Buenas Prácticas para el Desarrollo en Angular

📋 Índice

🏗️ Arquitectura y Estructura

Estructura de Carpetas Recomendada

```
src/app/
├── core/                          # Módulo Core (Singleton)
│   ├── guards/                    # Guards de autenticación
│   ├── interceptors/              # Interceptores HTTP
│   ├── services/                  # Servicios core
│   ├── models/                    # Modelos globales
│   ├── constants/                 # Constantes globales
│   ├── core.module.ts             # Módulo core
│   └── index.ts                   # Exportaciones
│
├── shared/                        # Módulo Compartido
│   ├── components/                # Componentes reutilizables
│   ├── directives/                # Directivas personalizadas
│   ├── pipes/                     # Pipes personalizados
│   ├── models/                    # Modelos compartidos
│   ├── shared.module.ts           # Módulo compartido
│   └── index.ts                   # Exportaciones
│
├── features/                      # Módulos de Funcionalidades
│   ├── feature1/                  # Feature Module 1
│   │   ├── pages/                 # Páginas del feature
│   │   ├── shared/                # Recursos compartidos del feature
│   │   │   ├── components/        # Componentes específicos
│   │   │   ├── models/            # Modelos del feature
│   │   │   └── services/          # Servicios del feature
│   │   ├── feature1.module.ts     # Módulo del feature
│   │   ├── feature1-routing.module.ts
│   │   └── index.ts               # Exportaciones
│   └── feature2/                  # Feature Module 2
│
├── app.module.ts                  # Módulo principal
├── app-routing.module.ts          # Routing principal
├── app.component.*                # Componente raíz
└── index.ts                       # Exportaciones principales
```

Principios de Arquitectura

1. **Separación de Responsabilidades**: Cada módulo tiene una responsabilidad específica
2. **Lazy Loading**: Los feature modules se cargan bajo demanda
3. **Core Module Singleton**: Se importa solo en AppModule
4. **Shared Module Reutilizable**: Se puede importar en cualquier módulo
5. **Feature Modules Independientes**: Cada feature es autónomo

---

Organización de Módulos

Core Module (Singleton)

```
// core/core.module.ts
import { NgModule, Optional, SkipSelf } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HTTP_INTERCEPTORS } from '@angular/common/http';

import { LoggerService } from './services/logger.service';
import { AuthInterceptor } from './interceptors/auth.interceptor';
import { AuthGuard } from './guards/auth.guard';

@NgModule({
    declarations: [],
    imports: [CommonModule],
    providers: [
        LoggerService,
        {
            provide: HTTP_INTERCEPTORS,
            useClass: AuthInterceptor,
            multi: true
        },
        AuthGuard
    ]
})
export class CoreModule {
    constructor(@Optional() @SkipSelf() parentModule: CoreModule) {
        if (parentModule) {
            throw new Error(
                'CoreModule ya está cargado. Solo debe importarse en AppModule.'
            );
        }
    }
}
```

Shared Module

```
// shared/shared.module.ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { LoadingSpinnerComponent } from './components/loading-spinner/loading-spinner.component';
import { ConfirmDialogComponent } from './components/confirm-dialog/confirm-dialog.component';

@NgModule({
    declarations: [LoadingSpinnerComponent, ConfirmDialogComponent],
    imports: [CommonModule, FormsModule, ReactiveFormsModule],
    exports: [
        CommonModule,
        FormsModule,
        ReactiveFormsModule,
        LoadingSpinnerComponent,
        ConfirmDialogComponent
    ]
})
export class SharedModule {}
```

Feature Module

```
// features/user/user.module.ts
import { NgModule } from '@angular/core';
import { SharedModule } from '../../shared/shared.module';

import { UserListComponent } from './pages/user-list/user-list.component';
import { UserDetailComponent } from './pages/user-detail/user-detail.component';
import { UserCardComponent } from './shared/components/user-card/user-card.component';
import { UserService } from './shared/services/user.service';

import { UserRoutingModule } from './user-routing.module';

@NgModule({
    declarations: [UserListComponent, UserDetailComponent, UserCardComponent],
    imports: [SharedModule, UserRoutingModule],
    providers: [UserService]
})
export class UserModule {}
```

