fill me-2 text-primary"></i> Tu Productividad </h3> <div class="d-flex flex-wrap gap-3"> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksCompleted }}</div> <small class="text-muted">Tareas completadas</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksPending }}</div> <small class="text-muted">Tareas pendientes</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.categories }}</div> <small class="text-muted">Categorías</small> </div> </div> </div> </div> </div> </div> <!-- Acciones --> <div class="d-flex justify-content-center gap-3"> <button class="btn btn-primary px-4"> <i class="bi bi-pencil-square me-2"></i> Editar Perfil </button> <button class="btn btn-outline-danger px-4" (click)="logout()"> <i class="bi bi-box-arrow-right me-2"></i> Cerrar Sesión </button> </div> </div> </div> </div> </div> </div> import { Component } from '@angular/core'; import { AuthService } from '../../shared/services/auth.service'; import { jwtDecode } from 'jwt-decode'; import { ApiService } from '../../shared/services/api.service'; @Component({ selector: 'app-profile', standalone: false, templateUrl: './profile.component.html', styleUrls: ['./profile.component.scss'] }) export class ProfileComponent { user = { id: 3, name: 'Alex García', email: 'alex@ejemplo.com', stats: { tasksCompleted: 0, tasksPending: 0, categories: 0 }, }; isLoading = true; errorMessage: string | null = null; constructor( private authService: AuthService, private apiService: ApiService ) {} ngOnInit(): void { this.loadUserData(); //this.loadStatistics(); } loadUserData(): void { this.isLoading = true; this.errorMessage = null; // Suponiendo que tienes un método para obtener los datos del usuario logueado const cut = this.authService.getDecodedToken(); // if (cut) { // this.user.id = cut.sub; // this.user.name = cut.name; // this.user.email = cut.email; // } // <- Actualmente desde el token podría sacarse todo //this.apiService.get(`users/id/${this.user.id}`) // <- Este endpoint sirve pero mejor usar el seguro this.apiService.get(`auth/getProfile`).subscribe({ next: (userDetails) => { this.user.id = userDetails.id this.user.name = userDetails.name this.user.email = userDetails.email this.isLoading = false; }, error: (err) => { console.error('Error loading profile:', err); this.errorMessage = 'Error al cargar el perfil'; this.isLoading = false; } }) } logout(): void { this.authService.logout(); // Llama al método logout de tu AuthService } }**

**Solución al Problema de Reactividad en el Perfil**

El problema principal es que Angular no está detectando los cambios cuando actualizas las propiedades del objeto user de forma individual. Aquí está la solución completa:

**Problemas Clave**

1. **Modificación directa de propiedades**:
   * Estás cambiando this.user.id, this.user.name, etc. directamente
   * Angular no siempre detecta estos cambios en objetos existentes
2. **Falta de inmutabilidad**:
   * Necesitas crear un nuevo objeto para que Angular detecte el cambio
3. **Estructura inicial vs respuesta API**:
   * El objeto inicial tiene valores por defecto que podrían interferir

**Solución Actualizada**

typescript

import { Component } from '@angular/core';

import { AuthService } from '../../shared/services/auth.service';

import { ApiService } from '../../shared/services/api.service';

@Component({

selector: 'app-profile',

templateUrl: './profile.component.html',

styleUrls: ['./profile.component.scss']

})

export class ProfileComponent {

user: any = null; // Inicializar como null

isLoading = true;

errorMessage: string | null = null;

constructor(

private authService: AuthService,

private apiService: ApiService

) {}

ngOnInit(): void {

this.loadUserData();

}

loadUserData(): void {

this.isLoading = true;

this.errorMessage = null;

this.apiService.get(`auth/getProfile`).subscribe({

next: (userDetails) => {

// Crear un NUEVO objeto user con los datos actualizados

this.user = {

...this.user, // Mantiene las stats existentes si las hay

id: userDetails.id,

name: userDetails.name,

email: userDetails.email,

stats: this.user?.stats || { // Preserva las stats o usa valores por defecto

tasksCompleted: 0,

tasksPending: 0,

categories: 0

}

};

this.isLoading = false;

},

error: (err) => {

console.error('Error loading profile:', err);

this.errorMessage = 'Error al cargar el perfil';

this.isLoading = false;

}

});

}

logout(): void {

this.authService.logout();

}

}

**¿Por qué funciona esto?**

1. **Inmutabilidad**:
   * Al crear un nuevo objeto con { ... }, Angular detecta el cambio de referencia
   * Esto dispara el mecanismo de detección de cambios
2. **Preservación de stats**:
   * Usamos el operador spread (...) para mantener las estadísticas existentes
   * Si no existen, se usan valores por defecto
3. **Estructura más limpia**:
   * Inicializamos user como null y lo construimos completamente con la respuesta

**Alternativa con ChangeDetectorRef**

Si aún tienes problemas, puedes forzar la detección de cambios manualmente:

typescript

import { ChangeDetectorRef } from '@angular/core';

// Inyectar en el constructor

constructor(

private authService: AuthService,

private apiService: ApiService,

private cdr: ChangeDetectorRef

) {}

// Luego en la suscripción:

next: (userDetails) => {

this.user = {

// ... misma lógica anterior

};

this.cdr.detectChanges(); // Forzar detección de cambios

this.isLoading = false;

}

**Verificación Adicional**

1. **Comprueba la respuesta API**:

typescript

 next: (userDetails) => {

console.log('Datos recibidos:', userDetails);

// ... resto del código

}

 **Asegúrate que los campos coincidan**:

* Verifica que userDetails.id, userDetails.name y userDetails.email existan

 **Template de debug**:

html

1. <div class="card-body">
2. <pre>Datos actuales: {{ user | json }}</pre>
3. <pre>Estado carga: {{ isLoading }}</pre>
4. </div>

Esta solución debería resolver tus problemas de reactividad manteniendo las estadísticas sin cambios mientras actualiza solo los datos básicos del usuario.

**import { Component } from '@angular/core'; import { AuthService } from '../../shared/services/auth.service'; import { jwtDecode } from 'jwt-decode'; import { ApiService } from '../../shared/services/api.service'; @Component({ selector: 'app-profile', standalone: false, templateUrl: './profile.component.html', styleUrls: ['./profile.component.scss'] }) export class ProfileComponent { user = { id: 3, name: 'Alex García', email: 'alex@ejemplo.com', stats: { tasksCompleted: 0, tasksPending: 0, categories: 0 }, }; isLoading = true; errorMessage: string | null = null; constructor( private authService: AuthService, private apiService: ApiService ) {} ngOnInit(): void { this.loadUserData(); //this.loadStatistics(); } loadUserData(): void { this.isLoading = true; this.errorMessage = null; // Suponiendo que tienes un método para obtener los datos del usuario logueado const cut = this.authService.getDecodedToken(); // if (cut) { // this.user.id = cut.sub; // this.user.name = cut.name; // this.user.email = cut.email; // } // <- Actualmente desde el token podría sacarse todo //this.apiService.get(`users/id/${this.user.id}`) // <- Este endpoint sirve pero mejor usar el seguro this.apiService.get(`auth/getProfile`).subscribe({ next: (userDetails) => { this.user = { ...this.user, // Mantiene las stats existentes si las hay id: userDetails.id, name: userDetails.name, email: userDetails.email, stats: this.user?.stats || { // Preserva las stats o usa valores por defecto tasksCompleted: 0, tasksPending: 0, categories: 0 } } this.isLoading = false; }, error: (err) => { console.error('Error loading profile:', err); this.errorMessage = 'Error al cargar el perfil'; this.isLoading = false; } }) } logout(): void { this.authService.logout(); // Llama al método logout de tu AuthService } } <div class="container py-5"> <div class="row justify-content-center"> <div class="col-lg-8"> <div class="card border-0 shadow-sm"> <div class="card-body p-4"> <!-- Header del perfil --> <div class="d-flex flex-column align-items-center text-center mb-4"> <div class="rounded-circle bg-primary bg-opacity-10 d-flex align-items-center justify-content-center mb-3" style="width: 120px; height: 120px;"> <i class="bi bi-person-fill text-primary" style="font-size: 3rem;"></i> </div> <h2 class="mb-1">{{ user.name }}</h2> <p class="text-muted mb-3">{{ user.email }}</p> <span class="badge bg-primary rounded-pill px-3">Usuario #{{ user.id }}</span> </div> <!-- Sección de información --> <div class="row g-3 mb-4"> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-person-lines-fill me-2 text-primary"></i> Información Básica </h3> <ul class="list-unstyled"> <li class="mb-2"> <strong>ID de usuario:</strong> #{{ user.id }} </li> <li class="mb-2"> <strong>Nombre:</strong> {{ user.name }} </li> <li class="mb-2"> <strong>Email:</strong> {{ user.email }} </li> </ul> </div> </div> </div> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-bar-chart-line-fill me-2 text-primary"></i> Tu Productividad </h3> <div class="d-flex flex-wrap gap-3"> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksCompleted }}</div> <small class="text-muted">Tareas completadas</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksPending }}</div> <small class="text-muted">Tareas pendientes</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.categories }}</div> <small class="text-muted">Categorías</small> </div> </div> </div> </div> </div> </div> <!-- Acciones --> <div class="d-flex justify-content-center gap-3"> <button class="btn btn-primary px-4"> <i class="bi bi-pencil-square me-2"></i> Editar Perfil </button> <button class="btn btn-outline-danger px-4" (click)="logout()"> <i class="bi bi-box-arrow-right me-2"></i> Cerrar Sesión </button> </div> </div> </div> </div> </div> </div> SIGUE SIN FUNCIONAR**

**Solución Definitiva para la Actualización del Perfil**

El problema persiste porque aunque estás creando un nuevo objeto, hay varios factores que podrían estar interfiriendo. Vamos a implementar una solución completa con pasos de verificación:

**Código Actualizado del Componente**

typescript

import { Component, ChangeDetectorRef } from '@angular/core';

import { AuthService } from '../../shared/services/auth.service';

import { ApiService } from '../../shared/services/api.service';

interface UserProfile {

id: number;

name: string;

email: string;

stats: {

tasksCompleted: number;

tasksPending: number;

categories: number;

};

}

@Component({

selector: 'app-profile',

templateUrl: './profile.component.html',

styleUrls: ['./profile.component.scss']

})

export class ProfileComponent {

user: UserProfile = {

id: 0,

name: 'Cargando...',

email: '',

stats: {

tasksCompleted: 0,

tasksPending: 0,

categories: 0

}

};

isLoading = true;

errorMessage: string | null = null;

constructor(

private authService: AuthService,

private apiService: ApiService,

private cdr: ChangeDetectorRef

) {}

ngOnInit(): void {

this.loadUserData();

}

loadUserData(): void {

this.isLoading = true;

this.errorMessage = null;

console.log('Iniciando carga de datos...'); // Debug 1

this.apiService.get('auth/getProfile').subscribe({

next: (response: any) => {

console.log('Respuesta recibida:', response); // Debug 2

if (!response) {

console.error('La respuesta está vacía');

return;

}

// Crear un NUEVO objeto completamente

const updatedUser: UserProfile = {

id: response.id || 0,

name: response.name || 'Nombre no disponible',

email: response.email || '',

stats: {

tasksCompleted: this.user.stats.tasksCompleted,

tasksPending: this.user.stats.tasksPending,

categories: this.user.stats.categories

}

};

console.log('Usuario actualizado:', updatedUser); // Debug 3

this.user = updatedUser;

this.isLoading = false;

// Forzar detección de cambios

this.cdr.detectChanges();

},

error: (err) => {

console.error('Error en la solicitud:', err); // Debug 4

this.errorMessage = 'Error al cargar el perfil';

this.isLoading = false;

this.cdr.detectChanges();

}

});

}

logout(): void {

this.authService.logout();

}

}

**Pasos para Diagnosticar el Problema**

1. **Verifica la respuesta del API**:
   * Revisa la consola del navegador (F12 > Console) para ver los logs de "Respuesta recibida"
   * Asegúrate que los campos id, name y email existen en la respuesta
2. **Comprueba el estado del componente**:
   * Los console.log te mostrarán el flujo completo de ejecución
   * Verifica que llegue hasta "Usuario actualizado"
3. **Inspecciona la red**:
   * En DevTools > Red, busca la llamada a auth/getProfile
   * Revisa el código de estado HTTP y el cuerpo de la respuesta
4. **Verifica el template**:
   * Añade temporalmente esto en tu HTML para ver los datos en tiempo real:

html

* + <div class="alert alert-info">
  + <pre>Datos actuales: {{ user | json }}</pre>
  + <div>Estado carga: {{ isLoading ? 'Cargando...' : 'Completado' }}</div>
  + </div>

**Posibles Causas y Soluciones**

1. **El endpoint no devuelve los datos esperados**:

typescript

 // Prueba con un objeto fijo para descartar problemas de API

next: (response: any) => {

response = { id: 1, name: 'TEST', email: 'test@test.com' };

// ... resto del código

}

 **Problema con la detección de cambios**:

* Asegúrate que ChangeDetectorRef está inyectado correctamente
* Prueba cambiar la estrategia de detección en el componente:

typescript

* 
* @Component({
* // ...
* changeDetection: ChangeDetectionStrategy.OnPush

})

 **Conflicto con valores iniciales**:

* Cambia la inicialización a null y maneja el caso en el template:

typescript

user: UserProfile | null = null;

html

* 
* <div \*ngIf="user">
* <!-- Tu template actual -->

</div>

 **Problema con el servicio ApiService**:

* Verifica que el método get() esté retornando correctamente el Observable
* Prueba con una implementación mínima:

typescript

* + this.apiService.get = () => of({
  + id: 99,
  + name: 'Usuario de prueba',
  + email: 'test@angular.com'
  + });

Si después de estas verificaciones el problema persiste, recomiendo:

1. Crear un stackblitz mínimo que reproduzca el error
2. Verificar si hay errores en la consola del navegador
3. Revisar si hay interceptores HTTP que puedan estar modificando la respuesta

**// edit-profile-modal.component.ts import { Component, Inject } from '@angular/core'; import { FormBuilder, FormGroup, Validators } from '@angular/forms'; import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog'; import { AuthService } from '../../../shared/services/auth.service'; @Component({ selector: 'app-edit-profile-modal', templateUrl: './edit-profile-modal.component.html', styleUrls: ['./edit-profile-modal.component.scss'] }) export class EditProfileModalComponent { editForm: FormGroup; isSubmitting = false; constructor( private fb: FormBuilder, private authService: AuthService, private dialogRef: MatDialogRef<EditProfileModalComponent>, @Inject(MAT\_DIALOG\_DATA) public data: { currentName: string } ) { this.editForm = this.fb.group({ name: [this.data.currentName, [Validators.minLength(3)]], currentPassword: [''], newPassword: ['', [Validators.minLength(6)]], confirmPassword: [''] }, { validator: this.passwordMatchValidator }); } passwordMatchValidator(form: FormGroup) { const newPassword = form.get('newPassword')?.value; const confirmPassword = form.get('confirmPassword')?.value; return newPassword === confirmPassword ? null : { mismatch: true }; } onSubmit() { if (this.editForm.invalid) return; this.isSubmitting = true; const { name, currentPassword, newPassword } = this.editForm.value; this.authService.updateProfile({ name, currentPassword, newPassword }).subscribe({ next: () => { this.dialogRef.close({ success: true, name }); }, error: (err) => { this.isSubmitting = false; console.error('Error updating profile:', err); } }); } } this.fb.group me dice que group está deprecado**

**Angular FormBuilder.group() Deprecation Warning**

The warning you're seeing about group being deprecated is likely from a newer version of Angular where they've started marking some FormBuilder methods as deprecated in favor of more type-safe alternatives.

**Solution**

Instead of using the untyped this.fb.group(), you should use the strongly typed this.fb.nonNullable.group() or provide explicit types:

**Option 1: Using nonNullable.group() (Angular 14+)**

typescript

this.editForm = this.fb.nonNullable.group({

name: [this.data.currentName, [Validators.minLength(3)]],

currentPassword: [''],

newPassword: ['', [Validators.minLength(6)]],

confirmPassword: ['']

}, { validators: this.passwordMatchValidator });

**Option 2: Explicitly typing the form group**

typescript

this.editForm = this.fb.group({

name: [this.data.currentName as string, [Validators.minLength(3)]],

currentPassword: ['' as string],

newPassword: ['' as string, [Validators.minLength(6)]],

confirmPassword: ['' as string]

}, { validators: this.passwordMatchValidator });

**Option 3: Using FormGroup with explicit interface**

typescript

interface EditProfileForm {

name: string;

currentPassword: string;

newPassword: string;

confirmPassword: string;

}

this.editForm = this.fb.group<EditProfileForm>({

name: this.fb.control(this.data.currentName, [Validators.minLength(3)]),

currentPassword: this.fb.control(''),

newPassword: this.fb.control('', [Validators.minLength(6)]),

confirmPassword: this.fb.control('')

}, { validators: this.passwordMatchValidator });

**Additional Notes**

1. Also notice I changed validator to validators in the options object (though both work, validators is the more correct plural form).
2. The deprecation is part of Angular's move toward more type-safe forms. While the old syntax still works, using the typed versions will give you better type checking and IDE support.
3. If you're using Angular 14+, consider using [Typed Forms](https://angular.io/guide/typed-forms) for even better type safety.

Pasame entonces todo el componente por favor.

Aquí tienes el componente completo actualizado con las mejores prácticas de Angular (versión 14+ con Typed Forms):

typescript

// edit-profile-modal.component.ts

import { Component, Inject } from '@angular/core';

import { FormBuilder, FormGroup, Validators, ValidatorFn, AbstractControl } from '@angular/forms';

import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog';

import { AuthService } from '../../../shared/services/auth.service';

interface ProfileForm {

name: string;

currentPassword: string;

newPassword: string;

confirmPassword: string;

}

@Component({

selector: 'app-edit-profile-modal',

templateUrl: './edit-profile-modal.component.html',

styleUrls: ['./edit-profile-modal.component.scss']

})

export class EditProfileModalComponent {

editForm: FormGroup<{

name: AbstractControl<string>;

currentPassword: AbstractControl<string>;

newPassword: AbstractControl<string>;

confirmPassword: AbstractControl<string>;

}>;

isSubmitting = false;

constructor(

private fb: FormBuilder,

private authService: AuthService,

private dialogRef: MatDialogRef<EditProfileModalComponent>,

@Inject(MAT\_DIALOG\_DATA) public data: { currentName: string }

) {

this.editForm = this.fb.nonNullable.group({

name: [this.data.currentName, [Validators.minLength(3)]],

currentPassword: [''],

newPassword: ['', [Validators.minLength(6)]],

confirmPassword: ['']

}, { validators: this.passwordMatchValidator() });

}

passwordMatchValidator(): ValidatorFn {

return (form: AbstractControl) => {

const passwordForm = form as FormGroup;

const newPassword = passwordForm.get('newPassword')?.value;

const confirmPassword = passwordForm.get('confirmPassword')?.value;

return newPassword === confirmPassword ? null : { mismatch: true };

};

}

onSubmit(): void {

if (this.editForm.invalid) return;

this.isSubmitting = true;

const { name, currentPassword, newPassword } = this.editForm.value;

this.authService.updateProfile({

name: name ?? this.data.currentName,

currentPassword: currentPassword ?? '',

newPassword: newPassword ?? ''

}).subscribe({

next: () => {

this.dialogRef.close({ success: true, name });

},

error: (err) => {

this.isSubmitting = false;

console.error('Error updating profile:', err);

}

});

}

}

**Cambios importantes realizados:**

1. **Typed Forms**: Implementé el sistema de tipos estrictos de Angular 14+
2. **Interfaz para el formulario**: Creé la interfaz ProfileForm
3. **nonNullable.group()**: Usé el constructor que asegura valores no nulos
4. **Validator tipado**: Actualicé el validador para usar ValidatorFn
5. **Manejo seguro de valores**: Uso del operador ?? para valores potencialmente undefined
6. **Tipado explícito**: Añadí tipos de retorno a los métodos

**Versión alternativa (si prefieres FormGroup explícito):**

typescript

// Otra opción con FormGroup explícito

this.editForm = new FormGroup<ProfileForm>({

name: new FormControl(this.data.currentName, {

nonNullable: true,

validators: [Validators.minLength(3)]

}),

currentPassword: new FormControl('', { nonNullable: true }),

newPassword: new FormControl('', {

nonNullable: true,

validators: [Validators.minLength(6)]

}),

confirmPassword: new FormControl('', { nonNullable: true })

}, { validators: this.passwordMatchValidator() });

Esta implementación es más robusta y aprovecha al máximo el sistema de tipos de Angular.

**// edit-profile-modal.component.ts import { Component, Inject } from '@angular/core'; import { FormBuilder, FormGroup, Validators, ValidatorFn, AbstractControl } from '@angular/forms'; import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog'; import { AuthService } from '../../../shared/services/auth.service'; interface ProfileForm { name: string; currentPassword: string; newPassword: string; confirmPassword: string; } @Component({ selector: 'app-edit-profile-modal', templateUrl: './edit-profile-modal.component.html', styleUrls: ['./edit-profile-modal.component.scss'] }) export class EditProfileModalComponent { editForm: FormGroup<{ name: AbstractControl<string>; currentPassword: AbstractControl<string>; newPassword: AbstractControl<string>; confirmPassword: AbstractControl<string>; }>; isSubmitting = false; constructor( private fb: FormBuilder, private authService: AuthService, private dialogRef: MatDialogRef<EditProfileModalComponent>, @Inject(MAT\_DIALOG\_DATA) public data: { currentName: string } ) { this.editForm = this.fb.nonNullable.group({ name: [this.data.currentName, [Validators.minLength(3)]], currentPassword: [''], newPassword: ['', [Validators.minLength(6)]], confirmPassword: [''] }, { validators: this.passwordMatchValidator() }); } passwordMatchValidator(): ValidatorFn { return (form: AbstractControl) => { const passwordForm = form as FormGroup; const newPassword = passwordForm.get('newPassword')?.value; const confirmPassword = passwordForm.get('confirmPassword')?.value; return newPassword === confirmPassword ? null : { mismatch: true }; }; } onSubmit(): void { if (this.editForm.invalid) return; this.isSubmitting = true; const { name, currentPassword, newPassword } = this.editForm.value; this.authService.updateProfile({ name: name ?? this.data.currentName, currentPassword: currentPassword ?? '', newPassword: newPassword ?? '' }).subscribe({ next: () => { this.dialogRef.close({ success: true, name }); }, error: (err) => { this.isSubmitting = false; console.error('Error updating profile:', err); } }); } } Type 'FormGroup<ɵNonNullableFormControls<{ name: (string | ValidatorFn[])[]; currentPassword: string[]; newPassword: (string | ValidatorFn[])[]; confirmPassword: string[]; }>>' is not assignable to type 'FormGroup<{ name: AbstractControl<string, string>; currentPassword: AbstractControl<string, string>; newPassword: AbstractControl<string, string>; confirmPassword: AbstractControl<...>; }>'. Type 'AbstractControl<string, string>' is missing the following properties from type 'FormControl<string>': defaultValue, registerOnChange, registerOnDisabledChangets(2322)**

Parece que hay un conflicto de tipos entre la definición de tu FormGroup y lo que realmente devuelve fb.nonNullable.group(). Vamos a corregirlo con una solución más limpia y compatible:

typescript

import { Component, Inject } from '@angular/core';

import { FormBuilder, FormGroup, Validators, ValidatorFn, AbstractControl, FormControl } from '@angular/forms';

import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog';

import { AuthService } from '../../../shared/services/auth.service';

interface ProfileForm {

name: string;

currentPassword: string;

newPassword: string;

confirmPassword: string;

}

@Component({

selector: 'app-edit-profile-modal',

templateUrl: './edit-profile-modal.component.html',

styleUrls: ['./edit-profile-modal.component.scss']

})

export class EditProfileModalComponent {

editForm: FormGroup<{

name: FormControl<string>;

currentPassword: FormControl<string>;

newPassword: FormControl<string>;

confirmPassword: FormControl<string>;

}>;

isSubmitting = false;

constructor(

private fb: FormBuilder,

private authService: AuthService,

private dialogRef: MatDialogRef<EditProfileModalComponent>,

@Inject(MAT\_DIALOG\_DATA) public data: { currentName: string }

) {

this.editForm = new FormGroup({

name: new FormControl(this.data.currentName, {

nonNullable: true,

validators: [Validators.minLength(3)]

}),

currentPassword: new FormControl('', { nonNullable: true }),

newPassword: new FormControl('', {

nonNullable: true,

validators: [Validators.minLength(6)]

}),

confirmPassword: new FormControl('', { nonNullable: true })

}, { validators: this.passwordMatchValidator() });

}

passwordMatchValidator(): ValidatorFn {

return (form: AbstractControl) => {

const passwordForm = form as FormGroup<ProfileForm>;

const newPassword = passwordForm.controls.newPassword.value;

const confirmPassword = passwordForm.controls.confirmPassword.value;

return newPassword === confirmPassword ? null : { mismatch: true };

};

}

onSubmit(): void {

if (this.editForm.invalid) return;

this.isSubmitting = true;

const formValue = this.editForm.getRawValue();

this.authService.updateProfile({

name: formValue.name,

currentPassword: formValue.currentPassword,

newPassword: formValue.newPassword

}).subscribe({

next: () => {

this.dialogRef.close({ success: true, name: formValue.name });

},

error: (err) => {

this.isSubmitting = false;

console.error('Error updating profile:', err);

}

});

}

}

**Cambios clave realizados:**

1. **Reemplacé fb.nonNullable.group()** con new FormGroup() y new FormControl() para mayor claridad en los tipos.
2. **Usé FormControl explícitamente** en la definición del tipo de editForm en lugar de AbstractControl.
3. **Utilicé getRawValue()** en lugar de value para obtener los valores del formulario de manera segura (especialmente importante con controles nonNullable).
4. **Simplifiqué el acceso a los controles** en el validador usando passwordForm.controls con tipado fuerte.
5. **Mantuve la estructura nonNullable** pero con una sintaxis más explícita y compatible con el sistema de tipos.

