ARQUITECTURAS DE MICRO FRONTENDS, COMPONENTES WEB Y LIBRERÍAS DE COMPONENTES

Leon Dario Arango Amaya

Informe de práctica para optar al título de Ingeniero de Sistemas

Tutores

Deisy Loaiza Berrío, Ingeniera de Sistemas

Jonathan Alexander Diosa Giraldo, Magíster (MSc) en Ingeniería

Frank Alexis Castrillon Giraldo, Ingeniero de Sistemas

Universidad de Antioquia

Facultad de Ingeniería

Ingeniería de Sistemas

Medellín, Antioquia, Colombia

2021

Cita	(Arango Amaya, 2021)
Referencia Estilo APA (2020)	Arango Amaya, L.D. (2021). *ARQUITECTURAS DE MICRO FRONTENDS, COMPONENTES WEB Y LIBRERÍAS DE COMPONENTES* [Trabajo de grado profesional]. Universidad de Antioquia, Medellín, Colombia..

Universidad de Antioquia - www.udea.edu.co
Rector: John Jairo Arboleda Céspedes.
Decano/Director: Jesús Francisco Vargas Bonilla.
Jefe departamento: Diego José Luis Botia Valderram.

Tabla de contenido

Introducción

La web ya no es igual… En una carrera a gran velocidad por llevar todo lo que la humanidad conoce a este mundo abstracto que llaman internet, la convirtieron en la web moderna, un mundo donde http no era suficiente y ahora se introdujo un concepto de http/3, donde ya no solo existen páginas web, sino aplicaciones web, donde estas comenzaron a crecer inmensurablemente y ya no era suficiente en el lado del servidor el monolito n capas on-premise de toda la vida; este también logró adaptarse en diferentes soluciones según las necesidades, como una arquitectura en la nube basada en microservicios, la cual puede traer beneficios como flexibilidad, escalabilidad, versatilidad, modularidad e incluso mejor rendimiento. ¿Pero qué pasa en el lado del cliente? ¿Cómo puede el cliente obtener beneficios? Es aquí donde entran las arquitecturas de microfrontends, componentes web y librerías de componentes, alternativas en el lado del cliente que buscan mejorar la forma en que se construye el frontend. Este proyecto consiste en explorar cómo funcionan cada uno de estos, en qué casos se deben utilizar, arquitecturas adecuadas, patrones, antipatrones y prácticas para su diseño e implementación, pruebas y finalmente como es un pipeline de integración y despliegue continuo en un microfrontend; Que finalmente serán recopilados a modo de guía, para facilitar en un futuro el trabajo de otras personas de la compañía o la comunidad en general que quieran abordar este tema y aplicarlo en algún proyecto.

Objetivo General

Elaborar una guía que permita construir arquitecturas basadas en micro frontends, componentes y librerías web, destacando los aspectos más relevantes en cada uno de estos.

Objetivo General

- Estudiar cómo funcionan, se usan e implementan las arquitecturas basadas en micro frontends, componentes y librerías web.
- Definir criterios que permitan identificar características de una arquitectura adecuada en el frontend.
- Hacer uso de los criterios, para definir arquitecturas adecuadas, patrones, antipatrones y prácticas para el diseño e implementación.
- Implementar una prueba de concepto ejecutando los pasos y prácticas definidas en la guía.
- Realizar pruebas de software sobre la implementación realizada.
- Configurar un pipeline de integración y despliegue continuo para la implementación, incluyendo las pruebas de software.
- Concluir el trabajo anexando a la guía las lecciones aprendidas en la implementación de la prueba de concepto, pruebas de software y pipeline de ci/cd

Teoría

Los microfrontends, un término que apareció a por primera vez en el radar de ThoughtWorks a mediados del 2016 y que se introdujo como una tecnología en adopción durante el 2019; la cual pretende convertir la interfaz monolítica que usualmente se ofrece al cliente en algo similar a lo que se hace en el servidor con los microservicios. Sin embargo, para lograr esto es importante entender algunos conceptos y establecer reglas o principios que sirvan como un estándar para facilitar el desarrollo de los mismos (Geers, 2019).

¿Qué es un microfrontend?