Componentes

Estructura de Componente

```
// components/user-card/user-card.component.ts
import {
    Component,
    Input,
    Output,
    EventEmitter,
    OnInit,
    OnDestroy
} from '@angular/core';
import { User } from '../../models/user.model';

@Component({
    selector: 'app-user-card',
    templateUrl: './user-card.component.html',
    styleUrls: ['./user-card.component.css'],
    changeDetection: ChangeDetectionStrategy.OnPush
})
export class UserCardComponent implements OnInit, OnDestroy {
    @Input() user!: User;
    @Input() showActions = true;
    @Output() userSelected = new EventEmitter<User>();
    @Output() userDeleted = new EventEmitter<number>();

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

    constructor() {}

    ngOnInit(): void {
        // Inicialización del componente
    }

    ngOnDestroy(): void {
        this.destroy$.next();
        this.destroy$.complete();
    }

    onUserSelect(): void {
        this.userSelected.emit(this.user);
    }

    onUserDelete(): void {
        this.userDeleted.emit(this.user.id);
    }
}
```

Template del Componente

```
<!-- user-card.component.html -->
<div class="user-card" [class.selected]="user.isSelected">
    <div class="user-avatar">
        <img [src]="user.avatar" [alt]="user.name" (error)="onImageError($event)" />
    </div>

    <div class="user-info">
        <h3 class="user-name">{{ user.name }}</h3>
        <p class="user-email">{{ user.email }}</p>
        <span class="user-role" [class]="'role-' + user.role">{{ user.role }}</span>
    </div>

    <div class="user-actions" *ngIf="showActions">
        <button
            type="button"
            class="btn btn-primary"
            (click)="onUserSelect()"
            [disabled]="user.isLoading"
        >
            Ver Detalles
        </button>

        <button
            type="button"
            class="btn btn-danger"
            (click)="onUserDelete()"
            [disabled]="user.isLoading"
        >
            Eliminar
        </button>
    </div>

    <div class="loading-overlay" *ngIf="user.isLoading">
        <app-loading-spinner></app-loading-spinner>
    </div>
</div>
```

Buenas Prácticas para Componentes

1. **OnPush Change Detection**: Usar cuando sea posible para mejor performance
2. **Input/Output Properties**: Usar para comunicación padre-hijo
3. **Lifecycle Hooks**: Implementar OnInit y OnDestroy apropiadamente
4. **Unsubscribe Pattern**: Usar Subject para cancelar suscripciones
5. **Template Reference Variables**: Usar para acceso directo a elementos
6. **TrackBy Functions**: Usar en *ngFor para optimizar rendering

```
// Ejemplo de trackBy function
trackByUserId(index: number, user: User): number {
  return user.id;
}
```

🔧 Servicios

Estructura de Servicio

```
// services/user.service.ts
import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse } from '@angular/common/http';
import { Observable, throwError, BehaviorSubject } from 'rxjs';
import { catchError, map, tap } from 'rxjs/operators';

import { User } from '../models/user.model';
import { environment } from '../../environments/environment';

@Injectable({
```

```
    providedIn: 'root'
})
export class UserService {
    private apiUrl = `${environment.apiUrl}/users`;
    private usersSubject = new BehaviorSubject<User[]>([]);
    public users$ = this.usersSubject.asObservable();

    constructor(private http: HttpClient) {}

    getUsers(): Observable<User[]> {
        return this.http.get<User[]>(this.apiUrl).pipe(
            tap((users) => this.usersSubject.next(users)),
            catchError(this.handleError)
        );
    }

    getUserById(id: number): Observable<User> {
        return this.http
            .get<User>(`${this.apiUrl}/${id}`)
            .pipe(catchError(this.handleError));
    }

    createUser(user: Partial<User>): Observable<User> {
        return this.http.post<User>(this.apiUrl, user).pipe(
            tap((newUser) => {
                const currentUsers = this.usersSubject.value;
                this.usersSubject.next([...currentUsers, newUser]);
            }),
            catchError(this.handleError)
        );
    }

    updateUser(id: number, user: Partial<User>): Observable<User> {
        return this.http.put<User>(`${this.apiUrl}/${id}`, user).pipe(
            tap((updatedUser) => {
                const currentUsers = this.usersSubject.value;
                const updatedUsers = currentUsers.map((u) =>
                    u.id === id ? updatedUser : u
                );
                this.usersSubject.next(updatedUsers);
            }),
            catchError(this.handleError)
        );
    }

    deleteUser(id: number): Observable<void> {
        return this.http.delete<void>(`${this.apiUrl}/${id}`).pipe(
            tap(() => {
                const currentUsers = this.usersSubject.value;
                const filteredUsers = currentUsers.filter((u) => u.id !== id);
                this.usersSubject.next(filteredUsers);
            }),
            catchError(this.handleError)
        );
    }

    private handleError(error: HttpErrorResponse): Observable<never> {
        let errorMessage = 'Ocurrió un error desconocido';

        if (error.error instanceof ErrorEvent) {
            // Error del cliente
            errorMessage = `Error: ${error.error.message}`;
        } else {
            // Error del servidor
            errorMessage = `Código: ${error.status}\nMensaje: ${error.message}`;
        }

        console.error(errorMessage);
        return throwError(() => new Error(errorMessage));
```

```
    }
}
```

