|  |  |
| --- | --- |
| Diseño de la Solución | Proyecto  Microfrontend |
|  | |

|  |  |
| --- | --- |
|  |  |

**NOTA DE CONFIDENCIALIDAD**

La información incluida en este documento ha sido preparada para ser utilizada en el contexto de este proyecto. No debe ser utilizada como modelo o precedente en ninguna situación fuera del presente trabajo.

Este documento no debe ser copiado o reproducido por ningún medio sin la autorización de las partes involucradas.

Se ha realizado un gran esfuerzo en la preparación de este documento para asegurar que la información presentada es correcta y completa.

**INFORMACION DEL DOCUMENTO**

|  |  |  |  |
| --- | --- | --- | --- |
| **Solicitado por:** | Nancy Bermurez / Jeisson Rozo | | |
| **Preparado por** | Nicolás Archila - Pragma | **Revisado por:** |  |
| **Versión documento :** | 1.0 | **Fecha documento:** | 19 de abril de 2021 |

**HISTORIAL DE VERSIONES**

|  |  |  |  |
| --- | --- | --- | --- |
| **Versión** | **Fecha Documento** | **Temas Revisados y Modificados** | **Autor** |
| 1.0 | 19/04/2021 | Documento inicial | Nicolás Archila |
| 1.1 | 19/04/2021 | Anexar especificaciones de Front y esquema de PWA: service worker | Nicolás Archila |

|  |
| --- |
|  |

Tabla de Contenido

[1. Arquitectura Front 4](#_Toc69736536)

[1.1. Diagrama de contexto de Front 5](#_Toc69736537)

[2. Manual de MicroFrontend 6](#_Toc69736538)

[2.1. Implementación de single spa en Angular 6](#_Toc69736539)

[*2.2.* *Instalación de dependencias en proyectos hijos* 6](#_Toc69736540)

[2.4. Configuración de Chasis 10](#_Toc69736541)

[2.4.1. Importándolo en el componente de angular (Recomendado) 11](#_Toc69736542)

[2.4.2. Importándolo en el componente de angular 13](#_Toc69736543)

[3. Comunicación Microfrontends 16](#_Toc69736544)

[4. Repositorio guía 20](#_Toc69736545)

[5. Documentación y links 20](#_Toc69736546)

# Arquitectura Front

Todas las webs deben ser creadas con componentes que sigan las buenas prácticas y estándares de voxel community.

Los archivos estáticos de front se deben almacenar en S3 y exponer a través de CDN de Akamai, todos los labels, banners y textos asociados, deben estar alojados en el CMS, en dónde se deben dejar los .json en una estructura de carpetas intuitiva para buscar los archivos a futuro, ejemplo (“raíz”/microfrontend/onboarding/chasis o “raíz”/microfrontend/pagos/terceros), aunque es claro que la ruta generada para consumir los archivos es automática y númerica. Los labels asociados a errores se deben homologar con modelos en angular que asocien los códigos de error entregados por el backend, de tal manera que se puedan personalizar los copies entregados al usuario final y se puedan modificar fácilmente desde el CMS. Este esquema se puede aplicar para cualquier tipo de respuesta que lo requiera.

Los MicroFrontend deben ser desplegados como js y llamados desde el chasis. Más adelante se puede visualizar un ejemplo de la comunicación entre ellos y el esquema de despliegue.

Cada Microfrontend debe estar compuesto de módulos diferentes y cada módulo de componentes, entre los cuales están los componentes de Voxel Mobile y Voxel community. El obejtivo de los módulos es aplicar el patrón de lazy loading y asociar cada módulo a una ruta, de tal manera que solo se carguen dichos módulos cuando el usuario ingrese a esta ruta y así no recargar el navegador con archivos que el usuario no necesita en ese momento; por otro lado, la aplicación debe ser hecha con el concepto de SPA para que cada en cada página nueva no viaje hasta el servidor, sino que el navegador se encargue de ello, de esta manera toda la aplicación estará en una sola página a nivel de servidor y mejorará notablemente el performance de la aplicación.

Se debe aplicar el concepto de PWA:service worker para cachear la aplicación en el navegador del usuario, mejorar el performance de la aplicación y aprovechar los beneficios de PWA.

Para hacer cualquier llamado a service workers se debe verificar si este esta habilitado y garantizar que si hay alguna falla en el service workers, la aplicación siga funcionando correctamente.

Cómo ya se menciónó cada módulo se debe cargar perezosamente y de igual manera en service workers se debe instalar en modo lazy; por otro lado, se debe mantener la integridad para todos los componentes dependientes; es decir, si tengo dos módulos que trabajan con un mismo js, al actualizar el service worker del js, se deben actualizar los dos módulos, de lo contrario alguno quedará desactualizado y no funcionará de manera correcta, para esto se puede usar el comportamiento de versionamiento de angular service worker.