Casi siempre que alguien habla de microfrontends se dice que es adaptar los microservicios del lado del cliente y es cierto, sin embargo, una definición puede ser la que ofrece Cam Jackson en el sitio de Martin Fowler que dice:

> "An architectural style where independently deliverable frontend applications are composed into a greater whole" (Jackson, 2019)

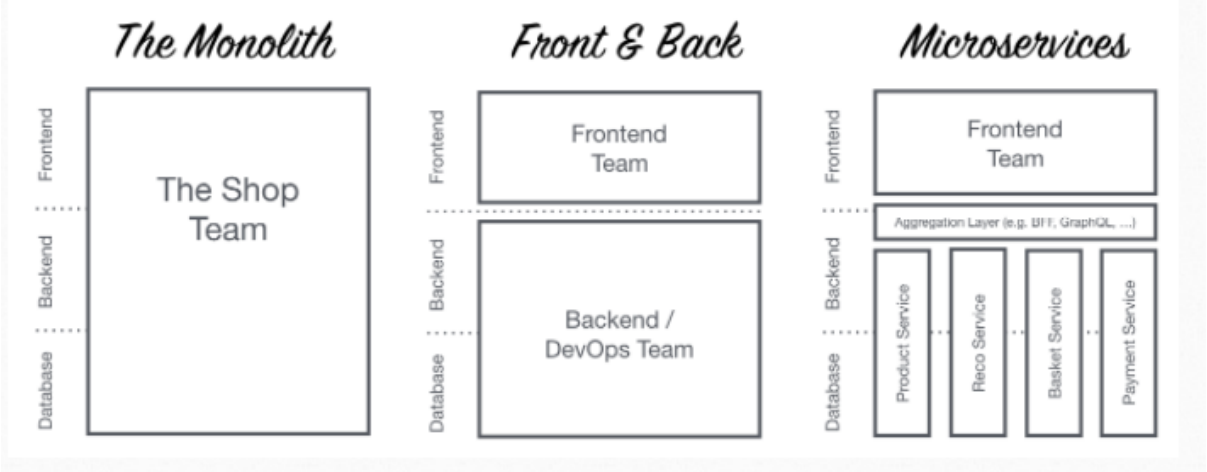
Pero más allá de esto, en los microfrontends lo que realmente se busca es entender cómo está estructurado un sitio web y cómo se relacionan sus partes, así que estos no son más que un estilo arquitectónico que pretende independizar en la medida de lo posible componentes, funcionalidades o características dentro de un sitio web.

¿Cuál es su importancia?

Para entender un poco más de la importancia de los microfrontends, es necesario contextualizar lo que sucede hoy en día.

Ya se ha dicho que la web cambió y que ahora es moderna, pero esto no dice mucho, así que tomaremos el siguiente esquema donde nos presentan tres arquitecturas que ahora son bien conocidas en el desarrollo de aplicaciones web. (Figura 1)

