Estructura del Proyecto

1. **`src/app/api`**
    - Este directorio es ideal para centralizar las llamadas a APIs externas o servicios. Aquí podrías organizar servicios que se comuniquen con el backend, como obtener productos, categorías, o manejar la lógica del carrito.

2. **`src/app/features`**
    - Contiene las características específicas de tu aplicación. Por ejemplo:
        - `categories`: Maneja todo lo relacionado con las categorías de productos.
        - `checkout`: Incluye la funcionalidad del proceso de compra.
        - `products`: Gestiona la lógica de productos.
    - Este patrón ayuda a encapsular cada funcionalidad de forma modular, facilitando la escalabilidad.

3. **`src/app/layout`**
    - Aquí puedes definir componentes compartidos del diseño general, como encabezados, pies de página, barras de navegación, etc.

4. **`src/app/shared`**
    - Contiene módulos, componentes, directivas, pipes o servicios que son reutilizables en diferentes partes de la aplicación.

5. **`src/app/store`**
    - Este directorio (por ejemplo, `cart-state`) gestiona el estado global usando patrones como Signals, NgRx o RxJS, asegurando que el estado sea consistente en toda la aplicación.

6. **Archivos principales:**
    - **`app.component.*`**: Punto de entrada de la aplicación.
    - **`app.routes.ts`**: Configuración de las rutas (navegación entre módulos/funcionalidades).
    - **`app.config.ts`**: Archivo para manejar configuraciones globales.

Agregar Nuevas Features Usando Standalone Components

1. Crear un Standalone Component

En lugar de generar un módulo, puedes generar directamente un componente standalone usando Angular CLI:

```
ng generate component features/new-feature
```

Esto creará:

- Un nuevo directorio dentro de `features/new-feature`.
- Un componente standalone que incluye la configuración necesaria en su decorador.

El componente se verá algo así:

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

@Component({
  selector: 'app-new-feature',
  standalone: true,
  templateUrl: './new-feature.component.html',
  styleUrls: ['./new-feature.component.scss'],
})
export class NewFeatureComponent {}
```

---

2. Registrar el Componente en las Rutas

En aplicaciones standalone, las rutas son la clave para organizar y cargar las funcionalidades. Agrega la nueva feature al archivo `app.routes.ts`:

```
import { Routes } from '@angular/router';
import{NewFeatureComponent}from
'./features/new-feature/new-feature.component';

export const routes: Routes = [
  { path: 'new-feature', component: NewFeatureComponent },
  // Otras rutas existentes];
```

Con esto, tu aplicación reconocerá el nuevo componente standalone y lo cargará cuando accedas a la ruta `/new-feature`.

---

5. Carga Diferida (Lazy Loading)

Si quieres cargar esta feature solo cuando el usuario acceda a ella, puedes usar `loadComponent` en las rutas:

```
export const routes: Routes = [
  {
    path: 'new-feature',
    loadComponent: () =>
      import('./features/new-feature/new-feature.component').then(
        (m) => m.NewFeatureComponent
      ),
  },
];
```

Este enfoque mejora el rendimiento inicial de la aplicación.

Ventajas del Enfoque Standalone

1. **Menor sobrecarga:** No necesitas definir y mantener múltiples módulos.
2. **Mayor claridad:** Cada componente es independiente y autoconfigurable.
3. **Facilita la carga perezosa:** La API de `loadComponent` hace que la carga diferida sea más directa.