Esta solución debería resolver el error de tipos mientras mantiene toda la funcionalidad original y añade un mejor soporte de tipos.

**Type 'ProfileForm' does not satisfy the constraint '{ name: AbstractControl<any, any>; currentPassword: AbstractControl<any, any>; newPassword: AbstractControl<any, any>; confirmPassword: AbstractControl<...>; }'. Types of property 'name' are incompatible. Type 'string' is not assignable to type 'AbstractControl<any, any>'.**

El error ocurre porque estás intentando usar la interfaz ProfileForm (que define los tipos de los **valores** del formulario) donde se espera una definición de los **controles** del formulario. Vamos a corregir esto con una solución limpia y bien tipada:

typescript

import { Component, Inject } from '@angular/core';

import { FormBuilder, FormGroup, Validators, ValidatorFn, FormControl } from '@angular/forms';

import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog';

import { AuthService } from '../../../shared/services/auth.service';

interface ProfileFormValues {

name: string;

currentPassword: string;

newPassword: string;

confirmPassword: string;

}

type ProfileFormControls = {

[key in keyof ProfileFormValues]: FormControl<ProfileFormValues[key]>;

};

@Component({

selector: 'app-edit-profile-modal',

templateUrl: './edit-profile-modal.component.html',

styleUrls: ['./edit-profile-modal.component.scss']

})

export class EditProfileModalComponent {

editForm: FormGroup<ProfileFormControls>;

isSubmitting = false;

constructor(

private fb: FormBuilder,

private authService: AuthService,

private dialogRef: MatDialogRef<EditProfileModalComponent>,

@Inject(MAT\_DIALOG\_DATA) public data: { currentName: string }

) {

this.editForm = this.fb.group<ProfileFormControls>({

name: this.fb.control(this.data.currentName, {

nonNullable: true,

validators: [Validators.minLength(3)]

}),

currentPassword: this.fb.control('', { nonNullable: true }),

newPassword: this.fb.control('', {

nonNullable: true,

validators: [Validators.minLength(6)]

}),

confirmPassword: this.fb.control('', { nonNullable: true })

}, { validators: this.passwordMatchValidator() });

}

private passwordMatchValidator(): ValidatorFn {

return (form: AbstractControl) => {

const f = form as FormGroup<ProfileFormControls>;

return f.controls.newPassword.value === f.controls.confirmPassword.value

? null

: { mismatch: true };

};

}

onSubmit(): void {

if (this.editForm.invalid) return;

this.isSubmitting = true;

const formValue = this.editForm.getRawValue();

this.authService.updateProfile({

name: formValue.name,

currentPassword: formValue.currentPassword,

newPassword: formValue.newPassword

}).subscribe({

next: () => {

this.dialogRef.close({ success: true, name: formValue.name });

},

error: (err) => {

this.isSubmitting = false;

console.error('Error updating profile:', err);

}

});

}

}

**Cambios principales:**

1. **Separé las interfaces**:
   * ProfileFormValues para los valores del formulario
   * ProfileFormControls (tipo mapeado) para los controles del formulario
2. **Usé fb.control()** en lugar de arrays para una mejor tipado
3. **Simplifiqué el validador** usando el tipo ProfileFormControls
4. **Mantuve nonNullable** pero con la sintaxis correcta
5. **Mejoré el tipado del FormGroup** usando el tipo mapeado ProfileFormControls

Esta solución proporciona:

* Tipado fuerte en todo el formulario
* Claridad en la separación entre valores y controles
* Compatibilidad con las últimas versiones de Angular
* Mantenimiento más sencillo del código

**ayudame a modificarlo. Ten en cuenta que los datos son solamente, el nombre y la nueva contraseña. import { Component, Inject } from '@angular/core'; import { FormBuilder, FormGroup, Validators, ValidatorFn, FormControl, AbstractControl } from '@angular/forms'; import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog'; import { AuthService } from '../../../shared/services/auth.service'; import { ApiService } from '../../../shared/services/api.service'; interface ProfileFormValues { name: string; currentPassword: string; newPassword: string; confirmPassword: string; } type ProfileFormControls = { [key in keyof ProfileFormValues]: FormControl<ProfileFormValues[key]>; }; @Component({ selector: 'app-edit-profile-modal', templateUrl: './edit-profile-modal.component.html', styleUrls: ['./edit-profile-modal.component.scss'] }) export class EditProfileModalComponent { editForm: FormGroup<ProfileFormControls>; isSubmitting = false; constructor( private fb: FormBuilder, private authService: AuthService, private apiService: ApiService, private dialogRef: MatDialogRef<EditProfileModalComponent>, @Inject(MAT\_DIALOG\_DATA) public data: { currentName: string } ) { this.editForm = this.fb.group<ProfileFormControls>({ name: this.fb.control(this.data.currentName, { nonNullable: true, validators: [Validators.minLength(3)] }), currentPassword: this.fb.control('', { nonNullable: true }), newPassword: this.fb.control('', { nonNullable: true, validators: [Validators.minLength(6)] }), confirmPassword: this.fb.control('', { nonNullable: true }) }, { validators: this.passwordMatchValidator() }); } private passwordMatchValidator(): ValidatorFn { return (form: AbstractControl) => { const f = form as FormGroup<ProfileFormControls>; return f.controls.newPassword.value === f.controls.confirmPassword.value ? null : { mismatch: true }; }; } onSubmit(): void { if (this.editForm.invalid) return; this.isSubmitting = true; const formValue = this.editForm.getRawValue(); this.apiService.put('/users/me',{ newname: formValue.newname, newPassword: formValue.newPassword }).subscribe({ next: () => { this.dialogRef.close({ success: true, name: formValue.name }); }, error: (err) => { this.isSubmitting = false; console.error('Error updating profile:', err); } }); } }**

**Edit Profile Modal Component (Simplified Version)**

Based on your requirements, I'll modify the component to handle only the name and new password fields. Here's the updated version:

typescript

import { Component, Inject } from '@angular/core';

import { FormBuilder, FormGroup, Validators, ValidatorFn, FormControl } from '@angular/forms';

import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog';

import { ApiService } from '../../../shared/services/api.service';

interface ProfileFormValues {

name: string;

newPassword: string;

confirmPassword: string;

}

type ProfileFormControls = {

[key in keyof ProfileFormValues]: FormControl<ProfileFormValues[key]>;

};

@Component({

selector: 'app-edit-profile-modal',

templateUrl: './edit-profile-modal.component.html',

styleUrls: ['./edit-profile-modal.component.scss']

})

export class EditProfileModalComponent {

editForm: FormGroup<ProfileFormControls>;

isSubmitting = false;

constructor(

private fb: FormBuilder,

private apiService: ApiService,

private dialogRef: MatDialogRef<EditProfileModalComponent>,

@Inject(MAT\_DIALOG\_DATA) public data: { currentName: string }

) {

this.editForm = this.fb.group<ProfileFormControls>({

name: this.fb.control(this.data.currentName, {

nonNullable: true,

validators: [Validators.minLength(3)]

}),

newPassword: this.fb.control('', {

nonNullable: true,

validators: [Validators.minLength(6)]

}),

confirmPassword: this.fb.control('', { nonNullable: true })

}, { validators: this.passwordMatchValidator() });

}

private passwordMatchValidator(): ValidatorFn {

return (form: AbstractControl) => {

const f = form as FormGroup<ProfileFormControls>;

return f.controls.newPassword.value === f.controls.confirmPassword.value

? null

: { mismatch: true };

};

}

onSubmit(): void {

if (this.editForm.invalid) return;

this.isSubmitting = true;

const formValue = this.editForm.getRawValue();

this.apiService.put('/users/me', {

name: formValue.name,

password: formValue.newPassword

}).subscribe({

next: () => {

this.dialogRef.close({ success: true, name: formValue.name });

},

error: (err) => {

this.isSubmitting = false;

console.error('Error updating profile:', err);

}

});

}

}

**Key Changes Made:**

1. **Removed unnecessary fields**:
   * Removed currentPassword since you only need name and new password
   * Kept confirmPassword for password confirmation validation
2. **Simplified the interface**:
   * Updated ProfileFormValues to only include name, newPassword, and confirmPassword
3. **Updated API call**:
   * Changed the payload to match your API endpoint expectations (name and password instead of newname and newPassword)
   * Removed the unused AuthService since you're using ApiService
4. **Maintained all validation**:
   * Kept the minimum length validators
   * Maintained the password confirmation validator
5. **Improved type safety**:
   * Kept the strong typing with ProfileFormControls
   * Used nonNullable for all form controls

This version is cleaner and focuses only on the name and password update functionality you requested. The form will still validate that:

* Name has at least 3 characters
* New password has at least 6 characters
* New password matches the confirmation field

puedes quitarme la interfaz y dejar sin tipar?

**Edit Profile Modal Component (Untyped Version)**

Here's the simplified version without TypeScript interfaces and strong typing, focusing only on name and password fields:

typescript

import { Component, Inject } from '@angular/core';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog';

import { ApiService } from '../../../shared/services/api.service';

@Component({

selector: 'app-edit-profile-modal',

templateUrl: './edit-profile-modal.component.html',

styleUrls: ['./edit-profile-modal.component.scss']

})

export class EditProfileModalComponent {

editForm: FormGroup;

isSubmitting = false;

constructor(

private fb: FormBuilder,

private apiService: ApiService,

private dialogRef: MatDialogRef<EditProfileModalComponent>,

@Inject(MAT\_DIALOG\_DATA) public data: { currentName: string }

) {

this.editForm = this.fb.group({

name: [this.data.currentName, [Validators.minLength(3)]],

newPassword: ['', [Validators.minLength(6)]],

confirmPassword: ['']

}, { validator: this.passwordMatchValidator });

}

passwordMatchValidator(form: FormGroup) {

const newPassword = form.get('newPassword')?.value;

const confirmPassword = form.get('confirmPassword')?.value;

return newPassword === confirmPassword ? null : { mismatch: true };

}

onSubmit() {

if (this.editForm.invalid) return;

this.isSubmitting = true;

const formData = this.editForm.value;

this.apiService.put('/users/me', {

name: formData.name,

password: formData.newPassword

}).subscribe({

next: () => {

this.dialogRef.close({ success: true, name: formData.name });

},

error: (err) => {

this.isSubmitting = false;

console.error('Error updating profile:', err);

}

});

}

}

**Cambios realizados:**

1. **Eliminé las interfaces y tipos complejos**:
   * Removí ProfileFormValues y ProfileFormControls
   * Simplifiqué la declaración de editForm a FormGroup básico
2. **Volví al formato tradicional de FormBuilder**:
   * Usé el método array-style para los controles ([valor, validadores])
   * Cambié validators por validator en el validador del grupo
3. **Simplifiqué el validador de contraseñas**:
   * Lo convertí a función simple en lugar de ValidatorFn
   * Uso directo de form.get() para acceder a los controles
4. **Mantuve toda la funcionalidad**:
   * Validación de longitud mínima para nombre (3) y contraseña (6)
   * Comparación de contraseña y confirmación
   * Llamada al API con los datos correctos
5. **Mejoré el manejo de datos**:
   * Usé formData como nombre más descriptivo que formValue
   * Mantuve la estructura de respuesta del API

Esta versión es más sencilla y directa, manteniendo toda la funcionalidad pero sin el sistema de tipos avanzado de Angular.

**import { Component } from '@angular/core'; import { ApiService } from '../../shared/services/api.service'; import { AuthService } from '../../shared/services/auth.service'; @Component({ selector: 'app-account', standalone: false, templateUrl: './account.component.html', styleUrl: './account.component.scss' }) export class AccountComponent { registerData = { name: '', email: '', password: '' }; loginData = { email: '', password: '' }; passwordRequirements = { length: false, lowerCase: false, upperCase: false, number: false, specialChar: false }; passwordTouched = false; constructor( private apiService: ApiService, private authService: AuthService, ) {} onRegisterSubmit() { this.apiService.post(`users`,this.registerData).subscribe({ next: (response) => { console.log('Registro exitoso', response); alert('¡Registro exitoso!'); }, error: (error) => { console.error('Error en el registro', error); alert('Error en el registro: ' + (error.error?.message || 'Intente nuevamente')); } }); } onLoginSubmit() { this.authService.login(this.loginData).subscribe({ next: (response) => { console.log('Registro exitoso', response); alert(response.access\_token); }, error: (error) => { console.error('Error en el registro', error); alert('Error en el registro: ' + (error.error?.message || 'Intente nuevamente')); } }); } checkPasswordRequirements() { const pass = this.registerData.password; this.passwordRequirements = { length: pass?.length >= 8, lowerCase: /[a-z]/.test(pass), upperCase: /[A-Z]/.test(pass), number: /\d/.test(pass), specialChar: /[\W\_]/.test(pass) }; if (pass && pass.length > 0) { this.passwordTouched = true; } else { this.passwordTouched = false; } } } En este componente ya tengo los requerimentos. Hazlo así, por cierto, vuelve a tiparlo. Y a parte de eso, la contraseña nueva no debe repetirse es un solo campo y ya.**

**Edit Profile Modal Component (Typed Version)**

Here's the typed version of your component with password requirements checking, similar to your account component:

typescript

import { Component, Inject } from '@angular/core';

import { FormBuilder, FormGroup, Validators, AbstractControl } from '@angular/forms';

import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog';

import { ApiService } from '../../../shared/services/api.service';

interface PasswordRequirements {

length: boolean;

lowerCase: boolean;

upperCase: boolean;

number: boolean;

specialChar: boolean;

}

@Component({

selector: 'app-edit-profile-modal',

templateUrl: './edit-profile-modal.component.html',

styleUrls: ['./edit-profile-modal.component.scss']

})

export class EditProfileModalComponent {

editForm: FormGroup;

isSubmitting = false;

passwordRequirements: PasswordRequirements = {

length: false,

lowerCase: false,

upperCase: false,

number: false,

specialChar: false

};

passwordTouched = false;

constructor(

private fb: FormBuilder,

private apiService: ApiService,

private dialogRef: MatDialogRef<EditProfileModalComponent>,

@Inject(MAT\_DIALOG\_DATA) public data: { currentName: string }

) {

this.editForm = this.fb.group({

name: [this.data.currentName, [Validators.required, Validators.minLength(3)]],

newPassword: ['', [Validators.required]]

});

this.editForm.get('newPassword')?.valueChanges.subscribe(password => {

this.checkPasswordRequirements(password);

});

}

checkPasswordRequirements(password: string): void {

this.passwordRequirements = {

length: password?.length >= 8,

lowerCase: /[a-z]/.test(password),

upperCase: /[A-Z]/.test(password),

number: /\d/.test(password),

specialChar: /[\W\_]/.test(password)

};

this.passwordTouched = !!password && password.length > 0;

}

get passwordValid(): boolean {

return Object.values(this.passwordRequirements).every(req => req);

}

onSubmit(): void {

if (this.editForm.invalid || !this.passwordValid) return;

this.isSubmitting = true;

const formData = this.editForm.value;

this.apiService.put('/users/me', {

name: formData.name,

password: formData.newPassword

}).subscribe({

next: () => {

this.dialogRef.close({ success: true, name: formData.name });

},

error: (err) => {

this.isSubmitting = false;

console.error('Error updating profile:', err);

alert('Error updating profile: ' + (err.error?.message || 'Please try again'));

}

});

}

}

**Key Features:**

1. **Typed Password Requirements**: Added PasswordRequirements interface matching your account component
2. **Simplified Form**:
   * Only name and newPassword fields
   * Removed password confirmation field as requested
3. **Password Validation**:
   * Real-time password requirement checking
   * Same validation rules as your account component
   * Visual feedback for password strength
4. **Form Submission**:
   * Checks both form validity and password requirements
   * Shows error messages in similar format to your account component
5. **Improved UX**:
   * passwordTouched flag to show requirements only after interaction
   * passwordValid computed property to check all requirements
6. **Consistent API Call**:
   * Matches your existing API service pattern
   * Similar error handling

The template would need to display the password requirements indicators similar to your account component, showing which requirements are met as the user types.

Would you like me to provide the corresponding HTML template as well to match this component?

New chat

**no se me está incorporando el token am la consulta a la api dando 401 import { Component } from '@angular/core'; import { AuthService } from '../../shared/services/auth.service'; import { jwtDecode } from 'jwt-decode'; import { ApiService } from '../../shared/services/api.service'; @Component({ selector: 'app-profile', standalone: false, templateUrl: './profile.component.html', styleUrls: ['./profile.component.scss'] }) export class ProfileComponent { user = { id: 3, name: 'Alex García', email: 'alex@ejemplo.com', stats: { tasksCompleted: 0, tasksPending: 0, categories: 0 }, }; isLoading = true; errorMessage: string | null = null; constructor( private authService: AuthService, private apiService: ApiService ) {} ngOnInit(): void { this.loadUserData(); //this.loadStatistics(); } loadUserData(): void { this.isLoading = true; this.errorMessage = null; // Suponiendo que tienes un método para obtener los datos del usuario logueado const cut = this.authService.getDecodedToken(); // if (cut) { // this.user.id = cut.sub; // this.user.name = cut.name; // this.user.email = cut.email; // } // <- Actualmente desde el token podría sacarse todo //this.apiService.get(`users/id/${this.user.id}`) // <- Este endpoint sirve pero mejor usar el seguro this.apiService.get(`auth/getProfile`).subscribe({ next: (userDetails) => { this.user = { ...this.user, id: userDetails.id, name: userDetails.name, email: userDetails.email, } this.isLoading = false; }, error: (err) => { console.error('Error loading profile:', err); this.errorMessage = 'Error al cargar el perfil'; this.isLoading = false; } }) } logout(): void { this.authService.logout(); } } <div class="container py-5"> <div class="row justify-content-center"> <div class="col-lg-8"> <div class="card border-0 shadow-sm"> <div class="card-body p-4"> <!-- Header del perfil --> <div class="d-flex flex-column align-items-center text-center mb-4"> <div class="rounded-circle bg-primary bg-opacity-10 d-flex align-items-center justify-content-center mb-3" style="width: 120px; height: 120px;"> <i class="bi bi-person-fill text-primary" style="font-size: 3rem;"></i> </div> <h2 class="mb-1">{{ user.name }}</h2> <p class="text-muted mb-3">{{ user.email }}</p> <span class="badge bg-primary rounded-pill px-3">Usuario #{{ user.id }}</span> </div> <!-- Sección de información --> <div class="row g-3 mb-4"> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-person-lines-fill me-2 text-primary"></i> Información Básica </h3> <ul class="list-unstyled"> <li class="mb-2"> <strong>ID de usuario:</strong> #{{ user.id }} </li> <li class="mb-2"> <strong>Nombre:</strong> {{ user.name }} </li> <li class="mb-2"> <strong>Email:</strong> {{ user.email }} </li> </ul> </div> </div> </div> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-bar-chart-line-fill me-2 text-primary"></i> Tu Productividad </h3> <div class="d-flex flex-wrap gap-3"> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksCompleted }}</div> <small class="text-muted">Tareas completadas</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksPending }}</div> <small class="text-muted">Tareas pendientes</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.categories }}</div> <small class="text-muted">Categorías</small> </div> </div> </div> </div> </div> </div> <!-- Acciones --> <div class="d-flex justify-content-center gap-3"> <button class="btn btn-primary px-4"> <i class="bi bi-pencil-square me-2"></i> Editar Perfil </button> <button class="btn btn-outline-danger px-4" (click)="logout()"> <i class="bi bi-box-arrow-right me-2"></i> Cerrar Sesión </button> </div> </div> </div> </div> </div> </div> // jwt.interceptor.ts import { Injectable } from '@angular/core'; import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/common/http'; import { Observable } from 'rxjs'; import { AuthService } from '../services/auth.service'; @Injectable() export class JwtInterceptor implements HttpInterceptor { constructor(private authService: AuthService) {} intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> { const token = this.authService.getToken(); if (token) { request = request.clone({ setHeaders: { Authorization: `Bearer ${token}` } }); } return next.handle(request); } }**

**Problema con el Token en la API (Error 401)**

Parece que estás teniendo un problema con la autenticación en tu aplicación Angular. El error 401 indica que el token no se está incluyendo correctamente en la solicitud a la API o que el token es inválido.

**Posibles causas y soluciones:**

**1. Verifica si el interceptor está configurado correctamente**

Asegúrate de que el JwtInterceptor está correctamente proporcionado en tu módulo:

typescript

// En tu app.module.ts o core.module.ts

import { HTTP\_INTERCEPTORS } from '@angular/common/http';

@NgModule({

providers: [

{ provide: HTTP\_INTERCEPTORS, useClass: JwtInterceptor, multi: true }

]

})

**2. Verifica el método getToken() en AuthService**

Asegúrate de que getToken() en tu AuthService está devolviendo correctamente el token:

typescript

getToken(): string | null {

return localStorage.getItem('auth\_token'); // o donde almacenes el token

}

**3. Verifica el token en el localStorage**

Usa las herramientas de desarrollo del navegador para comprobar:

* Si el token existe en el localStorage
* Si el token no ha expirado

**4. Prueba manualmente la solicitud**

Puedes modificar temporalmente tu método loadUserData() para verificar el token:

typescript

loadUserData(): void {

this.isLoading = true;

this.errorMessage = null;

const token = this.authService.getToken();

console.log('Token:', token); // Verifica si el token existe

this.apiService.get(`auth/getProfile`).subscribe({

next: (userDetails) => {

// ...

},

error: (err) => {

console.error('Error completo:', err); // Verifica el error completo

if (err.status === 401) {

console.error('Token inválido o faltante');

// Puedes redirigir al login o intentar renovar el token

}

this.errorMessage = 'Error al cargar el perfil';

this.isLoading = false;

}

});

}

**5. Verifica las cabeceras de la solicitud**

En las herramientas de desarrollo del navegador (pestaña Network):

* Comprueba que la cabecera Authorization está presente en la solicitud
* Verifica que el token incluido es correcto

**6. Verifica el lado del servidor**

Asegúrate de que:

* El endpoint auth/getProfile requiere autenticación
* El formato del token es el esperado por el servidor
* No hay problemas con CORS que puedan estar bloqueando las cabeceras

**Solución temporal para diagnóstico**

Puedes probar a hacer la solicitud manualmente para diagnosticar el problema:

typescript

loadUserData(): void {

this.isLoading = true;

this.errorMessage = null;

const token = this.authService.getToken();

const headers = new HttpHeaders().set('Authorization', `Bearer ${token}`);

this.http.get(`tu-api-url/auth/getProfile`, { headers }).subscribe({

next: (userDetails) => {

// ...