Figura 1
Arquitecturas conocidas en aplicaciones web

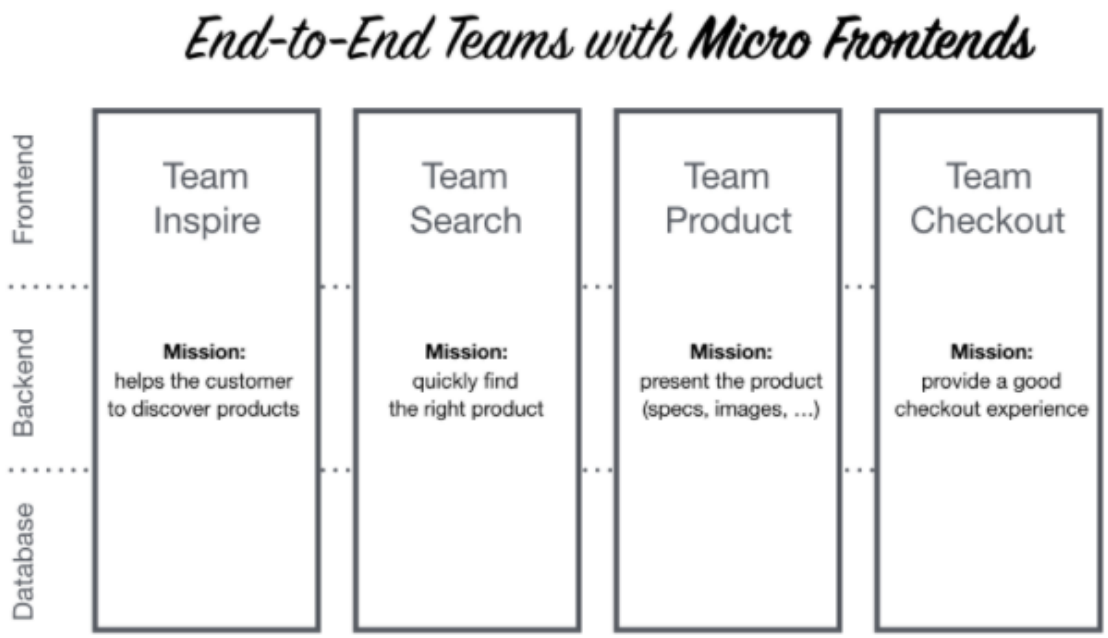


En la primera parte vemos un monolito, un tipo de arquitectura donde todo el sistema es uno solo, el frontend está atado al backend, en la cual existen problemas cuando se piensa en la escalabilidad de este, la cantidad de conflictos que se pueden presentar cuando se tienen grandes equipos de trabajo o incluso lo complejo que puede llegar a ser un despliegue, ya que requiere habilitar una ventana de indisponibilidad y tumbar todo el sistema durante un tiempo para integrar nuevos desarrollos en el sistema. Es cierto que hay tecnologías que pueden ayudar a resolver este tipo de problemas, pero no siempre es así.

Luego se presenta una arquitectura también muy conocida como cliente-servidor, la cual posee el backend y el frontend por separado, los cuales se suelen conectarse a través de APIs, que pueden ser arquitecturas como representational state transfer (REST) y Simple Object Access Protocol (SOAP), o tecnologías como GraphQL; Esta alternativa suele ser mucho mejor que tener un monolito. No obstante, aquí se siguen teniendo problemas para escalar, dividir mejor los equipos de acuerdo a funcionalidades, soportar alta concurrencia e incluso la comunicación entre backend y frontend. Por último tenemos una arquitectura basada en microservicios, donde el backend es un compuesto de pequeños sistemas cada uno con una responsabilidad en el dominio, los cuales se pueden escalar fácilmente integrando algunas tecnologías y también es posible evitar muchos conflictos si se tienen desarrolladores dedicados a algunos de estos. Pero es posible observar que a pesar de todos los cambios arquitectónicos realizados, el frontend sigue siendo el mismo, con los mismos problemas y con pocas posibilidades de aprovechar todos los beneficios de estas arquitecturas. Y por esto nace la idea de los microfrontends, un estilo arquitectónico que busca aprovechar los beneficios de principio a fin.

Figura 2

Arquitectura de microfrontends en aplicaciones web



En este esquema (Figura 2) es posible observar un modelo de trabajo aprovechando las arquitecturas de microfrontends, donde cada equipo posee una responsabilidad sobre el negocio y es quien se encarga de todo el flujo y las decisiones de principio a fin. Este modelo solo es posible si logramos dividir al igual que el backend en microservicios, el frontend en microfrontends, permitiendo a los equipos empoderarse completamente de una parte del negocio y que sean estos quienes tomen las decisiones sobre tecnologías a utilizar, metodología de trabajo, tiempos de entrega y mucho más. (Verma, 2020)

De hecho la web moderna trae consigo una cantidad enorme de tecnologías, buenas prácticas, patrones, estándares y otras cosas que aportan mucho valor a todas las aplicaciones; Se ha hecho bastante común hacer uso de los Single Page Application (SPA) en la web, con tecnologías como React, Angular, Vue, Ember o Svelte que siempre generan discusiones de cual es mejor, cual es framework, cual libreria y muchas veces no se toma una decisión fácilmente porque pueden darse casos en los que sea algo más de opiniones personales; Como se dijo antes los microfrontends permiten a los equipos tomar decisiones y las tecnologías, prácticas o estándares de uno microfrontend pueden llegar a ser totalmente diferentes de los demás. Sin embargo, es importante tener muy claro cuales son las reglas de juego a la hora de tratar con microfrontends.

¿Cuáles son las reglas del juego?

Es importante conocer tanto las bases como los límites en todo esto, ya que permite mantener un orden, por ello resaltamos varios principios o reglas que ya se han definido durante estos años para la construcción de microfrontends:

1. La tecnología debe ser agnóstica.
2. Cada equipo debe tener su código aislado.
3. Es importante diferenciar los equipos y su trabajo para evitar conflictos (prefijos).
4. Priorizar funciones nativas del navegador sobre funciones personalizadas.
5. En general el sitio debe ser resiliente, así que si algún microfontend falla, los demás deben permanecer estables.