Patrones de Servicios

1. **Singleton Services**: Usar `providedIn: 'root'`
2. **State Management**: Usar BehaviorSubject para estado local
3. **Error Handling**: Implementar manejo centralizado de errores
4. **Caching**: Implementar cache para datos frecuentemente usados
5. **Loading States**: Manejar estados de carga

Routing

Configuración de Rutas

```
// app-routing.module.ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { AuthGuard } from './core/guards/auth.guard';

const routes: Routes = [
    {
        path: '',
        redirectTo: 'dashboard',
        pathMatch: 'full'
    },
    {
        path: 'dashboard',
        loadChildren: () =>
            import('./features/dashboard/dashboard.module').then(
                (m) => m.DashboardModule
            ),
        canActivate: [AuthGuard]
    },
    {
        path: 'users',
        loadChildren: () =>
            import('./features/user/user.module').then((m) => m.UserModule),
        canActivate: [AuthGuard],
        data: { roles: ['admin', 'manager'] }
    },
    {
        path: 'auth',
        loadChildren: () =>
            import('./features/auth/auth.module').then((m) => m.AuthModule)
    },
    {
        path: '**',
        redirectTo: 'dashboard'
    }
];

@NgModule({
    imports: [
        RouterModule.forRoot(routes, {
            preloadingStrategy: PreloadAllModules,
            scrollPositionRestoration: 'enabled'
        })
    ],
    exports: [RouterModule]
})
export class AppRoutingModule {}
```

Feature Routing

```
// features/user/user-routing.module.ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { UserListComponent } from './pages/user-list/user-list.component';
import { UserDetailComponent } from './pages/user-detail/user-detail.component';
import { UserCreateComponent } from './pages/user-create/user-create.component';
import { UserEditComponent } from './pages/user-edit/user-edit.component';

const routes: Routes = [
    {
        path: '',
        component: UserListComponent
    },
    {
        path: 'create',
        component: UserCreateComponent
    },
    {
        path: ':id',
        component: UserDetailComponent
    },
    {
        path: ':id/edit',
        component: UserEditComponent
    }
];

@NgModule({
    imports: [RouterModule.forChild(routes)],
    exports: [RouterModule]
})
export class UserRoutingModule {}
```

Guards

```ts
// guards/auth.guard.ts
import { Injectable } from '@angular/core';
import {
    CanActivate,
    Router,
    ActivatedRouteSnapshot,
    RouterStateSnapshot
} from '@angular/router';
import { Observable } from 'rxjs';
import { map, take } from 'rxjs/operators';
import { AuthService } from '../services/auth.service';

@Injectable({
    providedIn: 'root'
})
export class AuthGuard implements CanActivate {
    constructor(
        private authService: AuthService,
        private router: Router
    ) {}

    canActivate(
        route: ActivatedRouteSnapshot,
        state: RouterStateSnapshot
    ): Observable<boolean> {
        return this.authService.isAuthenticated$.pipe(
            take(1),
            map((isAuthenticated) => {
                if (isAuthenticated) {
                    return true;
                } else {
                    this.router.navigate(['/auth/login'], {
                        queryParams: { returnUrl: state.url }
                    });
                    return false;
                }
            })
        );
    }
}
```

