```
/project-root
├── src/
│   ├── app/
│   │   ├── core/              # Servicios y funcionalidades globales
│   │   │   ├── http/            # Interceptores HTTP
│   │   │   ├── auth/            # Opcional solo para contextos de autenticación grandes(session.service.ts, token-manager.service.ts, refresh.service.ts, etc.)
│   │   │   ├── guards/          # Guards de rutas (auth.guard.ts)
│   │   │   ├── services/        # Servicios singleton (auth.service.ts, config, storage...)
│   │   │   └── models/          # Modelos/interfaces globales
│   │   │   └── tokens/          # Injection tokens, por si necesitas abstraer servicios (auth.token.ts)
│   │   │   └── state/           # <--- NUEVO: Gestión global de estado reactivo (opcional si se requiere)
│   │   │      ├── stores/       # Stores globales con signals (por dominio: user.store.ts, ui.store.ts, etc.) Ejemplo código abajo
│   │   │
│   │   ├── shared/            # Componentes y utilitarios reutilizables
│   │   │   ├── components/       # Componentes standalone compartidos
│   │   │   ├── directives/       # Directivas standalone
│   │   │   ├── pipes/            # Pipes standalone
│   │   │   └── utils/            # Funciones utilitarias
│   │   │
│   │   ├── features/          # Funcionalidades principales (Agrupadas por dominio Instalaciones, Expedientes, Documentos)
│   │   │   ├── feature-a/           # Característica A
│   │   │   │   ├── components/   # Componentes específicos de la característica
│   │   │   │   ├── services/     # Servicios específicos de la característica
│   │   │   │   ├── models/       # Modelos/interfaces
│   │   │   │   ├── routes.ts     # Rutas de la característica
│   │   │   │   └── feature-a.component.ts  # Componente principal
│   │   │   │
│   │   │   └── feature-b/       # Característica B (misma estructura)
│   │   │
│   │   ├── layout/            # Componentes de layout (estructura)
│   │   │   ├── header/
│   │   │   ├── footer/
│   │   │   ├── sidebar/
│   │   │   └── layout.component.ts     # <router-outlet></router-outlet>
│   │   │
│   │   ├── routes.ts          # Rutas principales de la aplicación (lazy load)
│   │   ├── app.config.ts        # Configuración de la app (providers globales)
│   │   ├── app.component.ts     # Componente raíz
│   │   └── app.component.html   # Template del componente raíz
│   │
│   ├── assets/                # Recursos estáticos
│   │   ├── images/
│   │   ├── icons/
│   │   └── fonts/
│   │
│   ├── environments/          # Configuraciones por entorno
│   │   ├── environment.ts
│   │   └── environment.prod.ts
│   │
│   ├── styles/                # Estilos globales
│   │   ├── global.scss
│   │   ├── variables.scss
│   │   └── themes/
│   │
│   ├── index.html
│   └── main.ts
│
├── angular.json
├── package.json
├── tsconfig.json
└── README.md
```

Características principales de esta arquitectura

1. **Enfoque Standalone**: Todos los componentes, directivas y pipes son standalone, eliminando la necesidad de módulos NgModule.
2. **Carga perezosa**: Las rutas están configuradas para cargar las características bajo demanda.
3. **Core**: Contiene servicios y funcionalidades esenciales que se cargan una sola vez en la aplicación.
4. **Shared**: Componentes reutilizables que pueden ser importados directamente donde se necesiten.
5. **Features**: Organización por funcionalidades de negocio, cada una con su propia estructura interna.

