## **Desarrollo de Frontend con Arquitectura Single Page Application (SPA)**

Las aplicaciones frontend desarrolladas bajo el paradigma de **Single Page Application (SPA)** constituyen una alternativa válida al enfoque de MicroFrontends, y pueden utilizarse cuando los requisitos del dominio, performance, mantenimiento, desacoplamiento y equipo así lo permiten. Esta sección establece los lineamientos que **DEBEN** aplicarse al construir **SPAs React** dentro de la organización, respetando el ecosistema tecnológico compartido.

### **Definición**

Una **Single Page Application (SPA)** es una aplicación web que **DEBE** cargar un único documento HTML que sirve de base para una interfaz de usuario que **DEBE** manipularse dinámicamente mediante JavaScript sin recargar completamente la página . En nuestro contexto, las SPAs **DEBEN** estar desarrolladas con **React 18.2.0** y **TypeScript 4.9.5**, utilizando **Webpack 5** como empacador.

### **Tecnologías Base (Obligatorias)**

La arquitectura SPA **DEBE** implementar las mismas tecnologías obligatorias que en el modelo de MicroFrontend:

* **React 18.2.0**
* **TypeScript 4.9.5**
* **Webpack 5**
* **Atomic Design**
* **ContentStack (CMS)**
* **ItauOne Design System (@itau-one/\*)**
* **CI/CD GitLab Pipelines**

No obstante, la aplicación **NO DEBE** dividirse en múltiples módulos federados o cargados dinámicamente a través de Module Federation. En su lugar, toda la lógica, rutas, vistas y servicios se **DEBE** empaquetar como una unidad desplegable coherente y aislada.

### **Principios Arquitectónicos para SPA**

#### **Modularidad y Dominios**

* La SPA **DEBE** estructurarse en **módulos funcionales desacoplados** (por dominio o feature), empleando **separación explícita de responsabilidades**. Cada módulo funcional (por ejemplo: Autenticación, Dashboard, Reportes) **DEBERÍA** ubicarse en una carpeta raíz src/modules/nombre-modulo.
* Cada módulo funcional **DEBE** contener sus propios componentes atómicos, servicios, rutas internas y pruebas.
* Se **DEBE** aplicar la estructura Atomic Design dentro de cada módulo, replicando los niveles: átomos, moléculas, organismos, plantillas y páginas.

Ejemplo:

src/

modules/

dashboard/

atoms/

molecules/

organisms/

templates/

pages/

services/

routes/

#### **Gestión del Estado**

* La SPA **DEBE** tener una política clara de gestión del estado.  
  + Se **PUEDE** usar Context API, Redux Toolkit, Zustand o Recoil, de forma justificada.
  + El **estado global compartido** entre dominios funcionales **DEBE** estar centralizado y documentado.
  + El estado local o de componente **DEBERÍA** resolverse con useState, useReducer o Context API según el caso.

#### **Enrutamiento**

* Se **DEBE** usar react-router-dom (v6 o superior) como sistema de enrutamiento de la SPA.
* Las rutas deben declararse de forma jerárquica en archivos modulares.
* Se **DEBE** implementar **carga perezosa** (lazy loading) de rutas utilizando React.lazy() y Suspense para dividir el código y optimizar la carga.

Ejemplo:

const Dashboard = React.lazy(() => import('./pages/Dashboard'));

<Route path="/dashboard" element={

<Suspense fallback={<Loading />}>

<Dashboard />

</Suspense>

} />

* Las rutas **NO DEBEN** ser declaradas de forma plana en un único archivo de gran tamaño. Cada módulo funcional **DEBE** declarar sus propias rutas locales, y éstas deben componerse desde un router/index.tsx.

#### **Comunicación entre módulos**

* La SPA **DEBE** limitar la comunicación entre módulos funcionales a través de mecanismos explícitos (props, eventos, contexto compartido).
* **NO DEBE** usarse acoplamiento implícito (importaciones cruzadas sin control).
* Se **DEBE** definir contratos claros entre módulos funcionales mediante interfaces y estructuras de datos tipadas con TypeScript.

### **Integración con ContentStack**

El uso de ContentStack en SPAs **DEBE** seguir las mismas prácticas descritas en la sección de MicroFrontends:

* El cliente SDK **DEBE** inicializarse una única vez (singleton).
* Se **DEBE** construir un ContentService general en src/services/contentstack.ts para encapsular llamadas al CMS.
* Los módulos funcionales **DEBERÍAN** consumir contenido mediante funciones de ese servicio y **NO** directamente con el SDK.

### **Ciclo de Vida y Despliegue**

#### **CI/CD**

* El pipeline **DEBE** validar el código con *linters*, *unit tests* y *build* antes del merge a ramas protegidas (main, release, etc.).
* La SPA **DEBE** empaquetarse como artefacto independiente (por ejemplo: Docker container o carpeta dist/ desplegable).
* El despliegue **DEBE** ser automatizado por GitLab CI/CD a los entornos correspondientes.
* Se **DEBE** incluir análisis de dependencias y seguridad (p.ej. Snyk, Trivy, OWASP tools).

#### **Versionamiento y Releases**

* Se **DEBE** aplicar versionado semántico (SemVer) a la SPA.
* Cada release **DEBE** documentarse y etiquetarse en el repositorio.
* El rollback **DEBE** estar soportado mediante tags o ramas de respaldo.

### **Buenas Prácticas y Mantenibilidad**

* **DEBE** existir una convención de estilos y nombres unificada (nombres de archivos en *kebab-case*, componentes en *PascalCase*, hooks en *camelCase*).
* **DEBEN** usarse herramientas de linting (eslint, prettier, stylelint) con configuración corporativa.
* Se **DEBE** documentar cada componente relevante con Storybook o documentación integrada (JSDoc, Typedoc, etc.).
* El uso de librerías externas **DEBE** ser justificado, revisado y aprobado por arquitectura antes de ser incluido.
* Se **DEBE** aplicar optimización de performance:  
  + Lazy loading de módulos y rutas.
  + Code splitting por chunk.
  + Tree shaking de imports.
  + Caché y almacenamiento local sólo cuando esté controlado.

### **Consideraciones Finales**

La elección de SPA frente a MicroFrontend **DEBE** ser evaluada desde arquitectura considerando:

* Tamaño del equipo
* Escalabilidad futura
* Necesidad de despliegue independiente
* Interoperabilidad con otros módulos
* Dependencias compartidas

Si se opta por SPA, esta **DEBE** cumplir todos los lineamientos aquí definidos, garantizando consistencia con la visión técnica corporativa y mantenibilidad a largo plazo.

¿Quieres que prepare esta nueva sección también en formato PDF o para publicación directa en intranet corporativa (con índice, metadatos, branding, etc.)?