},

error: (err) => {

console.error('Error manual:', err);

this.errorMessage = 'Error al cargar el perfil';

this.isLoading = false;

}

});

}

Si esto funciona pero el interceptor no, entonces el problema está en la configuración del interceptor. Si no funciona, el problema podría estar en el token mismo o en la configuración del servidor.

**Entonces tengo que meter el interceptor en cada módulo de cuyo componente quiera incorporar el token en las rutas protegidas?**

No, no es necesario incluir el interceptor en cada módulo. Angular tiene una jerarquía de inyección de dependencias que permite que los interceptores HTTP configurados en el módulo raíz (AppModule) sean aplicados a **todas las solicitudes HTTP de tu aplicación**, sin importar desde qué módulo o componente se originen.

**Solución correcta:**

1. **Configura el interceptor una sola vez** en el módulo principal (AppModule) o en un CoreModule (si usas módulos centralizados):

typescript

 // app.module.ts

import { HTTP\_INTERCEPTORS } from '@angular/common/http';

import { JwtInterceptor } from './core/interceptors/jwt.interceptor';

@NgModule({

providers: [

// Otros providers...

{ provide: HTTP\_INTERCEPTORS, useClass: JwtInterceptor, multi: true }

]

})

export class AppModule { }

 **Asegúrate de que el interceptor esté bien implementado**:

typescript

1. // jwt.interceptor.ts
2. intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {
3. const token = this.authService.getToken(); // Obtén el token desde tu AuthService
4. if (token) {
5. request = request.clone({
6. setHeaders: {
7. Authorization: `Bearer ${token}` // Agrega el token al header
8. }
9. });
10. }
11. return next.handle(request); // Continúa con la solicitud
12. }

**¿Por qué no es necesario repetirlo en otros módulos?**

* Angular aplica los interceptores definidos en el módulo raíz (AppModule) a **todas las solicitudes HTTP** de la aplicación.
* El parámetro multi: true permite que coexistan múltiples interceptores (por ejemplo, para manejo de errores, logging, etc.) sin sobrescribirse.

**Excepciones (casos especiales):**

Si hay **módulos lazy-loaded** (cargados perezosamente) y necesitas que el interceptor funcione también para sus solicitudes, debes:

1. **Exportar el interceptor en un CoreModule** y luego importar ese CoreModule en el módulo lazy-loaded.

typescript

1. // core.module.ts
2. @NgModule({
3. providers: [
4. { provide: HTTP\_INTERCEPTORS, useClass: JwtInterceptor, multi: true }
5. ]
6. })
7. export class CoreModule { }
8. **Importar CoreModule en el módulo lazy-loaded** (aunque esto no es lo ideal, ya que rompe la modularidad).

**Verificación adicional:**

* Usa las **DevTools del navegador** (pestaña *Network*) para confirmar que el header Authorization se envía en las solicitudes HTTP.
* Si el token no se adjunta, revisa:
  + Que el interceptor esté correctamente registrado en AppModule.
  + Que authService.getToken() retorne un token válido.
  + Que no haya otro interceptor que esté sobrescribiendo los headers.

**Conclusión:**

**No necesitas declarar el interceptor en cada módulo.** Basta con hacerlo una vez en el módulo raíz (AppModule) para que funcione en toda la aplicación. Si tienes un error 401, verifica:

1. El token existe y es válido.
2. El interceptor está bien configurado en el módulo principal.
3. El backend espera el token en el formato correcto (Bearer {token}).

Si el problema persiste, comparte cómo está estructurado tu módulo y cómo manejas el token en AuthService para ayudarte mejor. 🚀

**El token está, el problema creo que es que por alguna razón no se está incorporando el token a la solicitud. Failed to load resource: the server responded with a status of 401 (Unauthorized)Understand this error profile.component.ts:59 Error loading profile: HttpErrorResponseerror: {message: 'Unauthorized', statusCode: 401}headers: \_HttpHeaders {headers: undefined, normalizedNames: Map(0), lazyUpdate: null, lazyInit: ƒ}message: "Http failure response for http://localhost:3000/auth/getProfile: 401 Unauthorized"name: "HttpErrorResponse"ok: falsestatus: 401statusText: "Unauthorized"type: undefinedurl: "http://localhost:3000/auth/getProfile"[[Prototype]]: HttpResponseBase**

**Solución al Problema de Token no Incluido (Error 401)**

El error 401 indica claramente que el token no se está adjuntando a la solicitud HTTP, a pesar de que existe. Vamos a resolverlo paso a paso:

**1. Verificación del Interceptor**

Primero, asegurémonos que el interceptor está correctamente configurado:

typescript

// jwt.interceptor.ts

import { Injectable } from '@angular/core';

import {

HttpRequest,

HttpHandler,

HttpEvent,

HttpInterceptor

} from '@angular/common/http';

import { Observable } from 'rxjs';

import { AuthService } from '../services/auth.service';

@Injectable()

export class JwtInterceptor implements HttpInterceptor {

constructor(private authService: AuthService) {}

intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {

const token = this.authService.getToken();

console.log('Interceptor - Token:', token); // Debug

if (token) {

request = request.clone({

setHeaders: {

Authorization: `Bearer ${token}`

}

});

console.log('Request modificada:', request); // Debug

}

return next.handle(request);

}

}

**2. Registro del Interceptor**

Asegúrate que está registrado en el módulo principal:

typescript

// app.module.ts

import { HTTP\_INTERCEPTORS } from '@angular/common/http';

import { JwtInterceptor } from './core/interceptors/jwt.interceptor';

@NgModule({

providers: [

{

provide: HTTP\_INTERCEPTORS,

useClass: JwtInterceptor,

multi: true // ¡IMPORTANTE!

}

]

})

export class AppModule {}

**3. Verificación del AuthService**

Revisa tu servicio de autenticación:

typescript

// auth.service.ts

getToken(): string | null {

// Ejemplo común - ajusta según tu implementación

return localStorage.getItem('auth\_token');

}

getDecodedToken(): any {

const token = this.getToken();

return token ? jwtDecode(token) : null;

}

**4. Debugging Paso a Paso**

Agrega estos logs para diagnosticar:

typescript

// En tu componente

loadUserData(): void {

console.log('Token desde AuthService:', this.authService.getToken()); // Verifica token

this.apiService.get(`auth/getProfile`).subscribe({

next: (userDetails) => {

console.log('Respuesta exitosa:', userDetails);

},

error: (err) => {

console.error('Error completo:', err);

console.log('Request:', err.url); // Verifica URL

console.log('Headers enviados:', err.headers); // Verifica headers

}

});

}

**5. Verificación en Red**

Usa las DevTools (F12 → Network):

1. Busca la solicitud a /auth/getProfile
2. Verifica en la pestaña "Headers" si incluye:

text

1. Authorization: Bearer tu\_token

**Posibles Soluciones**

1. **Si el interceptor no se ejecuta**:
   * Verifica que no tienes otro interceptor que pueda estar interfiriendo
   * Asegúrate que el módulo donde está declarado el interceptor se importa correctamente
2. **Si el token no se adjunta**:
   * Verifica que authService.getToken() retorna el token correctamente
   * Comprueba que no hay errores de sincronización (el token podría no estar disponible cuando se hace la solicitud)
3. **Solución alternativa temporal**:

typescript

1. // En tu api.service.ts
2. get(url: string): Observable<any> {
3. const token = this.authService.getToken();
4. const headers = new HttpHeaders().set('Authorization', `Bearer ${token}`);
6. return this.http.get(url, { headers });
7. }

**Verificación Final**

1. ¿Ves los console.logs del interceptor?
2. ¿Aparece el header Authorization en la pestaña Network?
3. ¿El token que se muestra en los logs coincide con el almacenado en localStorage?

Si después de estas verificaciones el problema persiste, podríamos estar ante un problema de:

* Configuración de CORS en el backend
* Un interceptor que está modificando/eliminando los headers
* Un problema con el módulo donde se declara el interceptor

¿Podrías compartir:

1. La estructura de tu aplicación (cómo están organizados los módulos)?
2. El código de tu ApiService?
3. Si usas lazy loading para algún módulo?

**getToken(): string | null { const token = localStorage.getItem(this.TOKEN\_KEY); if (!token || this.isTokenExpired(token)) { this.logout(); return null; } return token } getDecodedToken(): any { const token = this.getToken() if (token == null) { return null } return jwtDecode(token); } import { NgModule } from '@angular/core'; import { BrowserModule } from '@angular/platform-browser'; import { AppRoutingModule } from './app-routing.module'; import { AppComponent } from './app.component'; import { LayoutModule } from './components/layout/layout/layout.module'; import { PagesModule } from './pages/pages.module'; import { provideHttpClient } from '@angular/common/http'; import { HTTP\_INTERCEPTORS } from '@angular/common/http'; import { JwtInterceptor } from './shared/interceptors/jwt.interceptor'; @NgModule({ declarations: [ AppComponent ], imports: [ BrowserModule, AppRoutingModule, LayoutModule, PagesModule, ], providers: [provideHttpClient(), { provide: HTTP\_INTERCEPTORS, useClass: JwtInterceptor, multi: true }], bootstrap: [AppComponent] }) export class AppModule { } // jwt.interceptor.ts import { Injectable } from '@angular/core'; import { HttpRequest, HttpHandler, HttpEvent, HttpInterceptor } from '@angular/common/http'; import { Observable } from 'rxjs'; import { AuthService } from '../services/auth.service'; @Injectable() export class JwtInterceptor implements HttpInterceptor { constructor(private authService: AuthService) {} intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> { const token = this.authService.getToken(); if (token) { request = request.clone({ setHeaders: { Authorization: `Bearer ${token}` } }); } return next.handle(request); } } import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/common'; import { AccountModule } from './account/account.module'; import { TasksModule } from './tasks/tasks.module'; import { HomeModule } from './home/home.module'; import { TutorialModule } from './tutorial/tutorial.module'; import { IncomingFeaturesModule } from './incoming-features/incoming-features.module'; @NgModule({ declarations: [], imports: [ AccountModule, TasksModule, HomeModule, TutorialModule, ProfileModule IncomingFeaturesModule, CommonModule, ], exports: [] }) export class PagesModule { } import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/common'; import { ProfileComponent } from './profile.component'; @NgModule({ declarations: [ProfileComponent], imports: [ CommonModule ], exports: [ProfileComponent] }) export class ProfileModule { } <div class="container py-5"> <div class="row justify-content-center"> <div class="col-lg-8"> <div class="card border-0 shadow-sm"> <div class="card-body p-4"> <!-- Header del perfil --> <div class="d-flex flex-column align-items-center text-center mb-4"> <div class="rounded-circle bg-primary bg-opacity-10 d-flex align-items-center justify-content-center mb-3" style="width: 120px; height: 120px;"> <i class="bi bi-person-fill text-primary" style="font-size: 3rem;"></i> </div> <h2 class="mb-1">{{ user.name }}</h2> <p class="text-muted mb-3">{{ user.email }}</p> <span class="badge bg-primary rounded-pill px-3">Usuario #{{ user.id }}</span> </div> <!-- Sección de información --> <div class="row g-3 mb-4"> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-person-lines-fill me-2 text-primary"></i> Información Básica </h3> <ul class="list-unstyled"> <li class="mb-2"> <strong>ID de usuario:</strong> #{{ user.id }} </li> <li class="mb-2"> <strong>Nombre:</strong> {{ user.name }} </li> <li class="mb-2"> <strong>Email:</strong> {{ user.email }} </li> </ul> </div> </div> </div> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-bar-chart-line-fill me-2 text-primary"></i> Tu Productividad </h3> <div class="d-flex flex-wrap gap-3"> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksCompleted }}</div> <small class="text-muted">Tareas completadas</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksPending }}</div> <small class="text-muted">Tareas pendientes</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.categories }}</div> <small class="text-muted">Categorías</small> </div> </div> </div> </div> </div> </div> <!-- Acciones --> <div class="d-flex justify-content-center gap-3"> <button class="btn btn-primary px-4"> <i class="bi bi-pencil-square me-2"></i> Editar Perfil </button> <button class="btn btn-outline-danger px-4" (click)="logout()"> <i class="bi bi-box-arrow-right me-2"></i> Cerrar Sesión </button> </div> </div> </div> </div> </div> </div> import { Component } from '@angular/core'; import { AuthService } from '../../shared/services/auth.service'; import { jwtDecode } from 'jwt-decode'; import { ApiService } from '../../shared/services/api.service'; @Component({ selector: 'app-profile', standalone: false, templateUrl: './profile.component.html', styleUrls: ['./profile.component.scss'] }) export class ProfileComponent { user = { id: 3, name: 'Alex García', email: 'alex@ejemplo.com', stats: { tasksCompleted: 0, tasksPending: 0, categories: 0 }, }; isLoading = true; errorMessage: string | null = null; constructor( private authService: AuthService, private apiService: ApiService ) {} ngOnInit(): void { this.loadUserData(); //this.loadStatistics(); } loadUserData(): void { this.isLoading = true; this.errorMessage = null; // Suponiendo que tienes un método para obtener los datos del usuario logueado const cut = this.authService.getDecodedToken(); // if (cut) { // this.user.id = cut.sub; // this.user.name = cut.name; // this.user.email = cut.email; // } // <- Actualmente desde el token podría sacarse todo //this.apiService.get(`users/id/${this.user.id}`) // <- Este endpoint sirve pero mejor usar el seguro this.apiService.get(`auth/getProfile`).subscribe({ next: (userDetails) => { this.user = { ...this.user, id: userDetails.id, name: userDetails.name, email: userDetails.email, } this.isLoading = false; }, error: (err) => { console.error('Error loading profile:', err); this.errorMessage = 'Error al cargar el perfil'; this.isLoading = false; } }) } logout(): void { this.authService.logout(); } }**

**Diagnóstico y Solución para el Problema del Token**

Basado en el código que compartiste, el problema parece estar en cómo se está configurando el HttpClient y el interceptor. Aquí está el análisis y la solución:

**Problema Identificado**

1. **Configuración del HttpClient**: Estás usando provideHttpClient() sin configurar los interceptores correctamente en Angular 15+.
2. **Registro del Interceptor**: Aunque has declarado el interceptor, la forma en que provees el HttpClient puede estar causando que no se aplique.

**Solución Paso a Paso**

**1. Modifica tu AppModule**

typescript

// app.module.ts

import { HttpClientModule, HTTP\_INTERCEPTORS } from '@angular/common/http';

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

AppRoutingModule,

LayoutModule,

PagesModule,

HttpClientModule, // Importa HttpClientModule aquí

],

providers: [

{ provide: HTTP\_INTERCEPTORS, useClass: JwtInterceptor, multi: true }

],

bootstrap: [AppComponent]

})

export class AppModule { }

**2. Verifica tu ApiService**

Asegúrate que tu ApiService esté correctamente implementado:

typescript

// api.service.ts

import { HttpClient } from '@angular/common/http';

@Injectable({

providedIn: 'root'

})

export class ApiService {

constructor(private http: HttpClient) {}

get(url: string): Observable<any> {

return this.http.get(url);

}

}

**3. Debugging Adicional en el Interceptor**

Agrega logs para verificar el flujo:

typescript

// jwt.interceptor.ts

intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {

console.log('Interceptor ejecutándose para URL:', request.url);

const token = this.authService.getToken();

console.log('Token obtenido:', token);

if (token) {

const modifiedRequest = request.clone({

setHeaders: {

Authorization: `Bearer ${token}`

}

});

console.log('Request modificada:', modifiedRequest);

return next.handle(modifiedRequest);

}

return next.handle(request);

}

**4. Verifica el Token en el Componente**

typescript

// profile.component.ts

ngOnInit(): void {

console.log('Token al iniciar componente:', this.authService.getToken());

this.loadUserData();

}

**Posibles Causas del Problema**

1. **Problema con provideHttpClient()**: En versiones recientes de Angular, provideHttpClient() necesita una configuración especial para interceptores.
2. **Lazy Loading**: Si tu módulo PagesModule o ProfileModule es lazy-loaded, podrías necesitar proveer el interceptor allí también.

**Solución Alternativa (si persiste el problema)**

Si el problema continúa, prueba esta alternativa:

1. **Crea un CoreModule**:

typescript

// core.module.ts

@NgModule({

providers: [

{ provide: HTTP\_INTERCEPTORS, useClass: JwtInterceptor, multi: true }

]

})

export class CoreModule {}

1. **Importa CoreModule en AppModule**:

typescript

// app.module.ts

imports: [

// otros imports

CoreModule

]

**Verificación Final**

1. Revisa la consola del navegador para ver los logs del interceptor
2. En las herramientas de desarrollo (F12 → Network):
   * Verifica que la solicitud a /auth/getProfile incluye el header Authorization
   * Comprueba que el token es el mismo que está en localStorage

**¿Por qué esto debería funcionar?**

* HttpClientModule proporciona correctamente el cliente HTTP con soporte para interceptores
* El interceptor está correctamente registrado a nivel de aplicación
* El token se verifica antes de adjuntarlo a la solicitud

Si después de estos cambios el problema persiste, podría ser necesario:

1. Verificar si hay múltiples instancias del HttpClient
2. Comprobar si hay otro interceptor que esté interfiriendo
3. Revisar la configuración del servidor para asegurar que acepta los headers CORS correctamente

**Importar el HttpClientModule en los imports me dice que está deprecado y vi que lo correcto era hacerlo así import { NgModule } from '@angular/core'; import { BrowserModule } from '@angular/platform-browser'; import { AppRoutingModule } from './app-routing.module'; import { AppComponent } from './app.component'; import { LayoutModule } from './components/layout/layout/layout.module'; import { PagesModule } from './pages/pages.module'; import { provideHttpClient } from '@angular/common/http'; import { HTTP\_INTERCEPTORS } from '@angular/common/http'; import { JwtInterceptor } from './shared/interceptors/jwt.interceptor'; @NgModule({ declarations: [ AppComponent ], imports: [ BrowserModule, AppRoutingModule, LayoutModule, PagesModule, ], providers: [provideHttpClient(), { provide: HTTP\_INTERCEPTORS, useClass: JwtInterceptor, multi: true }], bootstrap: [AppComponent] }) export class AppModule { } Que más tengo que verificar**

**Solución Definitiva para el Problema del Interceptor en Angular 15+**

Tienes razón que HttpClientModule está deprecado en favor de provideHttpClient(). El problema está en cómo se están combinando estas nuevas APIs con los interceptores. Aquí está la solución completa:

**Configuración Correcta con provideHttpClient()**

typescript

// app.module.ts

import { provideHttpClient, withInterceptorsFromDi } from '@angular/common/http';

@NgModule({

// ... otros imports

providers: [

provideHttpClient(

withInterceptorsFromDi() // <-- Esto es clave para usar interceptores DI

),

{

provide: HTTP\_INTERCEPTORS,

useClass: JwtInterceptor,

multi: true

}

]

})

export class AppModule { }

**Verificaciones Adicionales**

1. **Orden de los Providers**:
   * Asegúrate que provideHttpClient() está antes que los interceptores en el array de providers
2. **Standalone Components**:
   * Si usas componentes standalone, necesitas configurar el HttpClient diferente:

typescript

 // En routes o módulo principal

import { provideHttpClient, withInterceptors } from '@angular/common/http';

providers: [

provideHttpClient(

withInterceptors([jwtInterceptorFn]) // Interceptor funcional

)

]

 **Interceptor Funcional (Alternativa Moderna)**:

typescript

 // jwt.interceptor.fn.ts

export function jwtInterceptorFn(request: HttpRequest<unknown>, next: HttpHandlerFn) {

const token = inject(AuthService).getToken();

if (token) {

request = request.clone({

setHeaders: { Authorization: `Bearer ${token}` }

});

}

return next(request);

}

// app.config.ts

import { provideHttpClient, withInterceptors } from '@angular/common/http';

export const appConfig: ApplicationConfig = {

providers: [

provideHttpClient(withInterceptors([jwtInterceptorFn]))

]

};

 **Debugging Avanzado**:

* Agrega este código temporal para verificar el flujo:

typescript

 // En tu componente

constructor(private http: HttpClient) {

// Prueba directa sin ApiService

http.get('auth/getProfile').subscribe({

next: (res) => console.log('Respuesta directa:', res),

error: (err) => console.error('Error directo:', err)

});

}

 **Verifica el Token**:

* Asegúrate que no se está expirando justo antes de la solicitud:

typescript

1. // En tu AuthService
2. getToken(): string | null {
3. const token = localStorage.getItem(this.TOKEN\_KEY);
4. console.log('Token obtenido:', token);
5. console.log('¿Expirado?:', this.isTokenExpired(token));
6. return token && !this.isTokenExpired(token) ? token : null;
7. }

**Solución Definitiva**

1. **Para aplicaciones tradicionales (NgModule)**:

typescript

 providers: [

provideHttpClient(withInterceptorsFromDi()),

{ provide: HTTP\_INTERCEPTORS, useClass: JwtInterceptor, multi: true }

]

 **Para aplicaciones modernas (Standalone)**:

typescript

1. providers: [
2. provideHttpClient(withInterceptors([jwtInterceptorFn]))
3. ]

**¿Por qué esto funciona?**

* withInterceptorsFromDi() permite que los interceptores basados en DI funcionen con el nuevo provideHttpClient()
* Mantiene compatibilidad con el sistema de inyección de dependencias tradicional
* Es la forma oficial recomendada para migrar a las nuevas APIs

Si después de estos cambios sigues teniendo problemas, el error podría estar en:

1. El token realmente no existe cuando se hace la llamada
2. Hay algún middleware en el backend que está rechazando el header
3. Problemas de CORS que bloquean los headers personalizados

YA ESTÁ POR EL DI

**Solución Definitiva para el Interceptor con provideHttpClient()**

Entiendo que ya has configurado el interceptor mediante el sistema de Inyección de Dependencias (DI), pero persiste el problema. Vamos a solucionarlo definitivamente:

**Configuración 100% Funcional (Angular 15+)**

typescript

// app.module.ts

import { provideHttpClient, withInterceptorsFromDi } from '@angular/common/http';

@NgModule({

providers: [

// Configuración esencial:

provideHttpClient(

withInterceptorsFromDi() // Habilita interceptores DI

**YA ESTÁ QUE YA ESTÁ SOLUCIONADO ERA PORQUE LE FALTABA EL WITHINTERCEPTORSFROMDI. Por cierto eso qué hace para qué sirve**

¡Excelente que ya hayas solucionado el problema! Efectivamente, withInterceptorsFromDi() es la pieza clave que faltaba. Aquí te explico exactamente qué hace y por qué es necesaria:

**¿Qué hace withInterceptorsFromDi()?**

1. **Puente entre dos sistemas**:
   * Actúa como adaptador entre el nuevo sistema de provideHttpClient() y el tradicional sistema de interceptores basados en DI (inyección de dependencias).
2. **Habilita compatibilidad**:
   * Permite que los interceptores declarados clásicamente (con HTTP\_INTERCEPTORS) funcionen con el nuevo HttpClient.
3. **Mantiene la configuración existente**:
   * Sin esto, Angular ignora completamente los interceptores que registraste en los providers.

**¿Por qué fue necesario en tu caso?**

* **Nuevo vs Viejo**:

typescript

* // Viejo enfoque (Angular <15)
* imports: [HttpClientModule]
* // Nuevo enfoque (Angular 15+)
* providers: [provideHttpClient()]
* **Problema de compatibilidad**:
  + Cuando usas provideHttpClient() sin withInterceptorsFromDi(), Angular crea un HttpClient *sin soporte para interceptores DI*.

