📦 Header Component - Arquitectura Modular

📁 Estructura de Archivos

```
header/
├── header.tsx                    # 🎯 Componente principal (orquestador)
├── header.types.ts               # ✅ Tipos TypeScript
├── header.constants.ts           # ✅ Constantes globales
├── header.utils.ts               # ✅ Funciones auxiliares puras
│
├── components/
│   ├── AccessibleLinks.tsx       # ✅ Links de accesibilidad
│   ├── TopMenu.tsx               # ✅ Menú superior completo
│   ├── MainActions.tsx           # ✅ Barra de acciones
│   │
│   ├── search/
│   │   ├── AlgoliaSearch.tsx     # ✅ Búsqueda con Algolia
│   │   └── SalesforceSearch.tsx  # ✅ Búsqueda con Salesforce
│   │
│   ├── menu/
│   │   ├── MenuItem.tsx          # ✅ Item individual de menú
│   │   └── MenuDrawer.tsx        # ✅ Drawer del menú
│   │
│   └── parts/
│       ├── Logo.tsx              # ✅ Logo del header
│       ├── ShoppingCart.tsx      # ✅ Carrito de compras
│       ├── MyAccountButton.tsx   # ✅ Botón Mi Orange
│       ├── LogoutButton.tsx      # ✅ Botón de logout
│       ├── HamburgerButton.tsx   # ✅ Botón hamburguesa
│       └── SearchButton.tsx      # ✅ Botón de búsqueda
│
└── README.md                     # Este archivo
```

🎯 Ventajas de esta Arquitectura

✅ Mantenibilidad

- Cada componente tiene una responsabilidad única
- Fácil localizar y corregir bugs
- Cambios aislados que no afectan al resto del sistema

✅ Testabilidad

- Componentes funcionales fáciles de testear
- Utilidades puras sin efectos secundarios
- Mocks sencillos para dependencias

✅ Reutilización

- Componentes independientes exportables
- Utilidades reutilizables en otros proyectos
- Constantes compartidas

✅ Legibilidad

- Componente principal ~70% más pequeño
- Nombres descriptivos y claros
- Estructura lógica y predecible

✅ Escalabilidad

- Fácil añadir nuevos tipos de búsqueda
- Nuevos componentes sin modificar el principal
- Configuración centralizada

📚 Componentes Principales

🎯 header.tsx (Principal)

Orquestador que gestiona el estado global y coordina todos los sub-componentes.

Responsabilidades:

- Estado global del header
- Gestión de eventos
- Coordinación de componentes
- Integración con APIs (Algolia, Salesforce)

🔧 header.constants.ts

Constantes centralizadas para mantener consistencia.

```typescript
export const TIMING = {
  TRANSITION: 300,
  RESIZE_DEBOUNCE: 200,
  FOCUS_TRAP_DELAY: 300,
};

export const CSS_CLASSES = {
  HEADER_OPEN: 'hlx-header-open',
  DROPDOWN_SHOW: 'show',
  // ...
};
```

🛠️ header.utils.ts

Funciones puras y reutilizables.

```typescript
// Ejemplo de uso
updateHeaderHeight(element);
const classes = generateBreakpointClasses(['xs', 'sm']);
const isMobile = handleResponsiveLayout(parentWidth, childWidth);
```

🧩 Componentes de UI

🔍 Búsqueda

AlgoliaSearch.tsx

Componente de búsqueda con integración Algolia.

```tsx
<AlgoliaSearch
  searchTerm={searchTerm}
  searchResults={results}
  onInput={() => search()}
  onReset={() => reset()}
/>
```

SalesforceSearch.tsx

Componente de búsqueda con integración Salesforce.

```tsx
<SalesforceSearch
  searchTerm={searchTerm}
  onKeyDown={(e) => handleSearch(e)}
  onReset={() => reset()}
/>
```

🍔 Menú

MenuItem.tsx

Renderiza un item individual de menú con soporte para submenús.

```tsx
<MenuItem
  item={menuItem}
  isMainMenuOpen={isOpen}
  onOpenSubmenu={(e) => open(e)}
  onCloseMenu={() => close()}
/>
```

MenuDrawer.tsx

Drawer lateral que contiene todo el menú de navegación.

```tsx
<MenuDrawer
  menuData={menu}
  isLightThemeActive={isLight}
  searchBoxComponent={<Search />}
  onCloseMenu={() => close()}
/>
```

🔘 Botones y Acciones

ShoppingCart.tsx

```tsx
```

```tsx
<ShoppingCart
  cart={cartData}
  onClickTracking={(data) => track(data)}
/>
```

MyAccountButton.tsx

```tsx
<MyAccountButton
  myAccount={accountData}
  onClickDropdown={(e, data) => toggle(e)}
/>
```

HamburgerButton.tsx

```tsx
<HamburgerButton
  label="Abrir menú"
  onClick={(e) => openMenu(e)}
/>
```

🚀 Uso del Componente

Configuración Básica

```typescript
```

```typescript
// En tu aplicación
const headerElement = document.querySelector('hlx-header');

// Configurar datos
await headerElement.setData({
  config: {
    siteName: 'Mi Sitio',
    algolia: { /* config */ },
    labels: { /* labels */ }
  },
  bottom: {
    left: { logo: { /* logo */ } },
    right: { menu: { /* menu */ } }
  },
  myAccount: { /* account */ },
  searchBox: { /* search */ },
  shoppingCart: { /* cart */ }
});
```

Actualizar Datos Dinámicamente

typescript

```typescript
// Actualizar una propiedad específica
await headerElement.updateData(
  { label: 'Nuevo Label' },
  'myAccount.label'
);
```