Todos los llamados a microservicios deben implementar el patrón circuit breaker, se recomienda usar la librería resilience4j.

Se deben contemplar todas las pruebas unitarias con JEST.

Se debe trabajar taggeo y Biocatch de acuerdo con las HU que se presenten, para consumir el getScore de Biocatch se debe usar el API y el microservicio creado por Accenture que va hacia DataPower.

Se recomienda ofuscar los JS y las librerías para que no sean de fácil acceso para un defraudador.

A continuación, se exponen estándares planteados para los componentes de Voxel Community, estos componentes deben ser responsive.



Proyecto Quickstart:  <https://gitcorp.prod.cloud.ihf/Itau-latam-web/quickstart-responsive>

## Diagrama de contexto de Front

A continuación, se muestra el diagrama de contexto del FrontEnd de la aplicación

Ilustración 1 Diagrama General de Frontend

# Manual de MicroFrontend

Todo el código que se realizó para hacer esta guía queda en el siguiente repositorio de github:

<https://github.com/dayepesb/microfrontends-angular-single-spa>

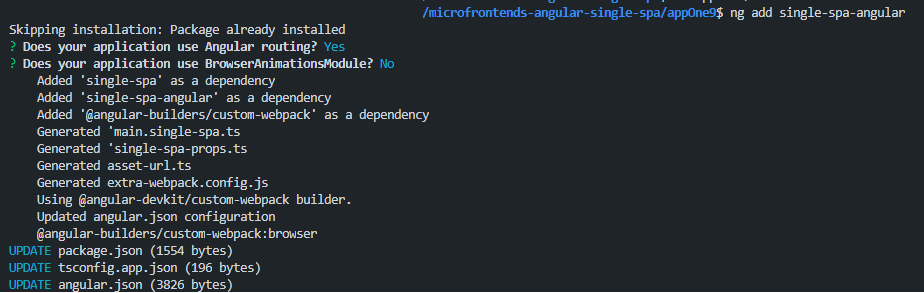
## Implementación de single spa en Angular

### *Instalación de dependencias en proyectos hijos*

Primero que nada, ya debemos tener una aplicación angular creada sobre cualquier versión de angular.

Mínimo se debe tener 2 proyectos, 1 proyecto que va a funcionar como chasis, que es el que llama e importa los microfrontend y los embebe en la app (padre), y otro que va a hacer el hijo o el microfrontend que va a tener el contenido principal de la aplicación.

Después de crear el proyecto hijo, agregamos la dependencia de single-spa-angular al proyecto, este lo añadimos con angular directamente, para ello ejecutamos el siguiente comando:



Este comando crea varios archivos, y añade una dependencia de custom-webpack, para más información sobre el contenido de cada archivo ingresar a la siguiente url:

<https://single-spa.js.org/docs/ecosystem-angular/>

Si quisiéramos probar la aplicación en local, solo deberíamos correr el siguiente comando en nuestro proyecto hijo:



La variable **–deploy-url** sirve para especificarle al proyecto donde buscar los assets de ese microfrontend, supongamos que en ese microfrontend tenemos una imagen en nuestro componente, esta imagen cuando lo vayamos a embeber en el chasis es necesario buscarla en la url donde esta desplegado el microfrontend, si no colocaremos esta variable, intentaría buscar el recurso en el servidor donde está desplegado en chasis, por ejemplo:

Tenemos el chasis desplegado en la url <http://localhost:4200/> y nuestro proyecto hijo en la url <http://localhost:4201/>, y en nuestro proyecto hijo tiene una imagen en el componente html de la aplicación, cuando estemos mirando nuestro proyecto de chasis que tiene embebido la aplicación hijo, esta imagen la intentara buscar en la url donde esta desplegado el chasis, a menos que se le especifique la variable **–deploy-url** y se haga la implementación para que busque el recurso en una url en específico, más adelante se especifica la implementación de esta variable.

**NOTA**: en ocasiones al momento de tratar de subir el proyecto hijo y desplegarlo, puede arrojar un error como el siguiente:

[Capte la atención de los lectores mediante una cita importante extraída del documento o utilice este espacio para resaltar un punto clave. Para colocar el cuadro de texto en cualquier lugar de la página, solo tiene que arrastrarlo.]