Store opcional cuando sea necesario para estado global, basado en Signals + Services**:**

```
src/app/core/state/stores/user.store.ts

import { Injectable, signal, computed } from '@angular/core';
import { User } from '../../models/user.model';

@Injectable({ providedIn: 'root' })
export class UserStore {
  private readonly _user = signal<User | null>(null);

  readonly user = computed(() => this._user());
  readonly isLoggedIn = computed(() => !!this._user());

  setUser(user: User) {
    this._user.set(user);
  }

  clearUser() {
    this._user.set(null);
  }
}
```

Ejemplo de uso consumir el Store:

```
// user-profile.component.ts
import { Component, inject, signal } from '@angular/core';
import { CommonModule } from '@angular/common';
import { UserStore } from '../../core/services/user.store';

@Component({
  selector: 'app-user-profile',
  standalone: true,
  imports: [CommonModule],
  template: `
    <ng-container *ngIf="user(); else notLoggedIn">
      <h2>Bienvenido, {{ user()?.name }}</h2>
      <p>Email: {{ user()?.email }}</p>
      <button (click)="logout()">Cerrar sesión</button>
    </ng-container>

    <ng-template #notLoggedIn>
      <p>No has iniciado sesión.</p>
    </ng-template>
  `,
})
export class UserProfileComponent {
```

```
  private readonly userStore = inject(UserStore);

  // Signals computados
  readonly user = this.userStore.user;
  readonly isLoggedIn = this.userStore.isLoggedIn;

  logout() {
    this.userStore.clearUser();
  }
}
```

Cómo setear el usuario tras login (por ejemplo en un login.component.ts)

```
login() {
  const fakeUser: User = {
    id: 1,
    name: 'Juan Pérez',
    email: 'juan@example.com',
    role: 'admin'
  };

  this.userStore.setUser(fakeUser);
}
```

Estructura de Carpetas con Redux para Angular

```
/project-root
├── src/
│   ├── app/
│   │   ├── core/
│   │   │   ├── auth/
│   │   │   ├── http/
│   │   │   ├── guards/
│   │   │   ├── services/
│   │   │   └── models/
│   │   │
│   │   ├── shared/
│   │   │   ├── components/
│   │   │   ├── directives/
│   │   │   ├── pipes/
│   │   │   └── utils/
│   │   │
│   │   ├── state/                # Directorio principal para NgRx
│   │   │   ├── actions/          # Acciones de Redux
│   │   │   │   ├── index.ts      # Exportaciones de acciones
│   │   │   │   ├── auth.actions.ts
│   │   │   │   └── ...
│   │   │   │
│   │   │   ├── reducers/         # Reducers
│   │   │   │   ├── index.ts      # Reducer principal combinado
│   │   │   │   ├── auth.reducer.ts
│   │   │   │   └── ...
│   │   │   │
│   │   │   ├── effects/          # Side Effects
│   │   │   │   ├── index.ts      # Exportaciones de effects
│   │   │   │   ├── auth.effects.ts
│   │   │   │   └── ...
│   │   │   │
│   │   │   ├── selectors/        # Selectores
│   │   │   │   ├── index.ts      # Exportaciones de selectores
│   │   │   │   ├── auth.selectors.ts
│   │   │   │   └── ...
│   │   │   │
│   │   │   └── models/           # Modelos/interfaces para el estado
│   │   │       ├── app.state.ts  # Interface del estado global
│   │   │       └── ...
│   │   │
│   │   ├── features/
│   │   │   ├── feature-a/
│   │   │   │   ├── components/
│   │   │   │   ├── services/
│   │   │   │   ├── state/        # Estado específico de la feature
│   │   │   │   │   ├── actions.ts   # Acciones
│   │   │   │   │   ├── reducer.ts   # Reducer
│   │   │   │   │   ├── effects.ts   # Effects
│   │   │   │   │   └── selectors.ts # Selectores
│   │   │   │   ├── routes.ts
│   │   │   │   └── feature-a.component.ts
│   │   │   │
│   │   │   └── feature-b/        # Estructura similar
│   │   │
│   │   ├── layout/
│   │   │   ├── header/
│   │   │   ├── footer/
│   │   │   ├── sidebar/
│   │   │   └── layout.component.ts
│   │   │
│   │   ├── routes.ts
│   │   ├── app.config.ts         # Incluye providers de NgRx
│   │   ├── app.component.ts
│   │   └── app.component.html
│   │
│   └── ...resto igual
```