Ventajas

Algunos de los beneficios que brindan puntualmente los microfrontends son:

- Repositorios con código más cohesivo, fácil de mantener y en menores cantidades (spaghettiless).
- Equipos empoderados, desacoplados y escalables.
- Cambios en partes específicas de la interfaz de forma más rápida, sin afectar a las demás.

Desventajas

Siempre lo he dicho, todo en el universo tiene un precio y en el caso de los microfrontends no hay una excepción, algunas de las desventajas que hay en este estilo arquitectónico son:

- Grandes retos para eliminar dependencias o comunicar los microfrontends.
- Duplicación de dependencias.
- Mayor cantidad de archivos que el cliente requiere descargar.
- Añade complejidad al ciclo de integración y despliegue continuo
- Requiere que el equipo maneje bien el tema.

En términos muy generales esto son los microfrontends, se expuso lo bueno y lo malo, pero ¿qué sigue? Pues es momento de entrar más a fondo en la parte técnica. Si alguna vez has trabajado con nuevas tecnologías para SPA, entonces conocerás algo sobre web components, los cuales son la base de todo esto.

Web Components

Los web componentes no son más que un conjunto de tecnologías que permiten crear nuevos elementos en HTML, encapsulados y reutilizables para cualquier página o aplicación web (Web Components, 2020). Una buena práctica en el desarrollo siempre será reutilizar

código, tanto como sea posible, siempre y cuando no genere dependencias que se interpongan al momento de escalar el sistema. Y los web components ayudan justamente en ello. Este conjunto de tecnologías se puede resumir en tres partes: Custom elements, Shadow DOM y HTML templates.

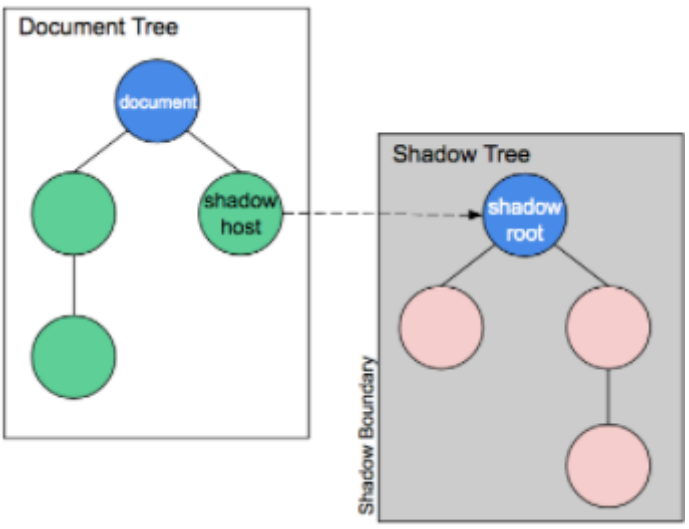
Custom elements: Son las bases para diseñar, crear y utilizar nuevos elementos en el DOM. Estos son los que se encargan de encapsular la funcionalidad del componente. Además son de gran importancia, ya que al ser la parte funcional, son los que tienen un ciclo de vida.

Ciclo de vida: El ciclo de vida se define a partir de callbacks, los cuales se ejecutan en diferentes momentos en la vida de un elemento, como lo puede ser (Discuss & share web components, 2016):

- connectedCallback en el momento en que se añade al DOM.
- attributeChangeCallback cuando se detecta un cambio en el elemento.
- adoptedCallback si el elemento es incrustado en algún lugar, este es raro ya que se suele dar más que todo en iframes.
- disconnectedCallback justo cuando el elemento es removido del DOM.