Texto

Descripción generada automáticamente

Esto es debido a que single-spa-angular agrega una dependencia al proyecto “@angular-builders/custom-webpack” esto funciona para construir la aplicación y realiza configuraciones en el archivo **angular.json** del proyecto hijo, para cambiar la manera como se construye y se compila la aplicación sin embargo, hay que cambiar la dependencia de “@angular-devkit/build-angular“ a una versión en específica, y en algunos casos single-spa no hace esto, para ellos nos dirigimos a nuestro package.json de nuestro proyecto hijo, ubicamos la sección de “devDependencies” y buscarnos la dependencia con el nombre “@angular-devkit/build-angular“ y cambiamos a la versión como se muestra a continuación:



Y ejecutamos **npm install** a nuestro proyecto hijo y listo

**NOTA**: después de implementar la librería de single-spa a su proyecto hijo se sugiere, borrar la carpeta node\_modules y el archivo package.json, esto debido a que se detectaron problemas en las dependencias y una solución que se encontró borrar este directorio y este archivo, y después volver a ejecutar el **npm install**.

Además de estás configuraciones se debe tener en cuenta que el selector en el componente de AppComponent de angular, hay que cambiar el nombre, en cualquier aplicación Angular sin el framework de single-spa, el nombre del selector por defecto es “**app-root**”, cuando se tienen varias aplicaciones angular para convivir como microfrontend este nombre se debe cambiar y debe ser diferente en todos los proyectos, además que este nombre por defecto también se encuentra en el archivo **main.single-spa.tss** y en el **index.html**

Captura de pantalla de un teléfono celular

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

* 1. ***Instalación de dependencias en proyecto chasis***

Instalación de dependencias en proyecto padre

Ahora vamos a trabajar en el componente padre, en el chasis que contiene todas las aplicaciones.

Primero que nada, vamos a instalar systemjs, esta librería nos va a ayudar a configurar todas las URLs que vamos a necesitar para importar los microfrontend en el chasis:



También instalamos la librería de single-spa y single-spa-angular, se sugiere instalar específicamente las versiones como se muestra a continuación (Abril-2021):

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Además, nos vamos al archivo tsconfig.app. json y modificamos la propiedad **compilerOptions** como se muestra a continuación:

Texto

Descripción generada automáticamente

También se debe tener en cuenta los siguientes cambios en el archivo **angular. json**.

Texto

Descripción generada automáticamente

### Configuración de Chasis

Ya implementado systemjs en nuestro proyecto de chasis ahora debemos crear en la carpeta de assets del proyecto, un archivo. json que va a contener todas las URLs de los hijos que queremos embeber en la aplicación de chasis, con la siguiente estructura:

Texto

Descripción generada automáticamente

En este archivo se guarda bajo unos seudónimos cada microfrontend que quisiéramos añadir a nuestro proyecto.

Ahora necesitamos importar este json en el proyecto para ello nos vamos al archivo index.html de nuestro proyecto de angular de chasis y agregamos la siguiente etiqueta:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**NOTA**: La etiqueta “import-map-overrides-full” es solo para verificar que lo que estemos importando sea correctamente y en ningún caso esta etiqueta debería estar en una versión productiva.

En el archivo import-map. json que agregamos anteriormente se especifica la URL de donde van a estar publicado cada microfrontend, la manera como se embebe la aplicación en el chasis es que el chasis hace el llamado a estas URLs, haciendo referencia a un archivo main.js, este archivo contiene toda la aplicación de angular del microfrontend, este es el archivo que el chasis llama y embebe, en cualquier caso, siempre toca especificar en la URL el acceso a este archivo **main.js.**

Ahora ya que hemos realizado todas las configuraciones en el chasis ahora debemos importar el microfrontend en el chasis donde se requiera, hay dos maneras especificas con lo que se pueden hacer:

### Importándolo en el componente de angular (Recomendado)

Para esto nos vamos directamente al componente donde queremos importar el microfrontend, en este caso lo vamos a hacer en main. component del proyecto de chasis.

Ya como tenemos todos los recursos instalados, vamos a utilizar la librería de single-spa y systemjs para agregar la aplicación del header.

Para ello nos dirigimos al componente, y en el constructor del componente registramos una de nuestras aplicaciones que colocamos en el import-map. json, en este caso el header, y lo registramos de la siguiente manera:

Texto

Descripción generada automáticamente

Después de esto nos dirigimos al HTML del componente y agregamos la etiqueta:

<div id="single-spa-application:<<NAME\_APPLICATION\_IMPORT\_MAP>>"></div>

Y reemplazamos “**<<NAME\_APPLICATION\_IMPORT\_MAP>>**” por el nombre de nuestro microfrontend registrado en el constructor de nuestro componente.



Como se puede observar el componente header se importó correctamente en nuestro proyecto.

Rectángulo

Descripción generada automáticamente

Y en network vemos lo siguiente:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Aclaremos: como registramos la aplicación, nosotros vemos 3 propiedades para registrar el microfrontend:

1. **name**: es el nombre sobre el cual se va a registrar la app, se sugiere utilizar el mismo que en el import-map. json, pero no es necesario, solo se debe tener en cuenta que es el mismo que se utiliza en la etiqueta html en donde se coloca el microfrontend.
2. **app**: Esta es una función que retorna la función de SystemJS import, esta función se encarga de traer el archivo main.js de nuestra aplicación.
3. **activeWhen**: Es la URL desde cuando está activa y cuando se debe mostrar el microfrontend, en este caso como era el header necesitamos que se muestre desde que aparezca la ruta ‘/’

### Importándolo en el componente de angular

Primero que nada debemos generar un folder en dónde vamos a guardar archivos de configuración que veremos más adelante.