Formularios

Reactive Forms

```ts
// components/user-form/user-form.component.ts
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { User } from '../../models/user.model';

@Component({
    selector: 'app-user-form',
    templateUrl: './user-form.component.html'
})
export class UserFormComponent implements OnInit {
    userForm!: FormGroup;
    isSubmitting = false;

    constructor(
        private fb: FormBuilder,
        private userService: UserService
    ) {}

    ngOnInit(): void {
        this.initForm();
    }
```

```
    private initForm(): void {
        this.userForm = this.fb.group({
            name: ['', [Validators.required, Validators.minLength(2)]],
            email: ['', [Validators.required, Validators.email]],
            role: ['user', Validators.required],
            isActive: [true]
        });
    }

    onSubmit(): void {
        if (this.userForm.valid) {
            this.isSubmitting = true;
            const userData = this.userForm.value;

            this.userService.createUser(userData).subscribe({
                next: (user) => {
                    console.log('Usuario creado:', user);
                    this.userForm.reset();
                    this.isSubmitting = false;
                },
                error: (error) => {
                    console.error('Error al crear usuario:', error);
                    this.isSubmitting = false;
                }
            });
        } else {
            this.markFormGroupTouched();
        }
    }

    private markFormGroupTouched(): void {
        Object.keys(this.userForm.controls).forEach((key) => {
            const control = this.userForm.get(key);
            control?.markAsTouched();
        });
    }

    getErrorMessage(controlName: string): string {
        const control = this.userForm.get(controlName);

        if (control?.hasError('required')) {
            return 'Este campo es requerido';
        }

        if (control?.hasError('email')) {
            return 'Ingrese un email válido';
        }

        if (control?.hasError('minlength')) {
            return `Mínimo ${control.errors?.['minlength'].requiredLength} caracteres`;
        }

        return '';
    }
}
```

Template del Formulario

```
<!-- user-form.component.html -->
<form [formGroup]="userForm" (ngSubmit)="onSubmit()" class="user-form">
    <div class="form-group">
        <label for="name">Nombre</label>
        <input
            type="text"
            id="name"
            formControlName="name"
            class="form-control"
            [class.is-invalid]="userForm.get('name')?.invalid && userForm.get('name')?.touched"
        />
```

```
        <div
            class="invalid-feedback"
            *ngIf="userForm.get('name')?.invalid && userForm.get('name')?.touched"
        >
            {{ getErrorMessage('name') }}
        </div>
    </div>

    <div class="form-group">
        <label for="email">Email</label>
        <input
            type="email"
            id="email"
            formControlName="email"
            class="form-control"
            [class.is-invalid]="userForm.get('email')?.invalid && userForm.get('email')?.touched"
        />

        <div
            class="invalid-feedback"
            *ngIf="userForm.get('email')?.invalid && userForm.get('email')?.touched"
        >
            {{ getErrorMessage('email') }}
        </div>
    </div>

    <div class="form-group">
        <label for="role">Rol</label>
        <select id="role" formControlName="role" class="form-control">
            <option value="user">Usuario</option>
            <option value="admin">Administrador</option>
            <option value="manager">Manager</option>
        </select>
    </div>

    <div class="form-group">
        <div class="form-check">
            <input
                type="checkbox"
                id="isActive"
                formControlName="isActive"
                class="form-check-input"
            />
            <label class="form-check-label" for="isActive"> Usuario Activo </label>
        </div>
    </div>

    <div class="form-actions">
        <button
            type="submit"
            class="btn btn-primary"
            [disabled]="userForm.invalid || isSubmitting"
        >
            {{ isSubmitting ? 'Guardando...' : 'Guardar Usuario' }}
        </button>

        <button type="button" class="btn btn-secondary" (click)="userForm.reset()">
            Limpiar
        </button>
    </div>
</form>
```