Shadow DOM: Este es una parte importante, ya que ayuda a encapsular características en los elementos generando una especie de árbol DOM como sombra (Figura 3). De esta forma, se evitan los conflictos en los estilos o scripts en el documento. (Using shadow DOM, 2019)

Figura 3
Shadow Root de un elemento



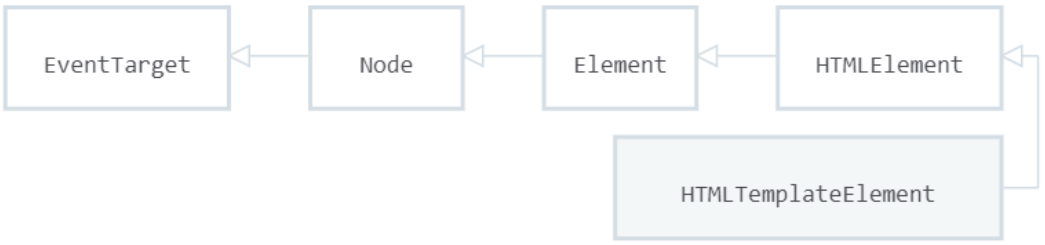
El Shadow DOM posee cuatro conceptos fundamentales:

- Shadow host: El nodo regular del DOM al que es atado el shadow DOM.
- Shadow tree: El árbol DOM dentro del shadow DOM.
- Shadow boundary: El punto en el que el shadow DOM termina y el DOM regular comienza.
- Shadow root: El nodo raíz del árbol Shadow.

HTML templates: Estos permiten escribir elementos que serán renderizados en la página más adelante en tiempo de ejecución. Estos se pueden reutilizar tantas veces como sea necesario y servirán como base para crear los custom elements. (Figura 4)

Figura 4

Composición de un Template en HTML



Librerías de componentes

Generalmente todos los proyectos cuentan los mismos problemas a la hora de crear interfaces, debido a la cantidad de elementos en común y que en muchos casos terminan duplicando código y adaptando los componentes según sea la necesidad. Sin embargo con todo esto de los web components, los desarrolladores comenzaron a crear librerías de componentes que permitan agrupar no sólo web components, sino también funciones para transformar elementos, contenidos o incluso acceder a datos. Estas librerías por lo general se suben en algún manejador de paquetes como npm y se añaden como dependencias en los proyectos cuando sea necesario. Usualmente estas librerías no poseen una arquitectura compleja, ya que solo buscan agrupar y exportar los componentes que poseen.

¿Cómo se pueden comunicar los microfrontends?

Para entender mejor la comunicación entre los microfrontends, primero es necesario conocer los medios, por los cuales se realizará esta, los cuales pueden ser Cookies, Local Storage o Session Storage.

Local Storage

Este hace parte del cliente y permite agregar pares de clave valor en el navegador, permite almacenar los datos sin fecha de vencimiento. Los datos persisten, incluso si se cierra el navegador.

Pros

- No tiene fecha de caducidad.
- Posee una capacidad de 5MB.
- Estos datos no se transfieren al servidor.

Contras

- Son en texto plano.
- Solo se puede leer del lado del cliente.
- Debe ser serializado.

Session Storage

Es otro tipo de almacenamiento en el cliente, bastante similar al localstorage, sin embargo posee diferencias significativas, la más importante es que estos datos persisten hasta que se cierra la sesión (La pestaña o el navegador.).

Pros

- Los datos están asociados a la sesión.
- Posee una capacidad de 5MB a 10MB.
- Estos datos no se transfieren al servidor.

Contras

- Son en texto plano.
- Solo se puede leer del lado del cliente.
- Debe ser serializado.
- Cada pestaña o ventana genera una sesión diferente.

Cookies

Estas llevan mucho más tiempo en el cliente que el local storage y el session storage, por lo tanto suele ser más común oír hablar de estas. No son más que otra forma de almacenar y transferir datos.

Pros

- Se pueden aprovechar para lectura del lado del servidor y el cliente.
- Poseen una capa de seguridad extra con httpOnly.

- La transferencia de archivos se realiza a través de XHR

Contras

- El tamaño debe ser inferior a 4KB
- Cada pestaña o ventana genera una sesión diferente.
- Se pueden configurar desde el navegador con document.cookie

Es importante aclarar que los pros y contras pueden ser así u opuestos según sea el caso, todo depende de la solución que se requiera, por ello es importante entender sus diferencias.

¿Cómo se migra un frontend monolito a microfrontends?