**¿Cuándo debes usarlo?**

Siempre que:

1. Uses provideHttpClient() (recomendado en Angular 15+)
2. Y tengas interceptores declarados tradicionalmente:

typescript

1. { provide: HTTP\_INTERCEPTORS, useClass: MiInterceptor, multi: true }

**Alternativa Moderna**

Si quieres evitar withInterceptorsFromDi(), puedes migrar a interceptores funcionales:

typescript

// Interceptor como función

export const jwtInterceptorFn = (req: HttpRequest<unknown>, next: HttpHandlerFn) => {

const token = inject(AuthService).getToken();

if (token) {

req = req.clone({ setHeaders: { Authorization: `Bearer ${token}` } });

}

return next(req);

};

// Configuración

providers: [

provideHttpClient(

withInterceptors([jwtInterceptorFn]) // Más moderno

)

]

**Conclusión**

withInterceptorsFromDi() es el "adaptador" que permite a tus interceptores tradicionales funcionar con el nuevo sistema de HttpClient. Sin él, Angular simplemente ignora tus interceptores, lo que explica por qué no se adjuntaba el token. ¡Buen trabajo detectando esta solución! 🚀

**<div class="container py-5"> <div class="row justify-content-center"> <div class="col-lg-8"> <div class="card border-0 shadow-sm"> <div class="card-body p-4"> <!-- Header del perfil --> <div class="d-flex flex-column align-items-center text-center mb-4"> <div class="rounded-circle bg-primary bg-opacity-10 d-flex align-items-center justify-content-center mb-3" style="width: 120px; height: 120px;"> <i class="bi bi-person-fill text-primary" style="font-size: 3rem;"></i> </div> <h2 class="mb-1">{{ user.name }}</h2> <p class="text-muted mb-3">{{ user.email }}</p> <span class="badge bg-primary rounded-pill px-3">Usuario #{{ user.id }}</span> </div> <!-- Sección de información --> <div class="row g-3 mb-4"> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-person-lines-fill me-2 text-primary"></i> Información Básica </h3> <ul class="list-unstyled"> <li class="mb-2"> <strong>Número de Usuario:</strong> {{ user.id }} </li> <li class="mb-2"> <strong>Nombre:</strong> {{ user.name }} </li> <li class="mb-2"> <strong>Email:</strong> {{ user.email }} </li> </ul> </div> </div> </div> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-bar-chart-line-fill me-2 text-primary"></i> Tu Productividad </h3> <div class="d-flex flex-wrap gap-3"> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksCompleted }}</div> <small class="text-muted">Tareas completadas</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksPending }}</div> <small class="text-muted">Tareas pendientes</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.categories }}</div> <small class="text-muted">Categorías</small> </div> </div> </div> </div> </div> </div> <!-- Acciones --> <div class="d-flex justify-content-center gap-3"> <button class="btn btn-primary px-4"> <i class="bi bi-pencil-square me-2"></i> Editar Perfil </button> <button class="btn btn-outline-danger px-4" (click)="logout()"> <i class="bi bi-box-arrow-right me-2"></i> Cerrar Sesión </button> </div> </div> </div> </div> </div> </div> import { Component } from '@angular/core'; import { AuthService } from '../../shared/services/auth.service'; import { jwtDecode } from 'jwt-decode'; import { ApiService } from '../../shared/services/api.service'; @Component({ selector: 'app-profile', standalone: false, templateUrl: './profile.component.html', styleUrls: ['./profile.component.scss'] }) export class ProfileComponent { user = { id: 3, name: 'Alex García', email: 'alex@ejemplo.com', stats: { tasksCompleted: 0, tasksPending: 0, categories: 0 }, }; isLoading = true; errorMessage: string | null = null; constructor( private authService: AuthService, private apiService: ApiService ) {} ngOnInit(): void { this.loadUserData(); //this.loadStatistics(); } loadUserData(): void { this.isLoading = true; this.errorMessage = null; // Suponiendo que tienes un método para obtener los datos del usuario logueado const cut = this.authService.getDecodedToken(); //if (cut) { // this.user.id = cut.sub; // this.user.name = cut.name; // this.user.email = cut.email; // } // <- Actualmente desde el token podría sacarse todo //this.apiService.get(`users/id/${this.user.id}`) // <- Este endpoint sirve pero mejor usar el seguro this.apiService.get(`auth/getProfile`).subscribe({ // <- Este endpoint actualmente saca la info del //propio JWT, lo que no es lo suyo para obtener más datos. Se puede modificar de cara a futuro next: (userDetails) => { this.user = { ...this.user, id: userDetails.sub, name: userDetails.name, email: userDetails.email, } this.isLoading = false; }, error: (err) => { console.error('Error loading profile:', err); this.errorMessage = 'Error al cargar el perfil'; this.isLoading = false; } }) } logout(): void { this.authService.logout(); } } Ahora quiero que el botón de editar perfil me abra un formulario a modo de ventana emergente en el que introducir los nuevos valores. A continuación, si se sale confirmando y se pasan las validaciones, se cambia el nombre, la contraseña o los dos (el usuario puede elegir y un campo puede ser vacío).**

**Implementación de Edición de Perfil con Modal**

Voy a guiarte para implementar un formulario de edición de perfil en modal con validaciones. Aquí está la solución completa:

**1. Primero, crea el componente del modal (puedes generarlo con Angular CLI)**

typescript

// edit-profile-modal.component.ts

import { Component, Inject } from '@angular/core';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog';

import { AuthService } from '../../shared/services/auth.service';

@Component({

selector: 'app-edit-profile-modal',

templateUrl: './edit-profile-modal.component.html',

styleUrls: ['./edit-profile-modal.component.scss']

})

export class EditProfileModalComponent {

editForm: FormGroup;

isSubmitting = false;

constructor(

private fb: FormBuilder,

private authService: AuthService,

private dialogRef: MatDialogRef<EditProfileModalComponent>,

@Inject(MAT\_DIALOG\_DATA) public data: { currentName: string }

) {

this.editForm = this.fb.group({

name: [this.data.currentName, [Validators.minLength(3)]],

currentPassword: [''],

newPassword: ['', [Validators.minLength(6)]],

confirmPassword: ['']

}, { validator: this.passwordMatchValidator });

}

passwordMatchValidator(form: FormGroup) {

const newPassword = form.get('newPassword')?.value;

const confirmPassword = form.get('confirmPassword')?.value;

return newPassword === confirmPassword ? null : { mismatch: true };

}

onSubmit() {

if (this.editForm.invalid) return;

this.isSubmitting = true;

const { name, currentPassword, newPassword } = this.editForm.value;

this.authService.updateProfile({ name, currentPassword, newPassword }).subscribe({

next: () => {

this.dialogRef.close({ success: true, name });

},

error: (err) => {

this.isSubmitting = false;

console.error('Error updating profile:', err);

}

});

}

}

**2. Template del modal (HTML)**

html

<!-- edit-profile-modal.component.html -->

<h2 mat-dialog-title>Editar Perfil</h2>

<mat-dialog-content>

<form [formGroup]="editForm" (ngSubmit)="onSubmit()">

<mat-form-field appearance="outline" class="w-100 mb-3">

<mat-label>Nombre</mat-label>

<input matInput formControlName="name">

<mat-error \*ngIf="editForm.get('name')?.hasError('minlength')">

El nombre debe tener al menos 3 caracteres

</mat-error>

</mat-form-field>

<h3 class="h5 mt-4 mb-3">Cambiar contraseña</h3>

<mat-form-field appearance="outline" class="w-100 mb-3">

<mat-label>Contraseña actual</mat-label>

<input matInput type="password" formControlName="currentPassword">

</mat-form-field>

<mat-form-field appearance="outline" class="w-100 mb-3">

<mat-label>Nueva contraseña</mat-label>

<input matInput type="password" formControlName="newPassword">

<mat-error \*ngIf="editForm.get('newPassword')?.hasError('minlength')">

La contraseña debe tener al menos 6 caracteres

</mat-error>

</mat-form-field>

<mat-form-field appearance="outline" class="w-100 mb-3">

<mat-label>Confirmar nueva contraseña</mat-label>

<input matInput type="password" formControlName="confirmPassword">

<mat-error \*ngIf="editForm.hasError('mismatch')">

Las contraseñas no coinciden

</mat-error>

</mat-form-field>

</form>

</mat-dialog-content>

<mat-dialog-actions align="end">

<button mat-button mat-dialog-close>Cancelar</button>

<button

mat-raised-button

color="primary"

(click)="onSubmit()"

[disabled]="isSubmitting || editForm.invalid">

<span \*ngIf="!isSubmitting">Guardar</span>

<mat-spinner \*ngIf="isSubmitting" diameter="20"></mat-spinner>

</button>

</mat-dialog-actions>

**3. Actualiza tu AuthService**

typescript

// auth.service.ts

updateProfile(data: { name?: string; currentPassword?: string; newPassword?: string }) {

return this.apiService.put('auth/update-profile', data);

}

**4. Modifica tu ProfileComponent**

typescript

// profile.component.ts

import { MatDialog } from '@angular/material/dialog';

import { EditProfileModalComponent } from './edit-profile-modal/edit-profile-modal.component';

// En la clase ProfileComponent:

constructor(

private authService: AuthService,

private apiService: ApiService,

private dialog: MatDialog

) {}

openEditDialog() {

const dialogRef = this.dialog.open(EditProfileModalComponent, {

width: '500px',

data: { currentName: this.user.name }

});

dialogRef.afterClosed().subscribe(result => {

if (result?.success) {

this.user.name = result.name;

// Opcional: Mostrar mensaje de éxito

}

});

}

**5. Actualiza el botón en tu template**

html

<!-- Cambia el botón de editar en profile.component.html -->

<button class="btn btn-primary px-4" (click)="openEditDialog()">

<i class="bi bi-pencil-square me-2"></i>

Editar Perfil

</button>

**6. Asegúrate de tener estos imports en tu módulo**

typescript

// En el módulo donde está ProfileComponent

import { MatDialogModule } from '@angular/material/dialog';

import { MatFormFieldModule } from '@angular/material/form-field';

import { MatInputModule } from '@angular/material/input';

import { ReactiveFormsModule } from '@angular/forms';

import { MatButtonModule } from '@angular/material/button';

@NgModule({

imports: [

// ...

MatDialogModule,

MatFormFieldModule,

MatInputModule,

ReactiveFormsModule,

MatButtonModule

]

})

**7. Implementación del backend (ejemplo conceptual)**

Tu endpoint auth/update-profile debería manejar:

typescript

// Ejemplo de controlador backend (Node.js/Express)

router.put('/update-profile', authenticate, async (req, res) => {

const { name, currentPassword, newPassword } = req.body;

const userId = req.user.id;

try {

if (newPassword) {

// Validar currentPassword

const user = await User.findById(userId);

const isValid = await bcrypt.compare(currentPassword, user.password);

if (!isValid) return res.status(400).json({ message: 'Contraseña actual incorrecta' });

// Actualizar contraseña

user.password = await bcrypt.hash(newPassword, 10);

}

if (name) {

user.name = name;

}

await user.save();

res.json({ success: true });

} catch (error) {

res.status(500).json({ message: 'Error al actualizar perfil' });

}

});

**Validaciones adicionales**

1. **Solo permitir cambios si hay al menos un campo válido**:

typescript

// En el modal component

get hasValidChanges(): boolean {

const { name, currentPassword, newPassword } = this.editForm.value;

return (!!name && name !== this.data.currentName) ||

(!!newPassword && !!currentPassword);

}

1. **Mejores mensajes de error**:

typescript

// En el servicio

updateProfile(data) {

return this.apiService.put('auth/update-profile', data).pipe(

catchError(err => {

let message = 'Error desconocido';

if (err.status === 400) message = err.error.message;

return throwError(() => new Error(message));

})

);

}

Esta implementación te dará:

* Un modal elegante para edición
* Validaciones en frontend y backend
* Cambios opcionales de nombre y/o contraseña
* Feedback visual durante la carga
* Manejo adecuado de errores

**import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog'; Cannot find module '@angular/material/dialog' or its corresponding type declarations**

**Solución para el Error del Módulo Angular Material**

El error ocurre porque no tienes instalado o importado correctamente los módulos de Angular Material necesarios para los diálogos. Veamos cómo solucionarlo:

**1. Instalar Angular Material (si no lo has hecho)**

Ejecuta en tu terminal:

bash

ng add @angular/material

Durante la instalación, selecciona un tema (por ejemplo, "Indigo/Pink") y responde "Yes" a las opciones de configuración.

**2. Instalar módulos específicos necesarios**

Necesitarás estos módulos para el diálogo y formularios:

bash

npm install @angular/material @angular/cdk @angular/animations

**3. Importar los módulos necesarios**

En tu módulo principal (generalmente app.module.ts) o en el módulo donde uses el diálogo:

typescript

import { MatDialogModule } from '@angular/material/dialog';

import { MatFormFieldModule } from '@angular/material/form-field';

import { MatInputModule } from '@angular/material/input';

import { MatButtonModule } from '@angular/material/button';

import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

@NgModule({

imports: [

// ... otros imports

BrowserAnimationsModule,

MatDialogModule,

MatFormFieldModule,

MatInputModule,

MatButtonModule,

FormsModule,

ReactiveFormsModule

]

})

export class AppModule { }

**4. Verificar versiones compatibles**

Asegúrate que las versiones sean compatibles con tu Angular:

json

// package.json

"dependencies": {

"@angular/material": "^15.2.0",

"@angular/cdk": "^15.2.0",

"@angular/animations": "^15.2.0"

// ... otras dependencias

}

**5. Solución alternativa si no quieres usar Angular Material**

Si prefieres no usar Angular Material, puedes implementar un modal con Bootstrap:

**a) Asegúrate de tener Bootstrap instalado:**

bash

npm install bootstrap @ng-bootstrap/ng-bootstrap

**b) Implementación alternativa:**

typescript

// edit-profile-modal.component.ts

import { NgbActiveModal } from '@ng-bootstrap/ng-bootstrap';

@Component({

// ... metadata

})

export class EditProfileModalComponent {

constructor(public activeModal: NgbActiveModal) {}

// Resto de tu código...

}

html

<!-- Template alternativo con Bootstrap -->

<div class="modal-header">

<h4 class="modal-title">Editar Perfil</h4>

<button type="button" class="close" (click)="activeModal.dismiss()">

<span>&times;</span>

</button>

</div>

<div class="modal-body">

<!-- Tu formulario aquí -->

</div>

<div class="modal-footer">

<button type="button" class="btn btn-outline-secondary" (click)="activeModal.dismiss()">Cancelar</button>

<button type="button" class="btn btn-primary" (click)="onSubmit()">Guardar</button>

</div>

**6. Importar módulos de Bootstrap (si eliges esta opción)**

En tu módulo:

typescript

import { NgbModule } from '@ng-bootstrap/ng-bootstrap';

@NgModule({

imports: [

NgbModule,

// ... otros imports

]

})

**Verificación final**

1. Revisa que todos los paquetes están instalados correctamente
2. Verifica que los módulos están importados en el módulo correcto
3. Asegúrate de que tu aplicación tiene incluidos los estilos de Angular Material o Bootstrap

¿Quieres que profundice en alguna de estas soluciones en particular?

**Y no se puede hacer sin esto?**

**Implementación de Modal sin Angular Material ni Bootstrap**

Sí, puedes crear un modal completamente personalizado sin depender de Angular Material ni Bootstrap. Aquí te muestro cómo hacerlo:

**1. Crear el componente del modal**

typescript

// edit-profile-modal.component.ts

import { Component, EventEmitter, Input, Output } from '@angular/core';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

@Component({

selector: 'app-edit-profile-modal',

templateUrl: './edit-profile-modal.component.html',

styleUrls: ['./edit-profile-modal.component.scss']

})

export class EditProfileModalComponent {

@Input() currentName: string = '';

@Output() save = new EventEmitter<any>();

@Output() close = new EventEmitter<void>();

editForm: FormGroup;

isVisible = false;

constructor(private fb: FormBuilder) {

this.editForm = this.fb.group({

name: ['', [Validators.minLength(3)]],

currentPassword: [''],

newPassword: ['', [Validators.minLength(6)]],

confirmPassword: ['']

}, { validator: this.passwordMatchValidator });

}

ngOnChanges() {

if (this.currentName) {

this.editForm.patchValue({ name: this.currentName });

}

}

passwordMatchValidator(form: FormGroup) {

const newPassword = form.get('newPassword')?.value;

const confirmPassword = form.get('confirmPassword')?.value;

return newPassword === confirmPassword ? null : { mismatch: true };

}

onSubmit() {

if (this.editForm.invalid) return;

this.save.emit(this.editForm.value);

}

onClose() {

this.close.emit();

}

}

**2. Template del modal personalizado**

html

<!-- edit-profile-modal.component.html -->

<div class="modal-overlay" \*ngIf="isVisible" (click)="onClose()">

<div class="modal-content" (click)="$event.stopPropagation()">

<div class="modal-header">

<h2>Editar Perfil</h2>

<button class="close-btn" (click)="onClose()">&times;</button>

</div>

<div class="modal-body">

<form [formGroup]="editForm" (ngSubmit)="onSubmit()">

<div class="form-group">

<label>Nombre</label>

<input type="text" formControlName="name">

<div class="error" \*ngIf="editForm.get('name')?.hasError('minlength')">

Mínimo 3 caracteres

</div>

</div>

<h3>Cambiar contraseña</h3>

<div class="form-group">

<label>Contraseña actual</label>

<input type="password" formControlName="currentPassword">

</div>

<div class="form-group">

<label>Nueva contraseña</label>

<input type="password" formControlName="newPassword">

<div class="error" \*ngIf="editForm.get('newPassword')?.hasError('minlength')">

Mínimo 6 caracteres

</div>

</div>

<div class="form-group">

<label>Confirmar nueva contraseña</label>

<input type="password" formControlName="confirmPassword">

<div class="error" \*ngIf="editForm.hasError('mismatch')">

Las contraseñas no coinciden

</div>

</div>

<div class="modal-footer">

<button type="button" class="btn-cancel" (click)="onClose()">Cancelar</button>

<button type="submit" class="btn-save" [disabled]="editForm.invalid">Guardar</button>

</div>

</form>

</div>

</div>

</div>

**3. Estilos CSS personalizados**

scss

/\* edit-profile-modal.component.scss \*/

.modal-overlay {

position: fixed;

top: 0;

left: 0;

right: 0;

bottom: 0;

background-color: rgba(0, 0, 0, 0.5);

display: flex;

justify-content: center;

align-items: center;

z-index: 1000;

}

.modal-content {

background: white;

border-radius: 8px;

width: 90%;

max-width: 500px;

max-height: 90vh;

overflow-y: auto;

box-shadow: 0 4px 20px rgba(0, 0, 0, 0.15);

}

.modal-header {

padding: 16px 24px;

border-bottom: 1px solid #eee;

display: flex;

justify-content: space-between;

align-items: center;

h2 {

margin: 0;

font-size: 1.5rem;

}

}

.close-btn {

background: none;

border: none;

font-size: 1.5rem;

cursor: pointer;

color: #666;

&:hover {

color: #333;

}

}

.modal-body {

padding: 24px;

}

.form-group {

margin-bottom: 16px;

label {

display: block;

margin-bottom: 8px;

font-weight: 500;

}

input {

width: 100%;

padding: 8px 12px;

border: 1px solid #ddd;

border-radius: 4px;

font-size: 1rem;

}

}

.error {

color: #d32f2f;

font-size: 0.875rem;

margin-top: 4px;

}

.modal-footer {

display: flex;

justify-content: flex-end;

gap: 12px;

padding-top: 16px;

margin-top: 16px;

border-top: 1px solid #eee;

}

.btn-cancel, .btn-save {

padding: 8px 16px;

border-radius: 4px;

cursor: pointer;

font-weight: 500;

}

.btn-cancel {

background: #f5f5f5;

border: 1px solid #ddd;

&:hover {

background: #e0e0e0;

}

}

.btn-save {

background: #1976d2;

color: white;

border: none;

&:hover {

background: #1565c0;

}

&:disabled {

background: #90caf9;

cursor: not-allowed;

}

}

**4. Modificar el ProfileComponent**

typescript

// profile.component.ts

export class ProfileComponent {

// ... código existente

showEditModal = false;

openEditDialog() {

this.showEditModal = true;

}

closeEditDialog() {

this.showEditModal = false;

}

handleProfileUpdate(updateData: any) {

this.authService.updateProfile(updateData).subscribe({

next: (response) => {

this.user.name = updateData.name || this.user.name;

this.closeEditDialog();

// Mostrar mensaje de éxito

},

error: (err) => {

console.error('Error updating profile:', err);

// Mostrar mensaje de error

}

});

}

}

**5. Actualizar el template principal**

html

<!-- profile.component.html -->

<!-- ... código existente ... -->

<button class="btn btn-primary px-4" (click)="openEditDialog()">

<i class="bi bi-pencil-square me-2"></i>

Editar Perfil

</button>

<app-edit-profile-modal

\*ngIf="showEditModal"

[currentName]="user.name"

(save)="handleProfileUpdate($event)"

(close)="closeEditDialog()">

</app-edit-profile-modal>

**6. Ventajas de esta solución**

1. **Sin dependencias externas**: No requiere Angular Material ni Bootstrap
2. **Totalmente personalizable**: Puedes modificar los estilos como desees
3. **Control completo**: Manejas toda la lógica del modal directamente
4. **Ligero**: No incluye código innecesario de librerías externas

**7. Consideraciones adicionales**

* **Animaciones**: Si quieres animaciones, puedes implementarlas con CSS puro
* **Accesibilidad**: Añade atributos ARIA para mejor accesibilidad
* **Responsive**: Los estilos proporcionados ya son responsive

Esta implementación te da total libertad y control sobre el comportamiento y apariencia del modal, sin necesidad de agregar dependencias adicionales a tu proyecto.

**Tarda mucho en instalar el angular material, se habrá quedado pillado?**

**Solución para la instalación lenta de Angular Material**

Es normal que la instalación de Angular Material pueda tardar varios minutos, pero si parece estar "pillado", aquí te indico cómo proceder:

**¿Se ha quedado realmente pillado?**

1. **Tiempos normales de instalación**:
   * Dependiendo de tu conexión y equipo, puede tardar entre 2-10 minutos
   * La primera instalación siempre es más lenta porque descarga muchos paquetes
2. **Señales de que está funcionando**:
   * El cursor parpadea en la terminal
   * Hay actividad en el disco (lucecita del HDD/SSD)
   * No aparece ningún mensaje de error

**Qué hacer si está tardando demasiado**

**Opción 1: Esperar un poco más**

Dale hasta 15 minutos antes de asumir que está bloqueado.

**Opción 2: Interrumpir y reinstalar**

1. Presiona Ctrl + C en la terminal para cancelar
2. Ejecuta estos comandos secuencialmente:

bash

# Limpiar cache de npm

npm cache clean --force

# Eliminar node\_modules y package-lock.json

rm -rf node\_modules package-lock.json

# Instalar solo dependencias básicas primero

npm install @angular/core @angular/common @angular/forms

# Luego instalar Angular Material específicamente

ng add @angular/material

**Opción 3: Instalación manual más rápida**

Si el asistente ng add falla, instala manualmente:

bash

# Instalar paquetes esenciales sin el asistente

npm install @angular/material @angular/cdk @angular/animations

# Luego importa manualmente los módulos necesarios en tu AppModule

**Alternativa definitiva: Usar tu modal personalizado**

Si prefieres no esperar, puedes implementar completamente el modal sin Angular Material como te mostré anteriormente, que tiene varias ventajas:

1. **Más ligero**: Sin dependencias adicionales
2. **Más rápido**: No requiere instalación pesada
3. **Total control**: Diseño 100% personalizable

**Verificación posterior a instalación**

Si finalmente se completa la instalación, verifica que todo está correcto con:

bash

npm list @angular/material @angular/cdk @angular/animations

Esto debería mostrar

**vale ya está me pregunta por temas. Azure/Blue, Rose/Red, Magenta/Violet y Cyan/Orange o Custom, qué escojo?**

**Server busy, please try again later.**

**set up global angular material typography styles?**

**Configuración de Estilos Tipográficos Globales en Angular Material**

Para configurar los estilos tipográficos globales de Angular Material, sigue estos pasos:

**1. Habilitar tipografía global**

En tu archivo styles.scss principal (o el archivo CSS/SCSS global de tu aplicación), añade:

scss

@use '@angular/material' as mat;

// Incluir los estilos tipográficos comunes

@include mat.core();

// Definir la tipografía

$custom-typography: mat.define-typography-config(

$font-family: 'Roboto, "Helvetica Neue", sans-serif',

$headline-1: mat.define-typography-level(96px, 96px, 300),

$headline-2: mat.define-typography-level(60px, 60px, 300),

// Puedes personalizar todos los niveles tipográficos aquí

$body-1: mat.define-typography-level(16px, 24px, 400)

);

// Aplicar la tipografía personalizada

@include mat.core($custom-typography);

**Estoy instalando angular material y me hace esa pregunta le digo yes o no**

**Cuando el instalador de Angular Material te pregunta:**

**"Set up global Angular Material typography styles?"**

**¿Qué significa esto?**

Angular Material incluye una tipografía predefinida basada en las [Material Design Guidelines](https://m2.material.io/design/typography). Si respondes **"Yes"**, hará lo siguiente:

1. **Aplicará estilos tipográficos globales** a tu aplicación (como font-family, tamaños de encabezados, párrafos, etc.).
2. **Inyectará automáticamente** estos estilos en tu styles.scss/styles.css:

css

1. html, body {
2. font-family: 'Roboto', sans-serif; /\* Fuente predeterminada \*/
3. }
4. h1, h2, h3, h4, h5, h6 {
5. margin: 0; /\* Reset de márgenes \*/
6. }
7. /\* ...otros estilos tipográficos \*/
8. **Asegurará consistencia visual** en toda tu app según Material Design.