Manejo de Estado

State Management con RxJS

```typescript
// services/app-state.service.ts
import { Injectable } from '@angular/core';
import { BehaviorSubject, Observable, combineLatest } from 'rxjs';
import { map } from 'rxjs/operators';

interface AppState {
    users: User[];
    selectedUser: User | null;
    isLoading: boolean;
    error: string | null;
}

const initialState: AppState = {
    users: [],
    selectedUser: null,
    isLoading: false,
    error: null
};

@Injectable({
    providedIn: 'root'
})
export class AppStateService {
    private state$ = new BehaviorSubject<AppState>(initialState);

    // Selectors
    users$ = this.state$.pipe(map((state) => state.users));
    selectedUser$ = this.state$.pipe(map((state) => state.selectedUser));
    isLoading$ = this.state$.pipe(map((state) => state.isLoading));
    error$ = this.state$.pipe(map((state) => state.error));

    // Computed selectors
    activeUsers$ = this.users$.pipe(
        map((users) => users.filter((user) => user.isActive))
    );

    userCount$ = this.users$.pipe(map((users) => users.length));

    // Actions
    setUsers(users: User[]): void {
        this.updateState({ users });
    }

    setSelectedUser(user: User | null): void {
        this.updateState({ selectedUser: user });
    }

    setLoading(isLoading: boolean): void {
        this.updateState({ isLoading });
    }

    setError(error: string | null): void {
        this.updateState({ error });
    }

    private updateState(partial: Partial<AppState>): void {
        this.state$.next({
            ...this.state$.value,
            ...partial
        });
    }
}
```

NgRx (Para aplicaciones complejas)

```typescript
// store/actions/user.actions.ts
import { createAction, props } from '@ngrx/store';
import { User } from '../../models/user.model';

export const loadUsers = createAction('[User] Load Users');
export const loadUsersSuccess = createAction(
    '[User] Load Users Success',
    props<{ users: User[] }>()
);
export const loadUsersFailure = createAction(
    '[User] Load Users Failure',
    props<{ error: string }>()
);

export const createUser = createAction(
    '[User] Create User',
    props<{ user: Partial<User> }>()
);
export const createUserSuccess = createAction(
    '[User] Create User Success',
    props<{ user: User }>()
);
export const createUserFailure = createAction(
    '[User] Create User Failure',
    props<{ error: string }>()
);
```

Performance

OnPush Change Detection

```typescript
@Component({
    selector: 'app-user-list',
    templateUrl: './user-list.component.html',
    changeDetection: ChangeDetectionStrategy.OnPush
})
export class UserListComponent {
    @Input() users: User[] = [];

    trackByUserId(index: number, user: User): number {
        return user.id;
    }
}
```

Lazy Loading

```typescript
// app-routing.module.ts
const routes: Routes = [
    {
        path: 'users',
        loadChildren: () =>
            import('./features/user/user.module').then((m) => m.UserModule)
    }
];
```

Virtual Scrolling

```html
<!-- Para listas grandes -->
<cdk-virtual-scroll-viewport itemSize="50" class="user-list">
    <div
        *cdkVirtualFor="let user of users; trackBy: trackByUserId"
        class="user-item"
    >
        <app-user-card [user]="user"></app-user-card>
    </div>
</cdk-virtual-scroll-viewport>
```

Memoización

```typescript
// Pipes personalizados para cálculos costosos
@Pipe({
    name: 'expensiveCalculation',
    pure: true
})
export class ExpensiveCalculationPipe implements PipeTransform {
    transform(value: any): any {
        // Cálculo costoso que se cachea automáticamente
        return this.performExpensiveCalculation(value);
    }

    private performExpensiveCalculation(value: any): any {
        // Implementación del cálculo
    }
}
```

Testing

Unit Testing de Componentes

```typescript
// user-card.component.spec.ts
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { UserCardComponent } from './user-card.component';
import { User } from '../../models/user.model';

describe('UserCardComponent', () => {
    let component: UserCardComponent;
    let fixture: ComponentFixture<UserCardComponent>;

    const mockUser: User = {
        id: 1,
        name: 'John Doe',
        email: 'john@example.com',
        role: 'user',
        isActive: true
    };

    beforeEach(async () => {
        await TestBed.configureTestingModule({
            declarations: [UserCardComponent]
        }).compileComponents();

        fixture = TestBed.createComponent(UserCardComponent);
        component = fixture.componentInstance;
        component.user = mockUser;
        fixture.detectChanges();
    });

    it('should create', () => {
        expect(component).toBeTruthy();
    });

    it('should display user name', () => {
        const compiled = fixture.nativeElement;
        expect(compiled.querySelector('.user-name').textContent).toContain(
            'John Doe'
        );
    });

    it('should emit userSelected event when select button is clicked', () => {
        spyOn(component.userSelected, 'emit');
        const selectButton = fixture.nativeElement.querySelector('.btn-primary');

        selectButton.click();

        expect(component.userSelected.emit).toHaveBeenCalledWith(mockUser);
    });
});
```