El propósito de todo esto no es migrarlos puntualmente, sin embargo es una pregunta que siempre se da, ya que hoy en día hay aplicaciones enormes como bancos, redes sociales, plataformas de streaming, las cuales sería un proceso largo y muy costoso rediseñarlas desde cero a microfrontends. Para ello podría dar únicamente algunos tips:

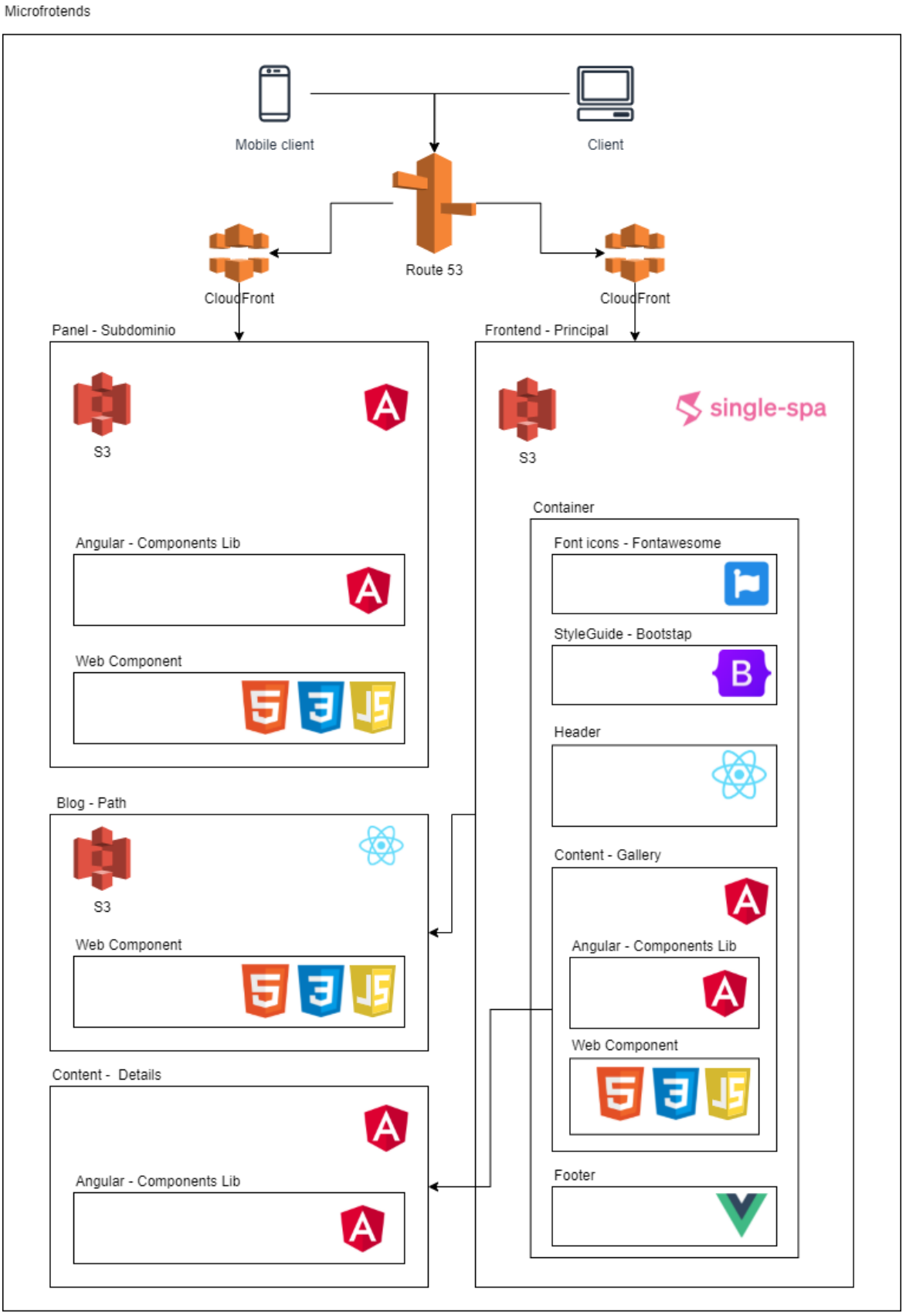
- Modularizar completamente el monolito, tratando de que no existan dependencias importantes entre ellos.
- Tener muy claro un sistema de diseño, ya que éste será compartido y es una parte fundamental en la identidad de la marca. Se puede utilizar alguna metodología con el fin de tener alta cohesión, como lo pueden ser:
    - Object-Oriented CSS (OOCSS)
    - Block, Element, Modifier (BEM)
    - Scalable and Modular Architecture for CSS (SMACSS)
- Extraer web components y librerías de componentes que se puedan reutilizar, ya que muchas veces en un monolito, esto se ve como un módulo aparte tipo” shared”.
- Cuando todo esté bien dividido dentro del monolito se puede comenzar con una implementación base y desacoplar lo más común que puede ser header, footer y el main que incluirá el contenido.