**¿Qué responder?**

✅ **"Yes"** (Recomendado) si:  
✔ Quieres que Angular Material gestione automáticamente la tipografía.  
✔ Te gusta la fuente Roboto y los estilos predeterminados de Material.  
✔ No tienes una guía de estilos personalizada.

❌ **"No"** si:  
✖ Ya tienes una tipografía personalizada (ej. otra fuente como 'Inter', 'Open Sans').  
✖ Quieres definir manualmente los estilos tipográficos.

**Si eliges "Yes" pero luego quieres personalizar**

Puedes sobrescribir los estilos en tu archivo global styles.scss:

scss

// Cambiar la fuente global después de instalar

body {

font-family: 'Tu-Fuente-Personalizada', sans-serif;

}

**Conclusión**

Si es tu primera vez con Angular Material, **elige "Yes"** para tener una base tipográfica consistente. Si ya tienes diseños personalizados, elige "No" y configura la tipografía manualmente.

¿Necesitas ayuda para personalizarla después? ¡Déjame saber! 🚀

**// edit-profile-modal.component.ts import { Component, Inject } from '@angular/core'; import { FormBuilder, FormGroup, Validators } from '@angular/forms'; import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog'; import { AuthService } from '../../../shared/services/auth.service'; @Component({ selector: 'app-edit-profile-modal', templateUrl: './edit-profile-modal.component.html', styleUrls: ['./edit-profile-modal.component.scss'] }) export class EditProfileModalComponent { editForm: FormGroup; isSubmitting = false; constructor( private fb: FormBuilder, private authService: AuthService, private dialogRef: MatDialogRef<EditProfileModalComponent>, @Inject(MAT\_DIALOG\_DATA) public data: { currentName: string } ) { this.editForm = this.fb.group({ name: [this.data.currentName, [Validators.minLength(3)]], currentPassword: [''], newPassword: ['', [Validators.minLength(6)]], confirmPassword: [''] }, { validator: this.passwordMatchValidator }); } passwordMatchValidator(form: FormGroup) { const newPassword = form.get('newPassword')?.value; const confirmPassword = form.get('confirmPassword')?.value; return newPassword === confirmPassword ? null : { mismatch: true }; } onSubmit() { if (this.editForm.invalid) return; this.isSubmitting = true; const { name, currentPassword, newPassword } = this.editForm.value; this.authService.updateProfile({ name, currentPassword, newPassword }).subscribe({ next: () => { this.dialogRef.close({ success: true, name }); }, error: (err) => { this.isSubmitting = false; console.error('Error updating profile:', err); } }); } } this.fb.group me dice que group está deprecado**

**Server busy, please try again later.**

**<!-- edit-profile-modal.component.html --> <div class="modal-overlay" \*ngIf="isVisible" (click)="onClose()"> <div class="modal-content" (click)="$event.stopPropagation()"> <div class="modal-header"> <h2>Editar Perfil</h2> <button class="close-btn" (click)="onClose()">&times;</button> </div> <div class="modal-body"> <form [formGroup]="editForm" (ngSubmit)="onSubmit()"> <div class="form-group"> <label>Nombre</label> <input type="text" formControlName="name"> <div class="error" \*ngIf="editForm.get('name')?.hasError('minlength')"> Mínimo 3 caracteres </div> </div> <h3>Cambiar contraseña</h3> <div class="form-group"> <label>Contraseña actual</label> <input type="password" formControlName="currentPassword"> </div> <div class="form-group"> <label>Nueva contraseña</label> <input type="password" formControlName="newPassword"> <div class="error" \*ngIf="editForm.get('newPassword')?.hasError('minlength')"> Mínimo 6 caracteres </div> </div> <div class="form-group"> <label>Confirmar nueva contraseña</label> <input type="password" formControlName="confirmPassword"> <div class="error" \*ngIf="editForm.hasError('mismatch')"> Las contraseñas no coinciden </div> </div> <div class="modal-footer"> <button type="button" class="btn-cancel" (click)="onClose()">Cancelar</button> <button type="submit" class="btn-save" [disabled]="editForm.invalid">Guardar</button> </div> </form> </div> </div> </div> De aquí el onclose y el editform se me iluminan en rojo import { Component, Inject } from '@angular/core'; import { FormBuilder, FormGroup, Validators, AbstractControl } from '@angular/forms'; import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog'; import { ApiService } from '../../../shared/services/api.service'; interface PasswordRequirements { length: boolean; lowerCase: boolean; upperCase: boolean; number: boolean; specialChar: boolean; } @Component({ selector: 'app-edit-profile-modal', templateUrl: './edit-profile-modal.component.html', styleUrls: ['./edit-profile-modal.component.scss'] }) export class EditProfileModalComponent { editForm: FormGroup; isSubmitting = false; passwordRequirements: PasswordRequirements = { length: false, lowerCase: false, upperCase: false, number: false, specialChar: false }; passwordTouched = false; constructor( private fb: FormBuilder, private apiService: ApiService, private dialogRef: MatDialogRef<EditProfileModalComponent>, @Inject(MAT\_DIALOG\_DATA) public data: { currentName: string } ) { this.editForm = this.fb.group({ name: [this.data.currentName, [Validators.required, Validators.minLength(3)]], newPassword: ['', [Validators.required]] }); this.editForm.get('newPassword')?.valueChanges.subscribe(password => { this.checkPasswordRequirements(password); }); } checkPasswordRequirements(password: string): void { this.passwordRequirements = { length: password?.length >= 8, lowerCase: /[a-z]/.test(password), upperCase: /[A-Z]/.test(password), number: /\d/.test(password), specialChar: /[\W\_]/.test(password) }; this.passwordTouched = !!password && password.length > 0; } get passwordValid(): boolean { return Object.values(this.passwordRequirements).every(req => req); } onSubmit(): void { if (this.editForm.invalid || !this.passwordValid) return; this.isSubmitting = true; const formData = this.editForm.value; this.apiService.put('/users/me', { name: formData.name, password: formData.newPassword }).subscribe({ next: () => { this.dialogRef.close({ success: true, name: formData.name }); }, error: (err) => { this.isSubmitting = false; console.error('Error updating profile:', err); alert('Error updating profile: ' + (err.error?.message || 'Please try again')); } }); } } Qué tengo qué hacer?**

**Solución para los Errores en el Modal de Edición de Perfil**

Los errores que estás viendo ocurren porque hay una discrepancia entre tu template HTML y tu componente TypeScript. Vamos a corregirlo:

**Problemas Identificados**

1. **onClose no existe** en tu componente (pero sí en el template)
2. **currentPassword y confirmPassword** están en el template pero no en el FormGroup
3. **Validación de contraseña** está implementada de forma diferente en ambos lugares

**Solución Completa**

**1. Actualiza tu Componente TypeScript**

typescript

import { Component, Inject } from '@angular/core';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog';

import { ApiService } from '../../../shared/services/api.service';

interface PasswordRequirements {

length: boolean;

lowerCase: boolean;

upperCase: boolean;

number: boolean;

specialChar: boolean;

}

@Component({

selector: 'app-edit-profile-modal',

templateUrl: './edit-profile-modal.component.html',

styleUrls: ['./edit-profile-modal.component.scss']

})

export class EditProfileModalComponent {

editForm: FormGroup;

isSubmitting = false;

passwordRequirements: PasswordRequirements = {

length: false,

lowerCase: false,

upperCase: false,

number: false,

specialChar: false

};

passwordTouched = false;

constructor(

private fb: FormBuilder,

private apiService: ApiService,

private dialogRef: MatDialogRef<EditProfileModalComponent>,

@Inject(MAT\_DIALOG\_DATA) public data: { currentName: string }

) {

this.editForm = this.fb.group({

name: [this.data.currentName, [Validators.required, Validators.minLength(3)]],

currentPassword: ['', [Validators.required]],

newPassword: ['', [Validators.required, Validators.minLength(8)]],

confirmPassword: ['', [Validators.required]]

}, { validator: this.passwordMatchValidator });

this.editForm.get('newPassword')?.valueChanges.subscribe(password => {

this.checkPasswordRequirements(password);

});

}

// Añade este método para cerrar el modal

onClose(): void {

this.dialogRef.close();

}

passwordMatchValidator(form: FormGroup) {

const newPassword = form.get('newPassword')?.value;

const confirmPassword = form.get('confirmPassword')?.value;

return newPassword === confirmPassword ? null : { mismatch: true };

}

checkPasswordRequirements(password: string): void {

this.passwordRequirements = {

length: password?.length >= 8,

lowerCase: /[a-z]/.test(password),

upperCase: /[A-Z]/.test(password),

number: /\d/.test(password),

specialChar: /[\W\_]/.test(password)

};

this.passwordTouched = !!password && password.length > 0;

}

get passwordValid(): boolean {

return Object.values(this.passwordRequirements).every(req => req);

}

onSubmit(): void {

if (this.editForm.invalid || !this.passwordValid) return;

this.isSubmitting = true;

const formData = this.editForm.value;

// Verifica que las contraseñas coincidan

if (formData.newPassword !== formData.confirmPassword) {

alert('Las contraseñas no coinciden');

this.isSubmitting = false;

return;

}

this.apiService.put('/users/me', {

name: formData.name,

currentPassword: formData.currentPassword,

newPassword: formData.newPassword

}).subscribe({

next: () => {

this.dialogRef.close({ success: true, name: formData.name });

},

error: (err) => {

this.isSubmitting = false;

console.error('Error updating profile:', err);

alert('Error al actualizar perfil: ' + (err.error?.message || 'Por favor, inténtalo de nuevo'));

}

});

}

}

**2. Actualiza tu Template HTML**

html

<div class="modal-overlay" \*ngIf="true" (click)="onClose()">

<div class="modal-content" (click)="$event.stopPropagation()">

<div class="modal-header">

<h2>Editar Perfil</h2>

<button class="close-btn" (click)="onClose()">&times;</button>

</div>

<div class="modal-body">

<form [formGroup]="editForm" (ngSubmit)="onSubmit()">

<div class="form-group">

<label>Nombre</label>

<input type="text" formControlName="name">

<div class="error" \*ngIf="editForm.get('name')?.hasError('required')">

El nombre es requerido

</div>

<div class="error" \*ngIf="editForm.get('name')?.hasError('minlength')">

Mínimo 3 caracteres

</div>

</div>

<h3>Cambiar contraseña</h3>

<div class="form-group">

<label>Contraseña actual</label>

<input type="password" formControlName="currentPassword">

<div class="error" \*ngIf="editForm.get('currentPassword')?.hasError('required')">

La contraseña actual es requerida

</div>

</div>

<div class="form-group">

<label>Nueva contraseña</label>

<input type="password" formControlName="newPassword">

<div class="error" \*ngIf="editForm.get('newPassword')?.hasError('required')">

La nueva contraseña es requerida

</div>

<div class="error" \*ngIf="editForm.get('newPassword')?.hasError('minlength')">

Mínimo 8 caracteres

</div>

<!-- Indicadores de fortaleza de contraseña -->

<div class="password-strength" \*ngIf="passwordTouched">

<div [class.valid]="passwordRequirements.length">✓ 8+ caracteres</div>

<div [class.valid]="passwordRequirements.lowerCase">✓ Minúscula</div>

<div [class.valid]="passwordRequirements.upperCase">✓ Mayúscula</div>

<div [class.valid]="passwordRequirements.number">✓ Número</div>

<div [class.valid]="passwordRequirements.specialChar">✓ Carácter especial</div>

</div>

</div>

<div class="form-group">

<label>Confirmar nueva contraseña</label>

<input type="password" formControlName="confirmPassword">

<div class="error" \*ngIf="editForm.get('confirmPassword')?.hasError('required')">

Confirma tu nueva contraseña

</div>

<div class="error" \*ngIf="editForm.hasError('mismatch')">

Las contraseñas no coinciden

</div>

</div>

<div class="modal-footer">

<button type="button" class="btn-cancel" (click)="onClose()">Cancelar</button>

<button type="submit" class="btn-save" [disabled]="editForm.invalid || !passwordValid">

<span \*ngIf="!isSubmitting">Guardar</span>

<span \*ngIf="isSubmitting">Guardando...</span>

</button>

</div>

</form>

</div>

</div>

</div>

**3. Añade estos estilos CSS**

scss

.password-strength {

margin-top: 10px;

div {

color: #ff0000;

font-size: 0.8rem;

margin: 2px 0;

&.valid {

color: #00aa00;

}

}

}

.btn-save:disabled {

opacity: 0.6;

cursor: not-allowed;

}

**Cambios Clave Realizados:**

1. **Añadido onClose()** al componente para cerrar el modal
2. **Incluidos todos los campos** del formulario en el FormGroup
3. **Mejorada la validación** de contraseña con indicadores visuales
4. **Sincronización completa** entre template y componente
5. **Mensajes de error** más descriptivos
6. **Estado de envío** (loading) en el botón Guardar

Con estos cambios, los errores deberían desaparecer y el formulario funcionará correctamente con todas las validaciones necesarias.

**import { Component, Inject } from '@angular/core'; import { FormBuilder, FormGroup, Validators, AbstractControl } from '@angular/forms'; import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog'; import { ApiService } from '../../../shared/services/api.service'; interface PasswordRequirements { length: boolean; lowerCase: boolean; upperCase: boolean; number: boolean; specialChar: boolean; } @Component({ selector: 'app-edit-profile-modal', templateUrl: './edit-profile-modal.component.html', styleUrls: ['./edit-profile-modal.component.scss'] }) export class EditProfileModalComponent { editForm: FormGroup; isSubmitting = false; passwordRequirements: PasswordRequirements = { length: false, lowerCase: false, upperCase: false, number: false, specialChar: false }; passwordTouched = false; constructor( private fb: FormBuilder, private apiService: ApiService, private dialogRef: MatDialogRef<EditProfileModalComponent>, @Inject(MAT\_DIALOG\_DATA) public data: { currentName: string } ) { this.editForm = this.fb.group({ name: [this.data.currentName, [Validators.required, Validators.minLength(3)]], newPassword: ['', [Validators.required]] }); this.editForm.get('newPassword')?.valueChanges.subscribe(password => { this.checkPasswordRequirements(password); }); } checkPasswordRequirements(password: string): void { this.passwordRequirements = { length: password?.length >= 8, lowerCase: /[a-z]/.test(password), upperCase: /[A-Z]/.test(password), number: /\d/.test(password), specialChar: /[\W\_]/.test(password) }; this.passwordTouched = !!password && password.length > 0; } get passwordValid(): boolean { return Object.values(this.passwordRequirements).every(req => req); } onSubmit(): void { if (this.editForm.invalid || !this.passwordValid) return; this.isSubmitting = true; const formData = this.editForm.value; this.apiService.put('/users/me', { name: formData.name, password: formData.newPassword }).subscribe({ next: () => { this.dialogRef.close({ success: true, name: formData.name }); }, error: (err) => { this.isSubmitting = false; console.error('Error updating profile:', err); alert('Error updating profile: ' + (err.error?.message || 'Please try again')); } }); } onClose(): void { this.dialogRef.close(); } } <!-- edit-profile-modal.component.html --> <div class="modal-overlay" \*ngIf="isVisible" (click)="onClose()"> <div class="modal-content" (click)="$event.stopPropagation()"> <div class="modal-header"> <h2>Editar Perfil</h2> <button class="close-btn" (click)="onClose()">&times;</button> </div> <div class="modal-body"> <form [formGroup]="editForm" (ngSubmit)="onSubmit()"> <div class="form-group"> <label>Nombre</label> <input type="text" formControlName="name"> <div class="error" \*ngIf="editForm.get('name')?.hasError('minlength')"> Mínimo 3 caracteres </div> </div> <h3>Cambiar contraseña</h3> <div class="form-group"> <label>Contraseña actual</label> <input type="password" formControlName="currentPassword"> </div> <div class="form-group"> <label>Nueva contraseña</label> <input type="password" formControlName="newPassword"> <div class="error" \*ngIf="editForm.get('newPassword')?.hasError('minlength')"> Mínimo 6 caracteres </div> </div> <div class="form-group"> <label>Confirmar nueva contraseña</label> <input type="password" formControlName="confirmPassword"> <div class="error" \*ngIf="editForm.hasError('mismatch')"> Las contraseñas no coinciden </div> </div> <div class="modal-footer"> <button type="button" class="btn-cancel" (click)="onClose()">Cancelar</button> <button type="submit" class="btn-save" [disabled]="editForm.invalid">Guardar</button> </div> </form> </div> </div> </div> por qué editForm sigue iluminandose en rojo?**

El problema de que editForm aparezca en rojo (error) en tu template HTML mientras que está correctamente definido en el componente se debe a una **inconsistencia entre las propiedades del FormGroup en el TypeScript y el template HTML**. Analicemos el problema y la solución:

**Causa del Error**

1. **Campos faltantes en el FormGroup**:
   * En tu template HTML usas formControlName="currentPassword" y formControlName="confirmPassword"
   * Pero en tu componente TypeScript solo defines name y newPassword en el FormGroup
2. **Validator faltante**:
   * En el template verificas editForm.hasError('mismatch') (para comparar contraseñas)
   * Pero no has definido el validador passwordMatchValidator en el componente

**Solución Actualizada**

**1. Actualiza el FormGroup en el componente:**

typescript

this.editForm = this.fb.group({

name: [this.data.currentName, [Validators.required, Validators.minLength(3)]],

currentPassword: ['', [Validators.required]], // Añade este campo

newPassword: ['', [Validators.required, Validators.minLength(8)]], // Cambié a 8 caracteres

confirmPassword: ['', [Validators.required]] // Añade este campo

}, {

validator: this.passwordMatchValidator // Añade el validador grupal

});

**2. Añade el validador de coincidencia de contraseñas:**

typescript

passwordMatchValidator(form: FormGroup) {

const newPassword = form.get('newPassword')?.value;

const confirmPassword = form.get('confirmPassword')?.value;

return newPassword === confirmPassword ? null : { mismatch: true };

}

**3. Actualiza el método onSubmit:**

typescript

onSubmit(): void {

if (this.editForm.invalid || !this.passwordValid) return;

// Verifica coincidencia de contraseñas

if (this.editForm.hasError('mismatch')) {

alert('Las contraseñas no coinciden');

return;

}

// Resto del código...

}

**Explicación de los Cambios**

1. **Consistencia de campos**:
   * Ahora el FormGroup contiene todos los campos usados en el template
   * Eliminarás el error porque Angular encontrará los controles referenciados
2. **Validación mejorada**:
   * El passwordMatchValidator compara ambas contraseñas
   * Se activa automáticamente al cambiar cualquier campo
3. **Mensajes de error**:
   * El validador establecerá el error mismatch que verificas en el template

**Consejo Adicional**

Si usas VS Code, instala la extensión **Angular Language Service** para mejor soporte de templates Angular. Esto te ayudará a detectar estos errores antes.

bash

code --install-extension Angular.ng-template

Con estos cambios, el error de editForm en rojo debería desaparecer y el formulario funcionará correctamente con todas las validaciones.

**<!-- edit-profile-modal.component.html --> <div class="modal-overlay" \*ngIf="isVisible" (click)="onClose()"> <div class="modal-content" (click)="$event.stopPropagation()"> <div class="modal-header"> <h2>Editar Perfil</h2> <button class="close-btn" (click)="onClose()">&times;</button> </div> <div class="modal-body"> <form [formGroup]="editForm" (ngSubmit)="onSubmit()"> <div class="form-group"> <label>Nombre</label> <input type="text" formControlName="name"> <div class="error" \*ngIf="editForm.get('name')?.hasError('minlength')"> Mínimo 3 caracteres </div> </div> <h3>Cambiar contraseña</h3> <div class="form-group"> <label>Contraseña actual</label> <input type="password" formControlName="currentPassword"> </div> <div class="form-group"> <label>Nueva contraseña</label> <input type="password" formControlName="newPassword"> <div class="error" \*ngIf="editForm.get('newPassword')?.hasError('minlength')"> Mínimo 6 caracteres </div> </div> <div class="form-group"> <label>Confirmar nueva contraseña</label> <input type="password" formControlName="confirmPassword"> <div class="error" \*ngIf="editForm.hasError('mismatch')"> Las contraseñas no coinciden </div> </div> <div class="modal-footer"> <button type="button" class="btn-cancel" (click)="onClose()">Cancelar</button> <button type="submit" class="btn-save" [disabled]="editForm.invalid">Guardar</button> </div> </form> </div> </div> </div> Can't bind to 'formGroup' since it isn't a known property of 'form'.ngtsc(-998002) edit-profile-modal.component.ts(13, 18): Error occurs in the template of component EditProfileModalComponent. import { Component, Inject } from '@angular/core'; import { FormBuilder, FormGroup, Validators, AbstractControl } from '@angular/forms'; import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog'; import { ApiService } from '../../../shared/services/api.service'; interface PasswordRequirements { length: boolean; lowerCase: boolean; upperCase: boolean; number: boolean; specialChar: boolean; } @Component({ selector: 'app-edit-profile-modal', templateUrl: './edit-profile-modal.component.html', styleUrls: ['./edit-profile-modal.component.scss'] }) export class EditProfileModalComponent { editForm: FormGroup; isSubmitting = false; passwordRequirements: PasswordRequirements = { length: false, lowerCase: false, upperCase: false, number: false, specialChar: false }; passwordTouched = false; constructor( private fb: FormBuilder, private apiService: ApiService, private dialogRef: MatDialogRef<EditProfileModalComponent>, @Inject(MAT\_DIALOG\_DATA) public data: { currentName: string } ) { this.editForm = this.fb.group({ name: [this.data.currentName, [Validators.required, Validators.minLength(3)]], newPassword: ['', [Validators.required]] }); this.editForm.get('newPassword')?.valueChanges.subscribe(password => { this.checkPasswordRequirements(password); }); } checkPasswordRequirements(password: string): void { this.passwordRequirements = { length: password?.length >= 8, lowerCase: /[a-z]/.test(password), upperCase: /[A-Z]/.test(password), number: /\d/.test(password), specialChar: /[\W\_]/.test(password) }; this.passwordTouched = !!password && password.length > 0; } get passwordValid(): boolean { return Object.values(this.passwordRequirements).every(req => req); } onSubmit(): void { if (this.editForm.invalid || !this.passwordValid) return; this.isSubmitting = true; const formData = this.editForm.value; this.apiService.put('/users/me', { name: formData.name, password: formData.newPassword }).subscribe({ next: () => { this.dialogRef.close({ success: true, name: formData.name }); }, error: (err) => { this.isSubmitting = false; console.error('Error updating profile:', err); alert('Error updating profile: ' + (err.error?.message || 'Please try again')); } }); } onClose(): void { this.dialogRef.close(); } } Ayudame. Además en el form debe haber solo dos campos. Nuevo nombre y nueva contraseña y deben ser opcionales (al menos completar 1 de ellos para que se envíe)**

**Solución para el Error "Can't bind to 'formGroup'"**

El error ocurre porque necesitas importar el módulo ReactiveFormsModule en el módulo que contiene tu componente. Aquí está cómo solucionarlo:

**Paso 1: Importar ReactiveFormsModule**

En el módulo donde está declarado tu componente (probablemente app.module.ts o un módulo compartido), agrega:

typescript

import { ReactiveFormsModule } from '@angular/forms';

@NgModule({

imports: [

// otros imports

ReactiveFormsModule

],

// ...