Escuchar Eventos

typescript

```typescript
headerElement.addEventListener('clickTracking', (event) => {
  console.log('Tracking:', event.detail);
  // Enviar a analytics
});
```

🎨 Personalización

Temas

tsx

```
// Tema claro en el drawer
<hlx-header drawer-theme-light={true} />

// Header estático (sin sticky)
<hlx-header position-static={true} />
```

Breakpoints Personalizados

typescript

```
// En tu JSON de configuración
{
  "hide": ["xs", "sm"],  // Oculto en móviles
  "drawerOnly": ["xs", "sm", "md"]  // Solo en drawer
}
```

Estilos CSS Personalizados

css

```
/* Variable CSS para altura del header */
.mi-componente {
  margin-top: var(--hlx-header-height);
}

/* Estados del header */
.hlx-header-open { /* Cuando está abierto */ }
.hlx-header-desktop { /* En desktop */ }
.hlx-header-is-resizing { /* Durante resize */ }
```

🧪 Testing

Test de Componentes Funcionales

typescript

```typescript
import { Logo } from './components/parts/Logo';

describe('Logo', () => {
  it('should render logo with correct src', () => {
    const logo = {
      href: '/home',
      label: 'Test',
      image: {
        src: { mobile: 'mobile.svg', desktop: 'desktop.svg' },
        alt: 'Logo'
      }
    };

    const rendered = <Logo logo={logo} />;
    // Asserts...
  });
});
```

Test de Utilidades

typescript

```typescript
import { generateBreakpointClasses } from './header.utils';

describe('generateBreakpointClasses', () => {
  it('should generate correct classes', () => {
    const classes = generateBreakpointClasses(['xs', 'sm']);
    expect(classes).toContain('d-none');
    expect(classes).toContain('d-md-block');
  });
});
```

📝 Convenciones de Código

Nomenclatura de Componentes

- **PascalCase** para componentes: `MyAccountButton.tsx`
- **camelCase** para props: `onClickTracking`
- **SCREAMING_SNAKE_CASE** para constantes: `TIMING.TRANSITION`

Estructura de Props

```typescript
interface ComponentProps {
  // Datos
  data: DataType;

  // Callbacks
  onClick?: (event: any) => void;
  onSomething?: () => void;

  // Funciones auxiliares
  helperFunction?: (param: any) => any;
}
```

Orden de Imports

```typescript
// 1. Stencil
import { h, FunctionalComponent } from '@stencil/core';

// 2. Librerías externas
import classNames from 'classnames';

// 3. Tipos locales
import { MyType } from './types';

// 4. Componentes locales
import { MyComponent } from './components';
```

🔄 Migración desde la Versión Anterior

Paso 1: Copiar Archivos

Copia todos los archivos de la nueva estructura a tu proyecto.

Paso 2: Actualizar Imports

```typescript
```

```
// Antes
import { Header } from './header/header';

// Ahora (igual, pero con dependencias internas)
import { Header } from './header/header';
```

Paso 3: Sin Cambios en la API

✅ La API pública del componente **NO cambia** ✅ Todos los métodos públicos siguen igual ✅ Los JSON de configuración son compatibles

---

⚡ Performance

Optimizaciones Implementadas

1. **Componentes Funcionales**: Más ligeros que componentes de clase
2. **Debouncing**: En resize events para evitar renders excesivos
3. **Lazy Evaluation**: Componentes solo se renderizan si tienen datos
4. **Memoización**: Funciones puras en utils pueden ser memoizadas

Recomendaciones

typescript

```
// ✅ BIEN: Pasar funciones estables
const handleClick = useCallback(() => {
  // ...
}, [dependencies]);

// ❌ MAL: Crear funciones en cada render
onClick={() => doSomething()}
```

🐛 Debugging

Habilitar Logs

typescript

```
// En header.constants.ts, añadir:
export const DEBUG = {
  ENABLED: true,
  LOG_EVENTS: true,
  LOG_STATE_CHANGES: true,
};
```

Inspeccionar Estado

```typescript
// Desde DevTools Console
const header = document.querySelector('hlx-header');
console.log(await header.getState()); // Añadir método si es necesario
```

📚 Recursos Adicionales

- Stencil.js Documentation
- TypeScript Handbook
- Web Components Best Practices

🤝 Contribuir

Añadir un Nuevo Componente

1. Crear archivo en `components/` o `components/parts/`
2. Seguir la estructura de props establecida
3. Exportar como `FunctionalComponent`
4. Importar en `header.tsx` si es necesario
5. Actualizar este README

Ejemplo

```tsx
```

```tsx
// components/parts/NewButton.tsx
import { h, FunctionalComponent } from '@stencil/core';

interface NewButtonProps {
  label: string;
  onClick: () => void;
}

export const NewButton: FunctionalComponent<NewButtonProps> = ({
  label,
  onClick
}) => {
  return (
    <button class="hl-btn" onClick={onClick}>
      {label}
    </button>
  );
};
```

📄 Licencia

Este componente es parte del sistema de diseño de Orange España.

✨ Changelog

v3.0.0 - Arquitectura Modular

- ✅ Refactorización completa en módulos independientes
- ✅ Componentes funcionales para mejor performance
- ✅ Utilidades extraídas a `header.utils.ts`
- ✅ Constantes centralizadas en `header.constants.ts`
- ✅ Mejor testabilidad y mantenibilidad

v2.0.12

- Eliminación del dinamismo del botón "Mi Orange"

¿Preguntas? ¿Problemas? Abre un issue en el repositorio del proyecto.