Implementación

Después de revisar la teoría y entender en términos generales estos conceptos de microfrontends, web components y librerías de componentes, se implementa una arquitectura basada en microfrontends, incluyendo en la medida de lo posible todos estos conceptos que se han expuesto.

Para ello, se crearán microfrontends utilizando múltiples formas como: iframes, subdominios, sitios por el path y el más robusto un framework para microfrontends llamado single spa. El objetivo principal es realizar todo un ciclo de desarrollo con pruebas y despliegue para una galería de autos que servirá como ejemplo. Esta incluirá tecnologías como web components, Angular, React, Vue y Svelte. A continuación se presenta un esquema del sitio (Figura 4).

Figura 4
Diagrama implementación


Microfrotends
Mobile client
Client
Route 53
CloudFront
CloudFront
Panel - Subdominio
S3
Angular - Components Lib
Web Component
Frontend - Principal
S3
single-spa
Container
Font icons - Fontawesome
StyleGuide - Bootstap
Header
Content - Gallery
Angular - Components Lib
Web Component
Footer
Blog - Path
S3
Web Component
Content - Details
Angular - Components Lib

Frontend Principal - Single SPA

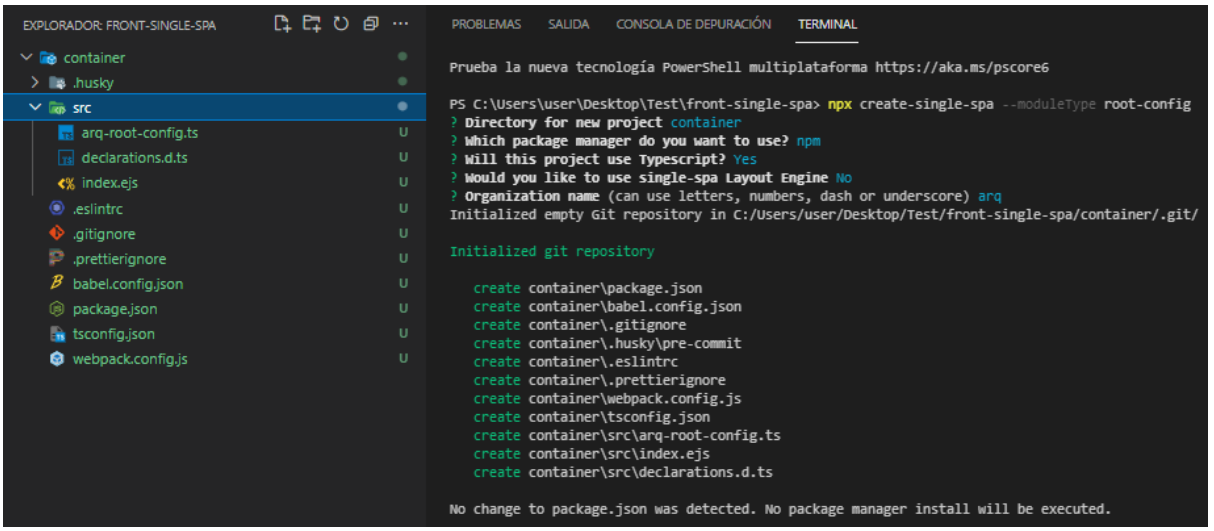
Este es el sitio principal, utiliza el framework single spa para gestionar los microfrontends, este tiene como base un root-config con un contenedor, en el cual se registran los microfrontends, es posible utilizar diferentes tecnologías para crear los microfrontends, cada uno será un registro independiente e incluso es posible realizar técnicas de lazy loading para la carga de microfronts.

Nota: En esta guía se escribirán los pasos más relevantes para crear desde cero una arquitectura de microfrontends, se omitirán temas de estilos y maquetado.