})

export class YourModule { }

**Paso 2: Modificar el componente según tus requisitos**

Aquí está el componente actualizado con solo dos campos (nombre y contraseña) y validación opcional:

typescript

import { Component, Inject } from '@angular/core';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog';

import { ApiService } from '../../../shared/services/api.service';

@Component({

selector: 'app-edit-profile-modal',

templateUrl: './edit-profile-modal.component.html',

styleUrls: ['./edit-profile-modal.component.scss']

})

export class EditProfileModalComponent {

editForm: FormGroup;

isSubmitting = false;

constructor(

private fb: FormBuilder,

private apiService: ApiService,

private dialogRef: MatDialogRef<EditProfileModalComponent>,

@Inject(MAT\_DIALOG\_DATA) public data: { currentName: string }

) {

this.editForm = this.fb.group({

name: [this.data.currentName, [Validators.minLength(3)]],

newPassword: ['', [Validators.minLength(6)]]

}, { validator: this.atLeastOneRequired });

}

// Validador personalizado para requerir al menos uno de los campos

atLeastOneRequired(group: FormGroup): { [key: string]: any } | null {

const name = group.get('name')?.value;

const password = group.get('newPassword')?.value;

return name || password ? null : { atLeastOneRequired: true };

}

onSubmit(): void {

if (this.editForm.invalid) return;

this.isSubmitting = true;

const formData = this.editForm.value;

// Solo enviar los campos que tienen valores

const updateData: any = {};

if (formData.name && formData.name !== this.data.currentName) {

updateData.name = formData.name;

}

if (formData.newPassword) {

updateData.password = formData.newPassword;

}

// Si no hay nada que actualizar, cerrar el diálogo

if (Object.keys(updateData).length === 0) {

this.dialogRef.close();

return;

}

this.apiService.put('/users/me', updateData).subscribe({

next: () => {

this.dialogRef.close({

success: true,

name: updateData.name || this.data.currentName

});

},

error: (err) => {

this.isSubmitting = false;

console.error('Error updating profile:', err);

alert('Error updating profile: ' + (err.error?.message || 'Please try again'));

}

});

}

onClose(): void {

this.dialogRef.close();

}

}

**Paso 3: Actualizar la plantilla HTML**

html

<div class="modal-overlay" \*ngIf="isVisible" (click)="onClose()">

<div class="modal-content" (click)="$event.stopPropagation()">

<div class="modal-header">

<h2>Editar Perfil</h2>

<button class="close-btn" (click)="onClose()">&times;</button>

</div>

<div class="modal-body">

<form [formGroup]="editForm" (ngSubmit)="onSubmit()">

<div class="form-group">

<label>Nuevo nombre</label>

<input type="text" formControlName="name" placeholder="Dejar vacío para no cambiar">

<div class="error" \*ngIf="editForm.get('name')?.hasError('minlength')">

Mínimo 3 caracteres

</div>

</div>

<div class="form-group">

<label>Nueva contraseña</label>

<input type="password" formControlName="newPassword" placeholder="Dejar vacío para no cambiar">

<div class="error" \*ngIf="editForm.get('newPassword')?.hasError('minlength')">

Mínimo 6 caracteres

</div>

</div>

<div class="error" \*ngIf="editForm.hasError('atLeastOneRequired') && (editForm.touched || editForm.dirty)">

Debes completar al menos uno de los campos

</div>

<div class="modal-footer">

<button type="button" class="btn-cancel" (click)="onClose()">Cancelar</button>

<button type="submit" class="btn-save" [disabled]="editForm.invalid || isSubmitting">

{{ isSubmitting ? 'Guardando...' : 'Guardar' }}

</button>

</div>

</form>

</div>

</div>

</div>

**Cambios realizados:**

1. **Formulario simplificado**: Solo dos campos (nombre y contraseña), ambos opcionales.
2. **Validador personalizado**: atLeastOneRequired que verifica que al menos uno de los campos tenga valor.
3. **Validación mejorada**:
   * Nombre: mínimo 3 caracteres (solo si se proporciona)
   * Contraseña: mínimo 6 caracteres (solo si se proporciona)
4. **Lógica de envío**: Solo envía los campos que tienen valores.
5. **Mejoras en la UI**:
   * Placeholders que indican que los campos son opcionales
   * Mensaje de error cuando no se completa ningún campo
   * Estado de carga en el botón de guardar

Asegúrate de que ReactiveFormsModule esté importado en tu módulo y estos cambios deberían resolver todos los problemas.

**<!-- Formulario de Registro --> <div class="col-md-6"> <div class="card"> <div class="card-header bg-success text-white"> <h3><i class="fas fa-user-plus me-2"></i>Registro</h3> </div> <div class="card-body"> <form (ngSubmit)="onRegisterSubmit()" #registerForm="ngForm"> <div class="mb-3"> <label class="form-label">Nombre</label> <input type="text" class="form-control" placeholder="Tu nombre" name="name" [(ngModel)]="registerData.name" required minlength="3" #name="ngModel" > <div \*ngIf="name.invalid && (name.dirty || name.touched)" class="text-danger"> <div \*ngIf="name.errors?.['required']">El nombre es requerido</div> <div \*ngIf="name.errors?.['minlength']">Mínimo 3 caracteres</div> </div> </div> <div class="mb-3"> <label class="form-label">Email</label> <input type="email" class="form-control" placeholder="tucorreo@ejemplo.com" name="email" [(ngModel)]="registerData.email" required email #registeremail="ngModel" > <div \*ngIf="registeremail.invalid && (registeremail.dirty || registeremail.touched)" class="text-danger"> <div \*ngIf="registeremail.errors?.['required']">El email es requerido</div> <div \*ngIf="registeremail.errors?.['email']">Email inválido</div> </div> </div> <div class="mb-3"> <label class="form-label">Contraseña</label> <input type="password" class="form-control" placeholder="••••••••" name="password" [(ngModel)]="registerData.password" (input)="checkPasswordRequirements()" #registerpassword="ngModel" > <div class="mt-2" \*ngIf="passwordTouched"> <div class="d-flex align-items-center mb-1"> <span class="me-2" [ngClass]="{'text-success': passwordRequirements.length, 'text-danger': !passwordRequirements.length}"> <i class="fas" [class.fa-check-circle]="passwordRequirements.length" [class.fa-times-circle]="!passwordRequirements.length"></i> </span> <span [ngClass]="{'text-success': passwordRequirements.length, 'text-muted': !passwordRequirements.length}"> Al menos 8 caracteres </span> </div> <div class="d-flex align-items-center mb-1" \*ngIf="passwordTouched"> <span class="me-2" [ngClass]="{'text-success': passwordRequirements.lowerCase, 'text-danger': !passwordRequirements.lowerCase}"> <i class="fas" [class.fa-check-circle]="passwordRequirements.lowerCase" [class.fa-times-circle]="!passwordRequirements.lowerCase"></i> </span> <span [ngClass]="{'text-success': passwordRequirements.lowerCase, 'text-muted': !passwordRequirements.lowerCase}"> Al menos 1 letra minúscula </span> </div> <div class="d-flex align-items-center mb-1" \*ngIf="passwordTouched"> <span class="me-2" [ngClass]="{'text-success': passwordRequirements.upperCase, 'text-danger': !passwordRequirements.upperCase}"> <i class="fas" [class.fa-check-circle]="passwordRequirements.upperCase" [class.fa-times-circle]="!passwordRequirements.upperCase"></i> </span> <span [ngClass]="{'text-success': passwordRequirements.upperCase, 'text-muted': !passwordRequirements.upperCase}"> Al menos 1 letra mayúscula </span> </div> <div class="d-flex align-items-center mb-1" \*ngIf="passwordTouched"> <span class="me-2" [ngClass]="{'text-success': passwordRequirements.number, 'text-danger': !passwordRequirements.number}"> <i class="fas" [class.fa-check-circle]="passwordRequirements.number" [class.fa-times-circle]="!passwordRequirements.number"></i> </span> <span [ngClass]="{'text-success': passwordRequirements.number, 'text-muted': !passwordRequirements.number}"> Al menos 1 número </span> </div> <div class="d-flex align-items-center" \*ngIf="passwordTouched"> <span class="me-2" [ngClass]="{'text-success': passwordRequirements.specialChar, 'text-danger': !passwordRequirements.specialChar}"> <i class="fas" [class.fa-check-circle]="passwordRequirements.specialChar" [class.fa-times-circle]="!passwordRequirements.specialChar"></i> </span> <span [ngClass]="{'text-success': passwordRequirements.specialChar, 'text-muted': !passwordRequirements.specialChar}"> Al menos 1 carácter especial ($!%\*?&) </span> </div> </div> </div> <button type="submit" class="btn btn-success w-100" [disabled]="!registerForm.form.valid">Crear cuenta</button> </form> </div> <div class="modal-overlay" \*ngIf="isVisible" (click)="onClose()"> <div class="modal-content" (click)="$event.stopPropagation()"> <div class="modal-header"> <h2>Editar Perfil</h2> <button class="close-btn" (click)="onClose()">&times;</button> </div> <div class="modal-body"> <form (ngSubmit)="onSubmit()"> <div class="form-group"> <label>Nuevo nombre</label> <input type="text" formControlName="name" placeholder="Dejar vacío para no cambiar"> <div class="error" \*ngIf="editForm.get('name')?.hasError('minlength')"> Mínimo 3 caracteres </div> </div> <div class="form-group"> <label>Nueva contraseña</label> <input type="password" formControlName="newPassword" placeholder="Dejar vacío para no cambiar"> <div class="error" \*ngIf="editForm.get('newPassword')?.hasError('minlength')"> Mínimo 8 caracteres </div> </div> <div class="error" \*ngIf="editForm.hasError('atLeastOneRequired') && (editForm.touched || editForm.dirty)"> Debes completar al menos uno de los campos </div> <div class="modal-footer"> <button type="button" class="btn-cancel" (click)="onClose()">Cancelar</button> <button type="submit" class="btn-save" [disabled]="editForm.invalid || isSubmitting"> {{ isSubmitting ? 'Guardando...' : 'Guardar' }} </button> </div> </form> </div> </div> </div> Actualizame mi edit-profile-modal con los nuevos requisitos y ajusta el componente como consideres. import { Component, Inject } from '@angular/core'; import { FormBuilder, FormGroup, Validators, AbstractControl } from '@angular/forms'; import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog'; import { ApiService } from '../../../shared/services/api.service'; interface PasswordRequirements { length: boolean; lowerCase: boolean; upperCase: boolean; number: boolean; specialChar: boolean; } @Component({ selector: 'app-edit-profile-modal', templateUrl: './edit-profile-modal.component.html', styleUrls: ['./edit-profile-modal.component.scss'] }) export class EditProfileModalComponent { editForm: FormGroup; isSubmitting = false; passwordRequirements: PasswordRequirements = { length: false, lowerCase: false, upperCase: false, number: false, specialChar: false }; passwordTouched = false; constructor( private fb: FormBuilder, private apiService: ApiService, private dialogRef: MatDialogRef<EditProfileModalComponent>, @Inject(MAT\_DIALOG\_DATA) public data: { currentName: string } ) { this.editForm = this.fb.group({ name: [this.data.currentName, [Validators.required, Validators.minLength(3)]], newPassword: ['', [Validators.required]] }); this.editForm.get('newPassword')?.valueChanges.subscribe(password => { this.checkPasswordRequirements(password); }); } checkPasswordRequirements(password: string): void { this.passwordRequirements = { length: password?.length >= 8, lowerCase: /[a-z]/.test(password), upperCase: /[A-Z]/.test(password), number: /\d/.test(password), specialChar: /[\W\_]/.test(password) }; this.passwordTouched = !!password && password.length > 0; } get passwordValid(): boolean { return Object.values(this.passwordRequirements).every(req => req); } onSubmit(): void { if (this.editForm.invalid || !this.passwordValid) return; this.isSubmitting = true; const formData = this.editForm.value; this.apiService.put('/users/me', { name: formData.name, password: formData.newPassword }).subscribe({ next: () => { this.dialogRef.close({ success: true, name: formData.name }); }, error: (err) => { this.isSubmitting = false; console.error('Error updating profile:', err); alert('Error updating profile: ' + (err.error?.message || 'Please try again')); } }); } onClose(): void { this.dialogRef.close(); } }**

**Actualización del Edit Profile Modal**

Basado en los requisitos del formulario de registro, voy a actualizar el componente de edición de perfil para que coincida con los mismos estándares de validación de contraseña y mejorar la experiencia de usuario.

**Componente Actualizado (edit-profile-modal.component.ts)**

typescript

import { Component, Inject } from '@angular/core';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog';

import { ApiService } from '../../../shared/services/api.service';

interface PasswordRequirements {

length: boolean;

lowerCase: boolean;

upperCase: boolean;

number: boolean;

specialChar: boolean;

}

@Component({

selector: 'app-edit-profile-modal',

templateUrl: './edit-profile-modal.component.html',

styleUrls: ['./edit-profile-modal.component.scss']

})

export class EditProfileModalComponent {

editForm: FormGroup;

isSubmitting = false;

passwordRequirements: PasswordRequirements = {

length: false,

lowerCase: false,

upperCase: false,

number: false,

specialChar: false

};

passwordTouched = false;

isVisible = true; // Añadido para controlar la visibilidad del modal

constructor(

private fb: FormBuilder,

private apiService: ApiService,

private dialogRef: MatDialogRef<EditProfileModalComponent>,

@Inject(MAT\_DIALOG\_DATA) public data: { currentName: string }

) {

this.editForm = this.fb.group({

name: [this.data.currentName, [Validators.minLength(3)]],

newPassword: ['', [Validators.minLength(8)]]

}, { validator: this.atLeastOneRequired });

this.editForm.get('newPassword')?.valueChanges.subscribe(password => {

this.checkPasswordRequirements(password);

});

}

// Validador personalizado para requerir al menos uno de los campos

atLeastOneRequired(group: FormGroup): { [key: string]: any } | null {

const name = group.get('name')?.value;

const password = group.get('newPassword')?.value;

return name || password ? null : { atLeastOneRequired: true };

}

checkPasswordRequirements(password: string): void {

this.passwordRequirements = {

length: password?.length >= 8,

lowerCase: /[a-z]/.test(password),

upperCase: /[A-Z]/.test(password),

number: /\d/.test(password),

specialChar: /[\W\_]/.test(password)

};

this.passwordTouched = !!password && password.length > 0;

}

get passwordValid(): boolean {

// Si no hay contraseña, es válido (campo opcional)

if (!this.editForm.get('newPassword')?.value) return true;

// Si hay contraseña, validar todos los requisitos

return Object.values(this.passwordRequirements).every(req => req);

}

onSubmit(): void {

if (this.editForm.invalid || !this.passwordValid) return;

this.isSubmitting = true;

const formData = this.editForm.value;

// Solo enviar los campos que tienen valores

const updateData: any = {};

if (formData.name && formData.name !== this.data.currentName) {

updateData.name = formData.name;

}

if (formData.newPassword) {

updateData.password = formData.newPassword;

}

// Si no hay nada que actualizar, cerrar el diálogo

if (Object.keys(updateData).length === 0) {

this.dialogRef.close();

return;

}

this.apiService.put('/users/me', updateData).subscribe({

next: () => {

this.dialogRef.close({

success: true,

name: updateData.name || this.data.currentName

});

},

error: (err) => {

this.isSubmitting = false;

console.error('Error updating profile:', err);

alert('Error updating profile: ' + (err.error?.message || 'Please try again'));

}

});

}

onClose(): void {

this.dialogRef.close();

}

}

**Plantilla Actualizada (edit-profile-modal.component.html)**

html

<div class="modal-overlay" \*ngIf="isVisible" (click)="onClose()">

<div class="modal-content" (click)="$event.stopPropagation()">

<div class="modal-header">

<h2>Editar Perfil</h2>

<button class="close-btn" (click)="onClose()">&times;</button>

</div>

<div class="modal-body">

<form [formGroup]="editForm" (ngSubmit)="onSubmit()">

<div class="form-group mb-3">

<label class="form-label">Nuevo nombre</label>

<input

type="text"

class="form-control"

formControlName="name"

placeholder="Dejar vacío para no cambiar"

>

<div class="text-danger mt-1" \*ngIf="editForm.get('name')?.hasError('minlength')">

Mínimo 3 caracteres

</div>

</div>

<div class="form-group mb-3">

<label class="form-label">Nueva contraseña</label>

<input

type="password"

class="form-control"

formControlName="newPassword"

placeholder="Dejar vacío para no cambiar"

(input)="passwordTouched = true"

>

<div class="mt-2" \*ngIf="passwordTouched">

<div class="d-flex align-items-center mb-1">

<span class="me-2" [ngClass]="{'text-success': passwordRequirements.length, 'text-danger': !passwordRequirements.length}">

<i class="fas" [class.fa-check-circle]="passwordRequirements.length" [class.fa-times-circle]="!passwordRequirements.length"></i>

</span>

<span [ngClass]="{'text-success': passwordRequirements.length, 'text-muted': !passwordRequirements.length}">

Al menos 8 caracteres

</span>

</div>

<div class="d-flex align-items-center mb-1">

<span class="me-2" [ngClass]="{'text-success': passwordRequirements.lowerCase, 'text-danger': !passwordRequirements.lowerCase}">

<i class="fas" [class.fa-check-circle]="passwordRequirements.lowerCase" [class.fa-times-circle]="!passwordRequirements.lowerCase"></i>

</span>

<span [ngClass]="{'text-success': passwordRequirements.lowerCase, 'text-muted': !passwordRequirements.lowerCase}">

Al menos 1 letra minúscula

</span>

</div>

<div class="d-flex align-items-center mb-1">

<span class="me-2" [ngClass]="{'text-success': passwordRequirements.upperCase, 'text-danger': !passwordRequirements.upperCase}">

<i class="fas" [class.fa-check-circle]="passwordRequirements.upperCase" [class.fa-times-circle]="!passwordRequirements.upperCase"></i>

</span>

<span [ngClass]="{'text-success': passwordRequirements.upperCase, 'text-muted': !passwordRequirements.upperCase}">

Al menos 1 letra mayúscula

</span>

</div>

<div class="d-flex align-items-center mb-1">

<span class="me-2" [ngClass]="{'text-success': passwordRequirements.number, 'text-danger': !passwordRequirements.number}">

<i class="fas" [class.fa-check-circle]="passwordRequirements.number" [class.fa-times-circle]="!passwordRequirements.number"></i>

</span>

<span [ngClass]="{'text-success': passwordRequirements.number, 'text-muted': !passwordRequirements.number}">

Al menos 1 número

</span>

</div>

<div class="d-flex align-items-center">

<span class="me-2" [ngClass]="{'text-success': passwordRequirements.specialChar, 'text-danger': !passwordRequirements.specialChar}">

<i class="fas" [class.fa-check-circle]="passwordRequirements.specialChar" [class.fa-times-circle]="!passwordRequirements.specialChar"></i>

</span>

<span [ngClass]="{'text-success': passwordRequirements.specialChar, 'text-muted': !passwordRequirements.specialChar}">

Al menos 1 carácter especial ($!%\*?&)

</span>

</div>

</div>

</div>

<div class="text-danger mb-3" \*ngIf="editForm.hasError('atLeastOneRequired') && (editForm.touched || editForm.dirty)">

Debes completar al menos uno de los campos

</div>

<div class="modal-footer">

<button type="button" class="btn btn-outline-secondary" (click)="onClose()">Cancelar</button>

<button

type="submit"

class="btn btn-primary"

[disabled]="editForm.invalid || !passwordValid || isSubmitting"

>

<span \*ngIf="!isSubmitting">Guardar</span>

<span \*ngIf="isSubmitting">

<span class="spinner-border spinner-border-sm" role="status" aria-hidden="true"></span>

Guardando...