Vamos a crear el archivo **spa-unmount. guard.ts** en el folder creado anteriormente, este archivo permite a single-spa montar y desmontar los microfrontend sin ningún problema.

Texto

Descripción generada automáticamente

Después creamos el archivo **spa-host. component.ts**, este archivo nos ayuda a evitar impactos negativos mientras cambia de un microfrontend a otro y que no arroje ningún error mientras hace el cambio de aplicación.

**Texto

Descripción generada automáticamente**

El ultimo archivo que generaremos es **spa-host-module.ts** este módulo nos ayuda a gestionar los dos archivos anteriores.

Texto

Descripción generada automáticamente

Ahora vamos al **app-routing. module**, acá vamos a importar directamente el microfrontend dependiendo de la URL del routing de angular, todo esto se hace con la ayuda de **spa-host.module** que creamos anteriormente, la forma como se hace es de la siguiente manera.

Texto

Descripción generada automáticamente

El spa-host.module nos ayuda a importar la app y a montarla en el chasis, el nombre de la aplicación que se le pasa en data, en este caso se llama **child1**.

Así es como podremos implementar la arquitectura de microfrontend en Angular, con cualquier versión, esta librería de single-spa-angular soporta varias versiones de angular y no presentan ningún conflicto.

# Comunicación Microfrontends

Para la comunicación entre microfrontend vamos a utilizar la librería opensource eventing-bus, que se encuentra en el siguiente repositorio:

<https://github.com/arkency/event-bus>

Esta librería nos permite realizar la comunicación entre microfrontend.

Primero que nada, creamos un servicio como el siguiente:

Texto

Descripción generada automáticamente

Lo agregamos en la carpeta lib

Hay dos maneras de inyectarlo en los microfrontends:

1. Inyectarlo en el componente (Recomendado)



Se inyecta el servicio en el componente y se registra en las propiedades del microfrontend:

Texto

Descripción generada automáticamente

1. Inyectarlo en el appRouting

Se crea la inyección en el routing



Texto

Descripción generada automáticamente

Y se envía en el registro del módulo parcel que registra la aplicación

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Se debe ir al spa-host.module, recibir el parámetro que llega y pasárselo al método del componente que lo recibe:

Texto

Descripción generada automáticamente

Ahora se debe modificar el componente que recibe está propiedad:

Texto

Descripción generada automáticamente

Para compartir información primero se crea el evento en dónde se recibe la información, para este caso en el microfrontend 2:

Texto

Descripción generada automáticamente

Y para emitir el evento se emite desde el microfrontend 1:

Texto

Descripción generada automáticamente

# Repositorio guía

Todo el proyecto de ejemplo está en este repositorio, puede utilizarlo como guía, en cualquier caso.

* <https://github.com/dayepesb/microfrontends-angular-single-spa>

# Documentación y links

* <https://ngrx.io/generated/images/guide/store/state-management-lifecycle.png>
* <https://github.com/matt-gold/single-spa-angular-cli>
* <https://single-spa.js.org/docs/examples/>
* <https://github.com/joeldenning/coexisting-angular-microfrontends>
* <https://single-spa.js.org/docs/building-applications/>
* <https://github.com/systemjs/systemjs/blob/master/docs/import-maps.md>
* <https://www.gitmemory.com/issue/systemjs/systemjs/1952/500203133>
* <https://single-spa.js.org/docs/ecosystem-angular/>
* <https://academia-binaria.com/flujo-reactivo-unidireccional-con-Angular-y-RxJs/>
* <https://ngrx.io/guide/store>
* <https://ngrx.io/guide/store-devtools>
* <https://ngrx.io/guide/store-devtools>
* <https://ngrx.io/api/router-store/StoreRouterConnectingModule>
* <https://ngrx.io/guide/router-store>
* <https://ngrx.io/api/router-store/RouterState#Minimal>
* <https://ngrx.io/guide/schematics>
* <https://ngrx.io/api/data/SetEntityCache>
* <https://ngrx.io/guide/store/reducers>
* <https://ngrx.io/guide/store/reducers>
* <https://ngrx.io/api/data/EntityCache>
* <https://stackblitz.com/angular/ggkagbvmrgx?file=src%2Fapp%2Fmy-counter%2Fmy-counter.component.ts>
* <https://rxjs-dev.firebaseapp.com/guide/subscription>
* <https://nils-mehlhorn.de/posts/ngrx-keep-state-refresh>