Testing de Servicios

```
// user.service.spec.ts
import { TestBed } from '@angular/core/testing';
import {
    HttpClientTestingModule,
    HttpTestingController
} from '@angular/common/http/testing';
import { UserService } from './user.service';
import { User } from '../models/user.model';

describe('UserService', () => {
    let service: UserService;
    let httpMock: HttpTestingController;

    beforeEach(() => {
        TestBed.configureTestingModule({
            imports: [HttpClientTestingModule],
            providers: [UserService]
        });

        service = TestBed.inject(UserService);
        httpMock = TestBed.inject(HttpTestingController);
    });

    afterEach(() => {
        httpMock.verify();
    });

    it('should be created', () => {
        expect(service).toBeTruthy();
    });

    it('should retrieve users from API', () => {
        const mockUsers: User[] = [
            {
                id: 1,
                name: 'John',
                email: 'john@example.com',
                role: 'user',
                isActive: true
            },
            {
                id: 2,
                name: 'Jane',
                email: 'jane@example.com',
                role: 'admin',
                isActive: true
            }
        ];

        service.getUsers().subscribe((users) => {
            expect(users).toEqual(mockUsers);
        });

        const req = httpMock.expectOne('api/users');
        expect(req.request.method).toBe('GET');
        req.flush(mockUsers);
    });
});
```

E2E Testing

```
// user-management.e2e-spec.ts
import { browser, logging } from 'protractor';
import { UserManagementPage } from './user-management.po';

describe('User Management E2E', () => {
    let page: UserManagementPage;

    beforeEach(() => {
        page = new UserManagementPage();
    });

    it('should display user list', () => {
        page.navigateTo();
        expect(page.getUserListTitle()).toEqual('Usuarios');
    });

    it('should create a new user', () => {
        page.navigateTo();
        page.clickCreateUserButton();
        page.fillUserForm('John Doe', 'john@example.com', 'user');
        page.clickSaveButton();

        expect(page.getSuccessMessage()).toContain('Usuario creado exitosamente');
    });

    afterEach(async () => {
        const logs = await browser.manage().logs().get(logging.Type.BROWSER);
        expect(logs).not.toContain(
            jasmine.objectContaining({
                level: logging.Level.SEVERE
            })
        );
    });
});
```

Seguridad

XSS Prevention

```
// Sanitización de contenido
import { DomSanitizer, SafeHtml } from '@angular/platform-browser';

@Component({...})
export class SafeContentComponent {
  constructor(private sanitizer: DomSanitizer) {}

  getSafeHtml(content: string): SafeHtml {
    return this.sanitizer.bypassSecurityTrustHtml(content);
  }
}
```

CSRF Protection

```typescript
// Interceptor para CSRF
@Injectable()
export class CsrfInterceptor implements HttpInterceptor {
    intercept(
        req: HttpRequest<any>,
        next: HttpHandler
    ): Observable<HttpEvent<any>> {
        const token = this.getCsrfToken();

        if (token) {
            req = req.clone({
                setHeaders: {
                    'X-CSRF-Token': token
                }
            });
        }

        return next.handle(req);
    }

    private getCsrfToken(): string | null {
        return (
            document
                .querySelector('meta[name="csrf-token"]')
                ?.getAttribute('content') || null
        );
    }
}
```

Content Security Policy

```html
<!-- index.html -->
<meta
    http-equiv="Content-Security-Policy"
    content="default-src 'self'; script-src 'self' 'unsafe-inline'; style-src 'self' 'unsafe-inline';"
/>
```

Convenciones de Código

Nomenclatura

```typescript
// Archivos
user-list.component.ts          // kebab-case
user-detail.component.ts
user.service.ts
user.model.ts
user.guard.ts

// Clases
export class UserListComponent   // PascalCase
export class UserService
export class UserModel

// Variables y métodos
const userList: User[] = [];     // camelCase
const isLoading = false;
getUserById(id: number): User { }

// Constantes
const API_BASE_URL = 'https://api.example.com';  // UPPER_SNAKE_CASE
const MAX_RETRY_ATTEMPTS = 3;

// Interfaces
interface UserResponse { }       // PascalCase
interface CreateUserRequest { }
```