</span>

</button>

</div>

</form>

</div>

</div>

</div>

**Cambios realizados:**

1. **Validación de contraseña mejorada**:
   * Mismos requisitos que en el formulario de registro (8 caracteres, mayúscula, minúscula, número y carácter especial)
   * Visualización interactiva de los requisitos como en el registro
2. **Campos opcionales**:
   * Validador personalizado atLeastOneRequired que asegura que al menos un campo esté completo
   * Los campos pueden dejarse vacíos (no son requeridos)
3. **Mejoras en la UI**:
   * Estilos consistentes con Bootstrap (clases form-control, btn, etc.)
   * Spinner de carga durante el envío
   * Mejor feedback visual para los requisitos de contraseña
4. **Lógica de envío optimizada**:
   * Solo envía los campos que han cambiado
   * No envía la contraseña si el campo está vacío
   * No envía el nombre si no ha cambiado
5. **Control de visibilidad**:
   * Añadida propiedad isVisible para controlar la visualización del modal

Asegúrate de:

1. Tener importado ReactiveFormsModule en tu módulo
2. Tener los estilos de Font Awesome para los iconos de validación
3. Que los estilos CSS sean consistentes con el resto de tu aplicación

**<div class="modal-overlay" \*ngIf="isVisible" (click)="onClose()"> <div class="modal-content" (click)="$event.stopPropagation()"> <div class="modal-header"> <h2>Editar Perfil</h2> <button class="close-btn" (click)="onClose()">&times;</button> </div> <div class="modal-body"> <form [formGroup]="editForm" (ngSubmit)="onSubmit()"> <div class="form-group mb-3"> <label class="form-label">Nuevo nombre</label> <input type="text" class="form-control" formControlName="name" placeholder="Dejar vacío para no cambiar" > <div class="text-danger mt-1" \*ngIf="editForm.get('name')?.hasError('minlength')"> Mínimo 3 caracteres </div> </div> <div class="form-group mb-3"> <label class="form-label">Nueva contraseña</label> <input type="password" class="form-control" formControlName="newPassword" placeholder="Dejar vacío para no cambiar" (input)="passwordTouched = true" > <div class="mt-2" \*ngIf="passwordTouched"> <div class="d-flex align-items-center mb-1"> <span class="me-2" [ngClass]="{'text-success': passwordRequirements.length, 'text-danger': !passwordRequirements.length}"> <i class="fas" [class.fa-check-circle]="passwordRequirements.length" [class.fa-times-circle]="!passwordRequirements.length"></i> </span> <span [ngClass]="{'text-success': passwordRequirements.length, 'text-muted': !passwordRequirements.length}"> Al menos 8 caracteres </span> </div> <div class="d-flex align-items-center mb-1"> <span class="me-2" [ngClass]="{'text-success': passwordRequirements.lowerCase, 'text-danger': !passwordRequirements.lowerCase}"> <i class="fas" [class.fa-check-circle]="passwordRequirements.lowerCase" [class.fa-times-circle]="!passwordRequirements.lowerCase"></i> </span> <span [ngClass]="{'text-success': passwordRequirements.lowerCase, 'text-muted': !passwordRequirements.lowerCase}"> Al menos 1 letra minúscula </span> </div> <div class="d-flex align-items-center mb-1"> <span class="me-2" [ngClass]="{'text-success': passwordRequirements.upperCase, 'text-danger': !passwordRequirements.upperCase}"> <i class="fas" [class.fa-check-circle]="passwordRequirements.upperCase" [class.fa-times-circle]="!passwordRequirements.upperCase"></i> </span> <span [ngClass]="{'text-success': passwordRequirements.upperCase, 'text-muted': !passwordRequirements.upperCase}"> Al menos 1 letra mayúscula </span> </div> <div class="d-flex align-items-center mb-1"> <span class="me-2" [ngClass]="{'text-success': passwordRequirements.number, 'text-danger': !passwordRequirements.number}"> <i class="fas" [class.fa-check-circle]="passwordRequirements.number" [class.fa-times-circle]="!passwordRequirements.number"></i> </span> <span [ngClass]="{'text-success': passwordRequirements.number, 'text-muted': !passwordRequirements.number}"> Al menos 1 número </span> </div> <div class="d-flex align-items-center"> <span class="me-2" [ngClass]="{'text-success': passwordRequirements.specialChar, 'text-danger': !passwordRequirements.specialChar}"> <i class="fas" [class.fa-check-circle]="passwordRequirements.specialChar" [class.fa-times-circle]="!passwordRequirements.specialChar"></i> </span> <span [ngClass]="{'text-success': passwordRequirements.specialChar, 'text-muted': !passwordRequirements.specialChar}"> Al menos 1 carácter especial ($!%\*?&) </span> </div> </div> </div> <div class="text-danger mb-3" \*ngIf="editForm.hasError('atLeastOneRequired') && (editForm.touched || editForm.dirty)"> Debes completar al menos uno de los campos </div> <div class="modal-footer"> <button type="button" class="btn btn-outline-secondary" (click)="onClose()">Cancelar</button> <button type="submit" class="btn btn-primary" [disabled]="editForm.invalid || !passwordValid || isSubmitting" > <span \*ngIf="!isSubmitting">Guardar</span> <span \*ngIf="isSubmitting"> <span class="spinner-border spinner-border-sm" role="status" aria-hidden="true"></span> Guardando... </span> </button> </div> </form> </div> </div> </div> LOS NGCLASS Y LOS FORMGROUP SE PONEN EN ROJO, EVITALOS, EN MIS OTROS FORMS NO LOS USO Y NO PASA NADA.**

**Solución sin ngClass y formGroup**

Entiendo que prefieres evitar el uso de ngClass y formGroup para mantener consistencia con tus otros formularios. Aquí tienes una versión simplificada usando ngModel y alternativas para los estilos condicionales:

**Componente Actualizado (edit-profile-modal.component.ts)**

typescript

import { Component, Inject } from '@angular/core';

import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog';

import { ApiService } from '../../../shared/services/api.service';

@Component({

selector: 'app-edit-profile-modal',

templateUrl: './edit-profile-modal.component.html',

styleUrls: ['./edit-profile-modal.component.scss']

})

export class EditProfileModalComponent {

name: string = '';

newPassword: string = '';

isSubmitting = false;

isVisible = true;

passwordTouched = false;

// Variables para los requisitos de contraseña

reqLength = false;

reqLowerCase = false;

reqUpperCase = false;

reqNumber = false;

reqSpecialChar = false;

constructor(

private apiService: ApiService,

private dialogRef: MatDialogRef<EditProfileModalComponent>,

@Inject(MAT\_DIALOG\_DATA) public data: { currentName: string }

) {

this.name = data.currentName;

}

checkPasswordRequirements(): void {

const password = this.newPassword;

this.reqLength = password?.length >= 8;

this.reqLowerCase = /[a-z]/.test(password);

this.reqUpperCase = /[A-Z]/.test(password);

this.reqNumber = /\d/.test(password);

this.reqSpecialChar = /[\W\_]/.test(password);

this.passwordTouched = !!password && password.length > 0;

}

get passwordValid(): boolean {

if (!this.newPassword) return true;

return this.reqLength && this.reqLowerCase && this.reqUpperCase &&

this.reqNumber && this.reqSpecialChar;

}

get formValid(): boolean {

return (this.name.length >= 3 || this.newPassword.length >= 8) &&

(this.newPassword ? this.passwordValid : true);

}

onSubmit(): void {

if (!this.formValid) return;

this.isSubmitting = true;

const updateData: any = {};

if (this.name && this.name !== this.data.currentName) {

updateData.name = this.name;

}

if (this.newPassword) {

updateData.password = this.newPassword;

}

if (Object.keys(updateData).length === 0) {

this.dialogRef.close();

return;

}

this.apiService.put('/users/me', updateData).subscribe({

next: () => {

this.dialogRef.close({ success: true, name: updateData.name || this.data.currentName });

},

error: (err) => {

this.isSubmitting = false;

console.error('Error updating profile:', err);

alert('Error updating profile: ' + (err.error?.message || 'Please try again'));

}

});

}

onClose(): void {

this.dialogRef.close();

}

}

**Plantilla Actualizada (edit-profile-modal.component.html)**

html

<div class="modal-overlay" \*ngIf="isVisible" (click)="onClose()">

<div class="modal-content" (click)="$event.stopPropagation()">

<div class="modal-header">

<h2>Editar Perfil</h2>

<button class="close-btn" (click)="onClose()">&times;</button>

</div>

<div class="modal-body">

<form (ngSubmit)="onSubmit()">

<div class="form-group mb-3">

<label class="form-label">Nuevo nombre</label>

<input

type="text"

class="form-control"

[(ngModel)]="name"

name="name"

placeholder="Dejar vacío para no cambiar"

>

<div class="text-danger mt-1" \*ngIf="name.length > 0 && name.length < 3">

Mínimo 3 caracteres

</div>

</div>

<div class="form-group mb-3">

<label class="form-label">Nueva contraseña</label>

<input

type="password"

class="form-control"

[(ngModel)]="newPassword"

name="newPassword"

placeholder="Dejar vacío para no cambiar"

(input)="checkPasswordRequirements()"

>

<div class="mt-2" \*ngIf="passwordTouched">

<div class="d-flex align-items-center mb-1">

<span class="me-2" [class.text-success]="reqLength" [class.text-danger]="!reqLength">

<i class="fas" [class.fa-check-circle]="reqLength" [class.fa-times-circle]="!reqLength"></i>

</span>

<span [class.text-success]="reqLength" [class.text-muted]="!reqLength">

Al menos 8 caracteres

</span>

</div>

<div class="d-flex align-items-center mb-1">

<span class="me-2" [class.text-success]="reqLowerCase" [class.text-danger]="!reqLowerCase">

<i class="fas" [class.fa-check-circle]="reqLowerCase" [class.fa-times-circle]="!reqLowerCase"></i>

</span>

<span [class.text-success]="reqLowerCase" [class.text-muted]="!reqLowerCase">

Al menos 1 letra minúscula

</span>

</div>

<div class="d-flex align-items-center mb-1">

<span class="me-2" [class.text-success]="reqUpperCase" [class.text-danger]="!reqUpperCase">

<i class="fas" [class.fa-check-circle]="reqUpperCase" [class.fa-times-circle]="!reqUpperCase"></i>

</span>

<span [class.text-success]="reqUpperCase" [class.text-muted]="!reqUpperCase">

Al menos 1 letra mayúscula

</span>

</div>

<div class="d-flex align-items-center mb-1">

<span class="me-2" [class.text-success]="reqNumber" [class.text-danger]="!reqNumber">

<i class="fas" [class.fa-check-circle]="reqNumber" [class.fa-times-circle]="!reqNumber"></i>

</span>

<span [class.text-success]="reqNumber" [class.text-muted]="!reqNumber">

Al menos 1 número

</span>

</div>

<div class="d-flex align-items-center">

<span class="me-2" [class.text-success]="reqSpecialChar" [class.text-danger]="!reqSpecialChar">

<i class="fas" [class.fa-check-circle]="reqSpecialChar" [class.fa-times-circle]="!reqSpecialChar"></i>

</span>

<span [class.text-success]="reqSpecialChar" [class.text-muted]="!reqSpecialChar">

Al menos 1 carácter especial ($!%\*?&)

</span>

</div>

</div>

</div>

<div class="text-danger mb-3" \*ngIf="(name.length < 3 || !newPassword) && (name.length > 0 || newPassword.length > 0)">

Debes completar al menos un campo válido

</div>

<div class="modal-footer">

<button type="button" class="btn btn-outline-secondary" (click)="onClose()">Cancelar</button>

<button

type="submit"

class="btn btn-primary"

[disabled]="!formValid || isSubmitting"

>

<span \*ngIf="!isSubmitting">Guardar</span>

<span \*ngIf="isSubmitting">

<span class="spinner-border spinner-border-sm" role="status" aria-hidden="true"></span>

Guardando...

</span>

</button>

</div>

</form>

</div>

</div>

</div>

**Cambios realizados:**

1. **Eliminado FormGroup**:
   * Ahora se usa [(ngModel)] directamente con propiedades del componente
   * La validación se hace con métodos simples en el componente
2. **Simplificación de clases condicionales**:
   * Reemplazado [ngClass] por [class.nombre-clase]="condición"
   * Mantiene la misma funcionalidad pero con sintaxis más simple
3. **Validación simplificada**:
   * La validación "al menos un campo" se hace con una condición en el template
   * Los requisitos de contraseña se controlan con variables booleanas
4. **Misma funcionalidad**:
   * Todos los requisitos de contraseña se mantienen igual
   * La lógica de envío es idéntica
   * La UI se ve igual pero con implementación más simple

**Notas importantes:**

1. Asegúrate de tener importado FormsModule en tu módulo (no ReactiveFormsModule)
2. Los iconos de Font Awesome siguen siendo necesarios para los indicadores de validación
3. La lógica de validación es ahora más imperativa (en el componente) en lugar de reactiva (FormGroup)

Esta versión debería funcionar igual que la anterior pero sin usar las características que marcaban en rojo en tu editor.

**Can't bind to 'ngModel' since it isn't a known property of 'input'.ngt Todos los NG me dan este error cosa que en mi otro componente no pasaba, que podría estar pasando?**

**Solución al error "Can't bind to 'ngModel'"**

Este error ocurre cuando el módulo FormsModule no está importado en el módulo donde se encuentra tu componente. Aunque en otros componentes funcione, cada módulo necesita sus propias importaciones.

**Pasos para solucionarlo:**

1. **Abre el módulo donde está declarado tu componente** (probablemente app.module.ts o un módulo específico)
2. **Agrega la importación necesaria**:

typescript

import { FormsModule } from '@angular/forms'; // Añade esta línea

@NgModule({

imports: [

// otros imports

FormsModule // Añade esto al array de imports

],

// ...

})

export class TuModulo { }

**¿Por qué ocurre esto si en otros componentes funciona?**

Hay varias posibilidades:

1. **El otro componente está en un módulo diferente** que ya tiene importado FormsModule
2. **Estás usando un módulo compartido** y este componente nuevo está en un módulo que no lo tiene importado
3. **Hubo un cambio reciente** en la estructura de módulos de tu aplicación

**Verificación adicional:**

1. Revisa que el componente esté declarado en el mismo módulo donde importas FormsModule
2. Si usas módulos lazy-loaded, cada módulo lazy necesita sus propias importaciones
3. Verifica que no tengas errores de importación circulares

**Versión alternativa sin ngModel:**

Si prefieres evitar ngModel completamente, puedes usar propiedades y eventos normales:

html

<input

type="text"

class="form-control"

[value]="name"

(input)="name = $event.target.value"

placeholder="Dejar vacío para no cambiar"

>

Pero la solución recomendada es importar FormsModule ya que es la forma estándar de Angular para manejar formularios template-driven.

¿Necesitas ayuda para identificar en qué módulo específico debes hacer esta importación? Podrías compartir la estructura de tu proyecto o los módulos que estás usando para darte una guía más precisa.

**Me estaba dando problemas porque no me lo reconocía como NO standalone**

**Solución para Componente No Standalone con ngModel**

El error ocurre porque estás usando un componente **no standalone** (el modo tradicional de Angular) pero falta la importación del FormsModule en el módulo que lo declara.

**Solución definitiva:**

1. **Abre el módulo que declara tu componente** (normalmente app.module.ts o un módulo feature)
2. **Asegúrate de tener estas importaciones**:

typescript

import { NgModule } from '@angular/core';

import { FormsModule } from '@angular/forms'; // <-- Esta es la clave

import { EditProfileModalComponent } from './edit-profile-modal.component';

@NgModule({

declarations: [

EditProfileModalComponent

],

imports: [

FormsModule, // <-- Añade esto

// otros módulos que necesites...

]

})

export class AppModule { }

**Si prefieres convertirlo a Standalone:**

Si quieres modernizar tu componente y hacerlo standalone (recomendado en Angular 14+):

1. **Modifica tu componente**:

typescript

@Component({

standalone: true, // <-- Añade esto

imports: [FormsModule], // <-- Y esto

selector: 'app-edit-profile-modal',

templateUrl: './edit-profile-modal.component.html',

styleUrls: ['./edit-profile-modal.component.scss']

})

export class EditProfileModalComponent {

// ... resto del código

}

1. **Luego elimínalo de los declarations** de tu módulo (si lo convertiste a standalone)

**¿Por qué funciona en otros componentes?**

Probablemente porque:

* Esos otros componentes están en módulos que sí importan FormsModule
* O son componentes standalone que importan FormsModule individualmente
* O usan ReactiveFormsModule en lugar de FormsModule

**Verificación adicional:**

1. Revisa que el componente no esté declarado en dos módulos diferentes
2. Verifica que no tengas errores de importación circular
3. Asegúrate de que el selector del componente coincida exactamente con cómo lo usas en las plantillas

¿Necesitas ayuda para identificar exactamente en qué módulo deberías hacer estas modificaciones? Podrías compartir la estructura de tu proyecto o los módulos relacionados para una solución más específica.

**<div class="modal-overlay" \*ngIf="isVisible" (click)="onClose()"> <div class="modal-content" (click)="$event.stopPropagation()"> <div class="modal-header"> <h2>Editar Perfil</h2> <button class="close-btn" (click)="onClose()">&times;</button> </div> <div class="modal-body"> <form (ngSubmit)="onSubmit()"> <div class="form-group mb-3"> <label class="form-label">Nuevo nombre</label> <input type="text" class="form-control" [(ngModel)]="name" name="name" placeholder="Dejar vacío para no cambiar" > <div class="text-danger mt-1" \*ngIf="name.length > 0 && name.length < 3"> Mínimo 3 caracteres </div> </div> <div class="form-group mb-3"> <label class="form-label">Nueva contraseña</label> <input type="password" class="form-control" [(ngModel)]="newPassword" name="newPassword" placeholder="Dejar vacío para no cambiar" (input)="checkPasswordRequirements()" > <div class="mt-2" \*ngIf="passwordTouched"> <div class="d-flex align-items-center mb-1"> <span class="me-2" [class.text-success]="reqLength" [class.text-danger]="!reqLength"> <i class="fas" [class.fa-check-circle]="reqLength" [class.fa-times-circle]="!reqLength"></i> </span> <span [class.text-success]="reqLength" [class.text-muted]="!reqLength"> Al menos 8 caracteres </span> </div> <div class="d-flex align-items-center mb-1"> <span class="me-2" [class.text-success]="reqLowerCase" [class.text-danger]="!reqLowerCase"> <i class="fas" [class.fa-check-circle]="reqLowerCase" [class.fa-times-circle]="!reqLowerCase"></i> </span> <span [class.text-success]="reqLowerCase" [class.text-muted]="!reqLowerCase"> Al menos 1 letra minúscula </span> </div> <div class="d-flex align-items-center mb-1"> <span class="me-2" [class.text-success]="reqUpperCase" [class.text-danger]="!reqUpperCase"> <i class="fas" [class.fa-check-circle]="reqUpperCase" [class.fa-times-circle]="!reqUpperCase"></i> </span> <span [class.text-success]="reqUpperCase" [class.text-muted]="!reqUpperCase"> Al menos 1 letra mayúscula </span> </div> <div class="d-flex align-items-center mb-1"> <span class="me-2" [class.text-success]="reqNumber" [class.text-danger]="!reqNumber"> <i class="fas" [class.fa-check-circle]="reqNumber" [class.fa-times-circle]="!reqNumber"></i> </span> <span [class.text-success]="reqNumber" [class.text-muted]="!reqNumber"> Al menos 1 número </span> </div> <div class="d-flex align-items-center"> <span class="me-2" [class.text-success]="reqSpecialChar" [class.text-danger]="!reqSpecialChar"> <i class="fas" [class.fa-check-circle]="reqSpecialChar" [class.fa-times-circle]="!reqSpecialChar"></i> </span> <span [class.text-success]="reqSpecialChar" [class.text-muted]="!reqSpecialChar"> Al menos 1 carácter especial ($!%\*?&) </span> </div> </div> </div> <div class="text-danger mb-3" \*ngIf="(name.length < 3 || !newPassword) && (name.length > 0 || newPassword.length > 0)"> Debes completar al menos un campo válido </div> <div class="modal-footer"> <button type="button" class="btn btn-outline-secondary" (click)="onClose()">Cancelar</button> <button type="submit" class="btn btn-primary" [disabled]="!formValid || isSubmitting" > <span \*ngIf="!isSubmitting">Guardar</span> <span \*ngIf="isSubmitting"> <span class="spinner-border spinner-border-sm" role="status" aria-hidden="true"></span> Guardando... </span> </button> </div> </form> </div> </div> </div> import { Component, Inject } from '@angular/core'; import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog'; import { ApiService } from '../../../shared/services/api.service'; @Component({ selector: 'app-edit-profile-modal', standalone: false, templateUrl: './ed-profile-modal.component.html', styleUrls: ['./ed-profile-modal.component.scss'] }) export class EdProfileModalComponent { name: string = ''; newPassword: string = ''; isSubmitting = false; isVisible = true; passwordTouched = false; // Variables para los requisitos de contraseña reqLength = false; reqLowerCase = false; reqUpperCase = false; reqNumber = false; reqSpecialChar = false; constructor( private apiService: ApiService, private dialogRef: MatDialogRef<EdProfileModalComponent>, @Inject(MAT\_DIALOG\_DATA) public data: { currentName: string } ) { this.name = data.currentName; } checkPasswordRequirements(): void { const password = this.newPassword; this.reqLength = password?.length >= 8; this.reqLowerCase = /[a-z]/.test(password); this.reqUpperCase = /[A-Z]/.test(password); this.reqNumber = /\d/.test(password); this.reqSpecialChar = /[\W\_]/.test(password); this.passwordTouched = !!password && password.length > 0; } get passwordValid(): boolean { if (!this.newPassword) return true; return this.reqLength && this.reqLowerCase && this.reqUpperCase && this.reqNumber && this.reqSpecialChar; } get formValid(): boolean { return (this.name.length >= 3 || this.newPassword.length >= 8) && (this.newPassword ? this.passwordValid : true); } onSubmit(): void { if (!this.formValid) return; this.isSubmitting = true; const updateData: any = {}; if (this.name && this.name !== this.data.currentName) { updateData.name = this.name; } if (this.newPassword) { updateData.password = this.newPassword; } if (Object.keys(updateData).length === 0) { this.dialogRef.close(); return; } this.apiService.put('/users/me', updateData).subscribe({ next: () => { this.dialogRef.close({ success: true, name: updateData.name || this.data.currentName }); }, error: (err) => { this.isSubmitting = false; console.error('Error updating profile:', err); alert('Error updating profile: ' + (err.error?.message || 'Please try again')); } }); } onClose(): void { this.dialogRef.close(); } } <div class="container py-5"> <div class="row justify-content-center"> <div class="col-lg-8"> <div class="card border-0 shadow-sm"> <div class="card-body p-4"> <!-- Header del perfil --> <div class="d-flex flex-column align-items-center text-center mb-4"> <div class="rounded-circle bg-primary bg-opacity-10 d-flex align-items-center justify-content-center mb-3" style="width: 120px; height: 120px;"> <i class="bi bi-person-fill text-primary" style="font-size: 3rem;"></i> </div> <h2 class="mb-1">{{ user.name }}</h2> <p class="text-muted mb-3">{{ user.email }}</p> <span class="badge bg-primary rounded-pill px-3">Usuario #{{ user.id }}</span> </div> <!-- Sección de información --> <div class="row g-3 mb-4"> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-person-lines-fill me-2 text-primary"></i> Información Básica </h3> <ul class="list-unstyled"> <li class="mb-2"> <strong>Número de Usuario:</strong> {{ user.id }} </li> <li class="mb-2"> <strong>Nombre:</strong> {{ user.name }} </li> <li class="mb-2"> <strong>Email:</strong> {{ user.email }} </li> </ul> </div> </div> </div> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-bar-chart-line-fill me-2 text-primary"></i> Tu Productividad </h3> <div class="d-flex flex-wrap gap-3"> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksCompleted }}</div> <small class="text-muted">Tareas completadas</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksPending }}</div> <small class="text-muted">Tareas pendientes</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.categories }}</div> <small class="text-muted">Categorías</small> </div> </div> </div> </div> </div> </div> <!-- Acciones --> <div class="d-flex justify-content-center gap-3"> <button class="btn btn-primary px-4"> <i class="bi bi-pencil-square me-2"></i> Editar Perfil </button> <button class="btn btn-outline-danger px-4" (click)="logout()"> <i class="bi bi-box-arrow-right me-2"></i> Cerrar Sesión </button> </div> </div> </div> </div> </div> </div> import { Component } from '@angular/core'; import { AuthService } from '../../shared/services/auth.service'; import { jwtDecode } from 'jwt-decode'; import { ApiService } from '../../shared/services/api.service'; @Component({ selector: 'app-profile', standalone: false, templateUrl: './profile.component.html', styleUrls: ['./profile.component.scss'] }) export class ProfileComponent { user = { id: 3, name: 'Alex García', email: 'alex@ejemplo.com', stats: { tasksCompleted: 0, tasksPending: 0, categories: 0 }, }; isLoading = true; errorMessage: string | null = null; constructor( private authService: AuthService, private apiService: ApiService ) {} ngOnInit(): void { this.loadUserData(); //this.loadStatistics(); } loadUserData(): void { this.isLoading = true; this.errorMessage = null; // Suponiendo que tienes un método para obtener los datos del usuario logueado const cut = this.authService.getDecodedToken(); //if (cut) { // this.user.id = cut.sub; // this.user.name = cut.name; // this.user.email = cut.email; // } // <- Actualmente desde el token podría sacarse todo //this.apiService.get(`users/id/${this.user.id}`) // <- Este endpoint sirve pero mejor usar el seguro this.apiService.get(`auth/getProfile`).subscribe({ // <- Este endpoint actualmente saca la info del //propio JWT, lo que no es lo suyo para obtener más datos. Se puede modificar de cara a futuro next: (userDetails) => { this.user = { ...this.user, id: userDetails.sub, name: userDetails.name, email: userDetails.email, } this.isLoading = false; }, error: (err) => { console.error('Error loading profile:', err); this.errorMessage = 'Error al cargar el perfil'; this.isLoading = false; } }) } logout(): void { this.authService.logout(); } } El botón editar perfil sigue sin abrir el cuadro emergente con el formulario.**

**Solución para Abrir el Modal de Edición de Perfil**

El problema es que no has implementado la lógica para abrir el modal cuando se hace clic en el botón "Editar Perfil". Aquí te muestro cómo solucionarlo:

**1. Primero, modifica el componente Profile (profile.component.ts):**

typescript

import { Component } from '@angular/core';

import { AuthService } from '../../shared/services/auth.service';

import { ApiService } from '../../shared/services/api.service';

import { MatDialog } from '@angular/material/dialog'; // Importa MatDialog

import { EdProfileModalComponent } from '../ed-profile-modal/ed-profile-modal.component'; // Asegúrate de que la ruta sea correcta

@Component({

selector: 'app-profile',

standalone: false,

templateUrl: './profile.component.html',

styleUrls: ['./profile.component.scss']

})

export class ProfileComponent {

user = {

id: 3,

name: 'Alex García',

email: 'alex@ejemplo.com',

stats: {

tasksCompleted: 0,

tasksPending: 0,

categories: 0

},

};

isLoading = true;

errorMessage: string | null = null;

constructor(

private authService: AuthService,

private apiService: ApiService,

private dialog: MatDialog // Inyecta MatDialog

) {}

ngOnInit(): void {

this.loadUserData();

}

loadUserData(): void {

this.isLoading = true;

this.errorMessage = null;

this.apiService.get(`auth/getProfile`).subscribe({

next: (userDetails) => {

this.user = {

...this.user,

id: userDetails.sub,

name: userDetails.name,

email: userDetails.email,

}

this.isLoading = false;

},

error: (err) => {

console.error('Error loading profile:', err);

this.errorMessage = 'Error al cargar el perfil';

this.isLoading = false;

}

})

}

openEditModal(): void {

const dialogRef = this.dialog.open(EdProfileModalComponent, {

width: '500px',

data: { currentName: this.user.name }

});

dialogRef.afterClosed().subscribe(result => {

if (result?.success) {

this.user.name = result.name; // Actualiza el nombre si se editó

// Aquí podrías agregar una recarga de datos si es necesario

}

});

}

logout(): void {

this.authService.logout();

}

}

**2. Modifica el botón en profile.component.html:**

html

<!-- Cambia esto: -->

<button class="btn btn-primary px-4">

<i class="bi bi-pencil-square me-2"></i>

Editar Perfil

</button>

<!-- Por esto: -->

<button class="btn btn-primary px-4" (click)="openEditModal()">

<i class="bi bi-pencil-square me-2"></i>

Editar Perfil

</button>

**3. Asegúrate de que el módulo tenga las importaciones necesarias:**

En el módulo donde declaras estos componentes (probablemente app.module.ts), asegúrate de tener:

typescript

import { MatDialogModule } from '@angular/material/dialog';

import { FormsModule } from '@angular/forms';

@NgModule({

imports: [

// ... otros imports

MatDialogModule,

FormsModule

],

// ...

})

export class AppModule { }

**4. Verifica que el modal esté declarado correctamente:**

El componente EdProfileModalComponent debe estar declarado en el mismo módulo donde está declarado ProfileComponent, o en un módulo que sea importado por este.

**Pasos adicionales si persiste el problema:**

1. Verifica que tengas instalado @angular/material:

bash

 npm install @angular/material @angular/cdk

 Asegúrate de que el MatDialogModule esté importado en el módulo correcto.

 Verifica que no haya errores en la consola del navegador que puedan estar bloqueando la apertura del modal.

 Si usas un tema de Angular Material, asegúrate de tenerlo configurado en tu styles.css:

css

1. @import "@angular/material/prebuilt-themes/indigo-pink.css";

Con estos cambios, el modal debería abrirse correctamente cuando hagas clic en el botón "Editar Perfil".

**X [ERROR] NG8002: Can't bind to 'ngModel' since it isn't a known property of 'input'. [plugin angular-compiler] src/app/components/ui/ed-profile-modal/ed-profile-modal.component.html:15:12: 15 │ [(ngModel)]="name" ╵ ~~~~~~~~~~~~~~~~~~ Error occurs in the template of component EdProfileModalComponent. src/app/components/ui/ed-profile-modal/ed-profile-modal.component.ts:8:15: 8 │ templateUrl: './ed-profile-modal.component.html', ╵ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ X [ERROR] NG8002: Can't bind to 'ngModel' since it isn't a known property of 'input'. [plugin angular-compiler] src/app/components/ui/ed-profile-modal/ed-profile-modal.component.html:29:12: 29 │ [(ngModel)]="newPassword" ╵ ~~~~~~~~~~~~~~~~~~~~~~~~~ Error occurs in the template of component EdProfileModalComponent. src/app/components/ui/ed-profile-modal/ed-profile-modal.component.ts:8:15: 8 │ templateUrl: './ed-profile-modal.component.html', ╵ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Watch mode enabled. Watching for file changes... <div class="modal-overlay" \*ngIf="isVisible" (click)="onClose()"> <div class="modal-content" (click)="$event.stopPropagation()"> <div class="modal-header"> <h2>Editar Perfil</h2> <button class="close-btn" (click)="onClose()">&times;</button> </div> <div class="modal-body"> <form (ngSubmit)="onSubmit()"> <div class="form-group mb-3"> <label class="form-label">Nuevo nombre</label> <input type="text" class="form-control" [(ngModel)]="name" name="name" placeholder="Dejar vacío para no cambiar" > <div class="text-danger mt-1" \*ngIf="name.length > 0 && name.length < 3"> Mínimo 3 caracteres </div> </div> <div class="form-group mb-3"> <label class="form-label">Nueva contraseña</label> <input type="password" class="form-control" [(ngModel)]="newPassword" name="newPassword" placeholder="Dejar vacío para no cambiar" (input)="checkPasswordRequirements()" > <div class="mt-2" \*ngIf="passwordTouched"> <div class="d-flex align-items-center mb-1"> <span class="me-2" [class.text-success]="reqLength" [class.text-danger]="!reqLength"> <i class="fas" [class.fa-check-circle]="reqLength" [class.fa-times-circle]="!reqLength"></i> </span> <span [class.text-success]="reqLength" [class.text-muted]="!reqLength"> Al menos 8 caracteres </span> </div> <div class="d-flex align-items-center mb-1"> <span class="me-2" [class.text-success]="reqLowerCase" [class.text-danger]="!reqLowerCase"> <i class="fas" [class.fa-check-circle]="reqLowerCase" [class.fa-times-circle]="!reqLowerCase"></i> </span> <span [class.text-success]="reqLowerCase" [class.text-muted]="!reqLowerCase"> Al menos 1 letra minúscula </span> </div> <div class="d-flex align-items-center mb-1"> <span class="me-2" [class.text-success]="reqUpperCase" [class.text-danger]="!reqUpperCase"> <i class="fas" [class.fa-check-circle]="reqUpperCase" [class.fa-times-circle]="!reqUpperCase"></i> </span> <span [class.text-success]="reqUpperCase" [class.text-muted]="!reqUpperCase"> Al menos 1 letra mayúscula </span> </div> <div class="d-flex align-items-center mb-1"> <span class="me-2" [class.text-success]="reqNumber" [class.text-danger]="!reqNumber"> <i class="fas" [class.fa-check-circle]="reqNumber" [class.fa-times-circle]="!reqNumber"></i> </span> <span [class.text-success]="reqNumber" [class.text-muted]="!reqNumber"> Al menos 1 número </span> </div> <div class="d-flex align-items-center"> <span class="me-2" [class.text-success]="reqSpecialChar" [class.text-danger]="!reqSpecialChar"> <i class="fas" [class.fa-check-circle]="reqSpecialChar" [class.fa-times-circle]="!reqSpecialChar"></i> </span> <span [class.text-success]="reqSpecialChar" [class.text-muted]="!reqSpecialChar"> Al menos 1 carácter especial ($!%\*?&) </span> </div> </div> </div> <div class="text-danger mb-3" \*ngIf="(name.length < 3 || !newPassword) && (name.length > 0 || newPassword.length > 0)"> Debes completar al menos un campo válido </div> <div class="modal-footer"> <button type="button" class="btn btn-outline-secondary" (click)="onClose()">Cancelar</button> <button type="submit" class="btn btn-primary" [disabled]="!formValid || isSubmitting" > <span \*ngIf="!isSubmitting">Guardar</span> <span \*ngIf="isSubmitting"> <span class="spinner-border spinner-border-sm" role="status" aria-hidden="true"></span> Guardando... </span> </button> </div> </form> </div> </div> </div> import { Component, Inject } from '@angular/core'; import { MAT\_DIALOG\_DATA, MatDialogRef } from '@angular/material/dialog'; import { ApiService } from '../../../shared/services/api.service'; @Component({ selector: 'app-edit-profile-modal', standalone: false, templateUrl: './ed-profile-modal.component.html', styleUrls: ['./ed-profile-modal.component.scss'] }) export class EdProfileModalComponent { name: string = ''; newPassword: string = ''; isSubmitting = false; isVisible = true; passwordTouched = false; // Variables para los requisitos de contraseña reqLength = false; reqLowerCase = false; reqUpperCase = false; reqNumber = false; reqSpecialChar = false; constructor( private apiService: ApiService, private dialogRef: MatDialogRef<EdProfileModalComponent>, @Inject(MAT\_DIALOG\_DATA) public data: { currentName: string } ) { this.name = data.currentName; } checkPasswordRequirements(): void { const password = this.newPassword; this.reqLength = password?.length >= 8; this.reqLowerCase = /[a-z]/.test(password); this.reqUpperCase = /[A-Z]/.test(password); this.reqNumber = /\d/.test(password); this.reqSpecialChar = /[\W\_]/.test(password); this.passwordTouched = !!password && password.length > 0; } get passwordValid(): boolean { if (!this.newPassword) return true; return this.reqLength && this.reqLowerCase && this.reqUpperCase && this.reqNumber && this.reqSpecialChar; } get formValid(): boolean { return (this.name.length >= 3 || this.newPassword.length >= 8) && (this.newPassword ? this.passwordValid : true); } onSubmit(): void { if (!this.formValid) return; this.isSubmitting = true; const updateData: any = {}; if (this.name && this.name !== this.data.currentName) { updateData.name = this.name; } if (this.newPassword) { updateData.password = this.newPassword; } if (Object.keys(updateData).length === 0) { this.dialogRef.close(); return; } this.apiService.put('/users/me', updateData).subscribe({ next: () => { this.dialogRef.close({ success: true, name: updateData.name || this.data.currentName }); }, error: (err) => { this.isSubmitting = false; console.error('Error updating profile:', err); alert('Error updating profile: ' + (err.error?.message || 'Please try again')); } }); } onClose(): void { this.dialogRef.close(); } } import { NgModule } from '@angular/core'; import { CommonModule } from '@angular/common'; import { EdProfileModalComponent } from './ed-profile-modal.component'; import { FormsModule } from '@angular/forms'; @NgModule({ declarations: [EdProfileModalComponent], imports: [ CommonModule, FormsModule ] }) export class EdProfileModalModule { }**

**Solución Completa para el Error NG8002 con ngModel**

El error ocurre porque aunque has creado un módulo para el componente EdProfileModalComponent, necesitas asegurarte de que:

1. El módulo esté correctamente importado donde usas el componente
2. El componente esté correctamente configurado para usar FormsModule

**Solución paso a paso:**

**1. Actualiza el módulo del componente modal (ed-profile-modal.module.ts):**

typescript

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { EdProfileModalComponent } from './ed-profile-modal.component';

import { FormsModule } from '@angular/forms';

import { MatDialogModule } from '@angular/material/dialog';

@NgModule({

declarations: [EdProfileModalComponent],

imports: [

CommonModule,

FormsModule, // Necesario para ngModel

MatDialogModule // Necesario para el diálogo

],

exports: [EdProfileModalComponent] // Añade esta línea

})

export class EdProfileModalModule { }

**2. Actualiza el módulo principal o el módulo que usa el modal:**

En el módulo donde vas a usar el diálogo (normalmente app.module.ts o el módulo de tu feature):

typescript

import { EdProfileModalModule } from './components/ui/ed-profile-modal/ed-profile-modal.module';

@NgModule({

imports: [

// ... otros imports

EdProfileModalModule

],

// ...

})

export class AppModule { }

**3. Alternativa si prefieres no usar un módulo separado:**

Si no quieres usar un módulo separado, puedes modificar directamente el módulo principal:

typescript

import { NgModule } from '@angular/core';

import { FormsModule } from '@angular/forms';

import { MatDialogModule } from '@angular/material/dialog';

import { EdProfileModalComponent } from './components/ui/ed-profile-modal/ed-profile-modal.component';

@NgModule({

declarations: [

// ... otros componentes

EdProfileModalComponent

],

imports: [

// ... otros módulos

FormsModule,

MatDialogModule

],

// ...

})

export class AppModule { }

**4. Verifica en tu componente de perfil (profile.component.ts):**

Asegúrate de que el módulo donde está declarado ProfileComponent también tenga FormsModule:

typescript

import { FormsModule } from '@angular/forms';

@NgModule({

imports: [

FormsModule,

// ... otros imports

],

// ...

})

export class ProfileModule { }

**5. Solución alternativa sin ngModel:**

Si prefieres evitar ngModel completamente, puedes modificar el modal para usar enlaces de propiedades y eventos:

html

<input

type="text"

class="form-control"

[value]="name"

(input)="name = $event.target.value"

name="name"

placeholder="Dejar vacío para no cambiar"

>

html

<input

type="password"

class="form-control"

[value]="newPassword"

(input)="newPassword = $event.target.value; checkPasswordRequirements()"

name="newPassword"

placeholder="Dejar vacío para no cambiar"

>

**Recomendación final:**

La solución más limpia es:

1. Mantener el EdProfileModalModule con las importaciones necesarias
2. Asegurarte de importar este módulo donde lo uses
3. Verificar que no haya conflictos con otros módulos

Si sigues teniendo problemas, verifica:

* Que no hay errores de importación circular
* Que el componente no esté declarado en más de un módulo
* Que la ruta del módulo en la importación sea correcta
* Que no haya errores en la consola del navegador que puedan estar bloqueando la compilación

**import { Component } from '@angular/core'; import { AuthService } from '../../shared/services/auth.service'; import { jwtDecode } from 'jwt-decode'; import { ApiService } from '../../shared/services/api.service'; import { MatDialog } from '@angular/material/dialog'; import { EdProfileModalComponent } from '../../components/ui/ed-profile-modal/ed-profile-modal.component'; @Component({ selector: 'app-profile', standalone: false, templateUrl: './profile.component.html', styleUrls: ['./profile.component.scss'] }) export class ProfileComponent { user = { id: 3, name: 'Alex García', email: 'alex@ejemplo.com', stats: { tasksCompleted: 0, tasksPending: 0, categories: 0 }, }; isLoading = true; errorMessage: string | null = null; constructor( private authService: AuthService, private apiService: ApiService, private dialog: MatDialog ) {} ngOnInit(): void { this.loadUserData(); //this.loadStatistics(); } loadUserData(): void { this.isLoading = true; this.errorMessage = null; // Suponiendo que tienes un método para obtener los datos del usuario logueado const cut = this.authService.getDecodedToken(); //if (cut) { // this.user.id = cut.sub; // this.user.name = cut.name; // this.user.email = cut.email; // } // <- Actualmente desde el token podría sacarse todo //this.apiService.get(`users/id/${this.user.id}`) // <- Este endpoint sirve pero mejor usar el seguro this.apiService.get(`auth/getProfile`).subscribe({ // <- Este endpoint actualmente saca la info del //propio JWT, lo que no es lo suyo para obtener más datos. Se puede modificar de cara a futuro next: (userDetails) => { this.user = { ...this.user, id: userDetails.sub, name: userDetails.name, email: userDetails.email, } this.isLoading = false; }, error: (err) => { console.error('Error loading profile:', err); this.errorMessage = 'Error al cargar el perfil'; this.isLoading = false; } }) } openEditModal(): void { const dialogRef = this.dialog.open(EdProfileModalComponent, { width: '500px', data: { currentName: this.user.name } }); dialogRef.afterClosed().subscribe(result => { if (result?.success) { this.user.name = result.name; // Actualiza el nombre si se editó // Aquí podrías agregar una recarga de datos si es necesario } }); } logout(): void { this.authService.logout(); } } <div class="container py-5"> <div class="row justify-content-center"> <div class="col-lg-8"> <div class="card border-0 shadow-sm"> <div class="card-body p-4"> <!-- Header del perfil --> <div class="d-flex flex-column align-items-center text-center mb-4"> <div class="rounded-circle bg-primary bg-opacity-10 d-flex align-items-center justify-content-center mb-3" style="width: 120px; height: 120px;"> <i class="bi bi-person-fill text-primary" style="font-size: 3rem;"></i> </div> <h2 class="mb-1">{{ user.name }}</h2> <p class="text-muted mb-3">{{ user.email }}</p> <span class="badge bg-primary rounded-pill px-3">Usuario #{{ user.id }}</span> </div> <!-- Sección de información --> <div class="row g-3 mb-4"> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-person-lines-fill me-2 text-primary"></i> Información Básica </h3> <ul class="list-unstyled"> <li class="mb-2"> <strong>Número de Usuario:</strong> {{ user.id }} </li> <li class="mb-2"> <strong>Nombre:</strong> {{ user.name }} </li> <li class="mb-2"> <strong>Email:</strong> {{ user.email }} </li> </ul> </div> </div> </div> <div class="col-md-6"> <div class="card h-100 border-0 bg-light"> <div class="card-body"> <h3 class="h5 mb-3"> <i class="bi bi-bar-chart-line-fill me-2 text-primary"></i> Tu Productividad </h3> <div class="d-flex flex-wrap gap-3"> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksCompleted }}</div> <small class="text-muted">Tareas completadas</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.tasksPending }}</div> <small class="text-muted">Tareas pendientes</small> </div> <div class="text-center p-3 bg-white rounded-3 shadow-sm flex-grow-1"> <div class="h4 text-primary mb-0">{{ user.stats.categories }}</div> <small class="text-muted">Categorías</small> </div> </div> </div> </div> </div> </div> <!-- Acciones --> <div class="d-flex justify-content-center gap-3"> <button class="btn btn-primary px-4"> <i class="bi bi-pencil-square me-2"></i> Editar Perfil </button> <button class="btn btn-outline-danger px-4" (click)="logout()"> <i class="bi bi-box-arrow-right me-2"></i> Cerrar Sesión </button> </div> </div> </div> </div> </div> </div> Nada ocurre cuando le doy al botón**

**Solución para el Botón de Editar Perfil que no Funciona**

El problema principal es que aunque tienes el método openEditModal() en tu componente, el botón de "Editar Perfil" no está conectado a él. Aquí está la solución completa:

**1. Corrige el botón en profile.component.html**

Cambia el botón de "Editar Perfil" para que llame al método:

html

<!-- Cambia esto: -->

<button class="btn btn-primary px-4">

<i class="bi bi-pencil-square me-2"></i>

Editar Perfil

</button>

<!-- Por esto: -->

<button class="btn btn-primary px-4" (click)="openEditModal()">

<i class="bi bi-pencil-square me-2"></i>

Editar Perfil

</button>

**2. Verifica las importaciones necesarias**

Asegúrate de que tu módulo principal (normalmente app.module.ts) tenga estas importaciones:

typescript

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { FormsModule } from '@angular/forms';

import { MatDialogModule } from '@angular/material/dialog';

@NgModule({

declarations: [

// ... tus componentes

],

imports: [

BrowserModule,

FormsModule, // Necesario para ngModel

MatDialogModule, // Necesario para los diálogos

// ... otros módulos

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

**3. Verifica que el componente modal esté declarado correctamente**

El EdProfileModalComponent debe estar declarado en el mismo módulo que ProfileComponent o en un módulo que sea importado por este.

**4. Solución completa para profile.component.ts**

typescript

import { Component } from '@angular/core';

import { AuthService } from '../../shared/services/auth.service';

import { ApiService } from '../../shared/services/api.service';

import { MatDialog } from '@angular/material/dialog';

import { EdProfileModalComponent } from '../../components/ui/ed-profile-modal/ed-profile-modal.component';

@Component({

selector: 'app-profile',

templateUrl: './profile.component.html',

styleUrls: ['./profile.component.scss']

})

export class ProfileComponent {

user = {

id: 3,

name: 'Alex García',

email: 'alex@ejemplo.com',

stats: {

tasksCompleted: 0,

tasksPending: 0,

categories: 0

},

};

isLoading = true;

errorMessage: string | null = null;

constructor(

private authService: AuthService,

private apiService: ApiService,

private dialog: MatDialog

) {}

ngOnInit(): void {

this.loadUserData();

}

loadUserData(): void {

this.isLoading = true;

this.errorMessage = null;

this.apiService.get(`auth/getProfile`).subscribe({

next: (userDetails) => {

this.user = {

...this.user,

id: userDetails.sub,

name: userDetails.name,

email: userDetails.email,

};

this.isLoading = false;

},

error: (err) => {

console.error('Error loading profile:', err);

this.errorMessage = 'Error al cargar el perfil';

this.isLoading = false;

}

});

}

openEditModal(): void {

console.log('Intentando abrir modal...'); // Para depuración

const dialogRef = this.dialog.open(EdProfileModalComponent, {

width: '500px',

data: { currentName: this.user.name }

});

dialogRef.afterClosed().subscribe(result => {

console.log('El modal se cerró con resultado:', result); // Para depuración

if (result?.success) {

this.user.name = result.name;

// Opcional: Recargar datos del usuario

this.loadUserData();

}

});

}

logout(): void {

this.authService.logout();

}

}

**5. Verificación adicional**

Si después de estos cambios sigue sin funcionar:

1. **Verifica la consola del navegador** (F12 > Console) por errores
2. **Asegúrate de que Material Dialog esté instalado**:

bash

 npm install @angular/material @angular/cdk

 **Verifica que tengas un tema de Angular Material** en styles.css:

css

1. @import "@angular/material/prebuilt-themes/indigo-pink.css";
2. **Prueba añadiendo un console.log** en el método openEditModal() para ver si se está llamando

**6. Si usas un módulo específico para el perfil**

Si ProfileComponent está en un módulo de características, asegúrate de importar los módulos necesarios allí:

typescript

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { ProfileComponent } from './profile.component';

import { FormsModule } from '@angular/forms';

import { MatDialogModule } from '@angular/material/dialog';

@NgModule({

declarations: [ProfileComponent],

imports: [

CommonModule,

FormsModule,

MatDialogModule

]

})

export class ProfileModule { }

Con estos cambios, el modal debería abrirse correctamente al hacer clic en el botón "Editar Perfil".

**// services/api.service.ts import { Injectable } from '@angular/core'; import { HttpClient } from '@angular/common/http'; import { Observable } from 'rxjs'; @Injectable({ providedIn: 'root' }) export class ApiService { private apiUrl = 'http://localhost:3000'; constructor(private http: HttpClient) { } // Métodos genéricos para diferentes tipos de solicitudes get(endpoint: string): Observable<any> { return this.http.get(`${this.apiUrl}/${endpoint}`); } post(endpoint: string, data: any): Observable<any> { return this.http.post(`${this.apiUrl}/${endpoint}`, data); } put(endpoint: string, data: any): Observable<any> { return this.http.put(`${this.apiUrl}/${endpoint}`, data); } delete(endpoint: string): Observable<any> { return this.http.delete(`${this.apiUrl}/${endpoint}`); } añademe el patch**

**Implementación del Método PATCH en ApiService**

Aquí está el servicio ApiService actualizado con el método patch():

typescript

// services/api.service.ts

import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class ApiService {

private apiUrl = 'http://localhost:3000';

constructor(private http: HttpClient) { }

// Métodos genéricos para diferentes tipos de solicitudes

get(endpoint: string): Observable<any> {

return this.http.get(`${this.apiUrl}/${endpoint}`);

}

post(endpoint: string, data: any): Observable<any> {

return this.http.post(`${this.apiUrl}/${endpoint}`, data);

}

put(endpoint: string, data: any): Observable<any> {

return this.http.put(`${this.apiUrl}/${endpoint}`, data);

}

patch(endpoint: string, data: any): Observable<any> {

return this.http.patch(`${this.apiUrl}/${endpoint}`, data);

}

delete(endpoint: string): Observable<any> {

return this.http.delete(`${this.apiUrl}/${endpoint}`);

}

}

**Características del método PATCH:**

1. **Actualización parcial**: A diferencia de PUT que reemplaza todo el recurso, PATCH permite enviar solo los campos que quieres actualizar.
2. **Uso típico**:

typescript

// Ejemplo de uso para actualizar solo el nombre de usuario

this.apiService.patch('users/123', { name: 'Nuevo nombre' })

.subscribe(response => {

console.log('Actualización exitosa', response);

});

1. **Ventajas**:
   * Más eficiente que PUT cuando solo necesitas actualizar algunos campos
   * Menos datos enviados en la solicitud
   * Compatible con operaciones de actualización parcial en el servidor

**Implementación completa con tipos genéricos (opcional):**

Si quieres hacer el servicio más type-safe, puedes usar tipos genéricos:

typescript

patch<T>(endpoint: string, data: any): Observable<T> {

return this.http.patch<T>(`${this.apiUrl}/${endpoint}`, data);

}

Y luego usarlo así:

typescript

interface User {

id: number;

name: string;

email: string;

}

this.apiService.patch<User>('users/123', { name: 'Nuevo nombre' })

.subscribe(user => {

console.log('Usuario actualizado:', user);

});