Imports Organizados

```typescript
// 1. Angular imports
import { Component, OnInit, OnDestroy } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { Router, ActivatedRoute } from '@angular/router';

// 2. Third-party libraries
import { Observable, Subject } from 'rxjs';
import { map, takeUntil } from 'rxjs/operators';

// 3. Application imports
import { User } from '../../models/user.model';
import { UserService } from '../../services/user.service';
import { LoggerService } from '../../core/services/logger.service';

// 4. Relative imports
import { UserCardComponent } from '../user-card/user-card.component';
```

Comentarios

```typescript
/**
 * Servicio para gestionar usuarios del sistema
 * Proporciona métodos CRUD para operaciones con usuarios
 */
@Injectable({
  providedIn: 'root'
})
export class UserService {
  /**
   * Obtiene la lista de usuarios activos
   * @param page Número de página (opcional, por defecto 1)
   * @param limit Límite de elementos por página (opcional, por defecto 10)
   * @returns Observable con la lista de usuarios
   */
  getUsers(page: number = 1, limit: number = 10): Observable<UserResponse> {
    // Implementación
  }

  // Comentario de una línea para lógica compleja
  const filteredUsers = users.filter(user => user.isActive && user.role === 'admin');
  }
}
```

🛠️ Herramientas y Configuración

ESLint Configuration

```
// .eslintrc.json
{
    "extends": [
        "@angular-eslint/recommended",
        "@angular-eslint/template/recommended"
    ],
    "rules": {
        "@angular-eslint/component-selector": [
            "error",
            {
                "type": "element",
                "prefix": "app",
                "style": "kebab-case"
            }
        ],
        "@angular-eslint/directive-selector": [
            "error",
            {
                "type": "attribute",
                "prefix": "app",
                "style": "camelCase"
            }
        ],
        "@typescript-eslint/no-unused-vars": "error",
        "@typescript-eslint/explicit-function-return-type": "warn"
    }
}
```

Prettier Configuration

```
// .prettierrc
{
    "semi": true,
    "trailingComma": "es5",
    "singleQuote": true,
    "printWidth": 80,
    "tabWidth": 2,
    "useTabs": false
}
```

Angular CLI Configuration

```
// angular.json
{
    "projects": {
        "my-app": {
            "architect": {
                "build": {
                    "options": {
                        "optimization": true,
                        "outputHashing": "all",
                        "sourceMap": false,
                        "namedChunks": false,
                        "aot": true,
                        "extractLicenses": true,
                        "vendorChunk": false,
                        "buildOptimizer": true
                    }
                }
            }
        }
    }
}
```

Environment Configuration

```typescript
// environments/environment.ts
export const environment = {
    production: false,
    apiUrl: 'http://localhost:3000/api',
    appName: 'Mi Aplicación',
    version: '1.0.0',
    enableDebug: true
};

// environments/environment.prod.ts
export const environment = {
    production: true,
    apiUrl: 'https://api.miapp.com',
    appName: 'Mi Aplicación',
    version: '1.0.0',
    enableDebug: false
};
```

Recursos Adicionales

Documentación Oficial

- Angular Style Guide (https://angular.io/guide/styleguide)
- Angular Architecture (https://angular.io/guide/architecture)
- Angular Testing (https://angular.io/guide/testing)

Herramientas Recomendadas

- **Angular CLI**: Herramienta oficial de línea de comandos
- **Angular DevTools**: Extensión de Chrome para debugging
- **Prettier**: Formateador de código
- **ESLint**: Linter para TypeScript/JavaScript
- **Husky**: Git hooks para pre-commit
- **Lint-staged**: Linting solo de archivos modificados

Patrones de Diseño

- **Observer Pattern**: RxJS Observables
- **Factory Pattern**: Servicios de creación
- **Singleton Pattern**: Core Module
- **Strategy Pattern**: Guards y Interceptors
- **Decorator Pattern**: Angular Decorators

Desarrollado siguiendo las mejores prácticas de Angular y TypeScript