1. Se crea el root-config, el cual es un proyecto base para los microfrontends (Figura 5). Puedes encontrar mas información de la instalación de single spa en single.spa.js.org

Figura 5

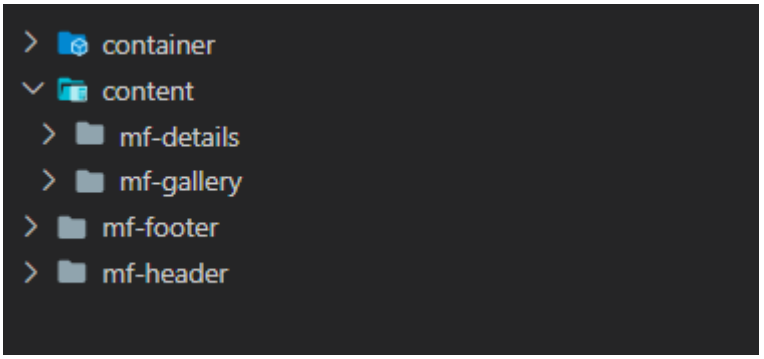
Creación del root-config



2. Se crea cada uno de los microfrontends (Figura 6), como se ha dicho anteriormente es posible elegir cualquier tecnología para ellos, en este caso se utilizará: React(Header), Angular(Contenidos) y Vue(Footer).

Figura 6

Creación de los microfrontends



3. Luego en cada microfrontend es necesario realizar una configuración, para que el root config, quien se encargará de orquestar estos, los pueda reconocer.
    a. En el caso de Angular se modifica el template en el archivo main.single-spa por el nombre que se le dará al app component, en este caso mf-gallery. Además se debe configurar el prefijo.

Figura 7

Configuración root de microfrontend tipo Angular

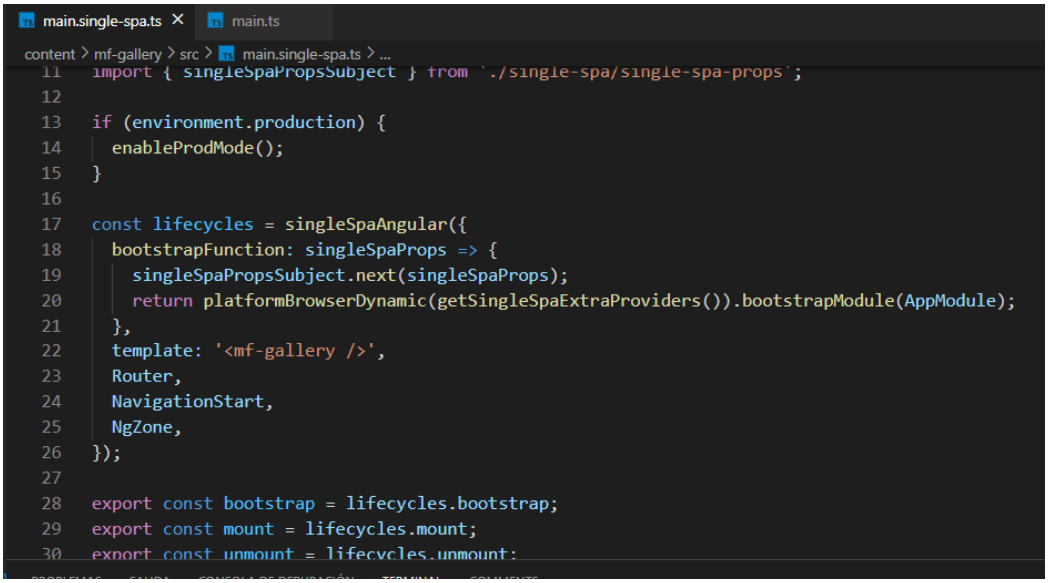


Figura 8

Configuración App component de microfrontend tipo Angular

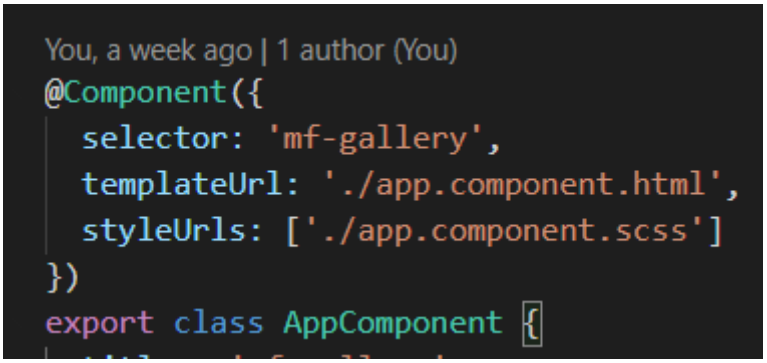
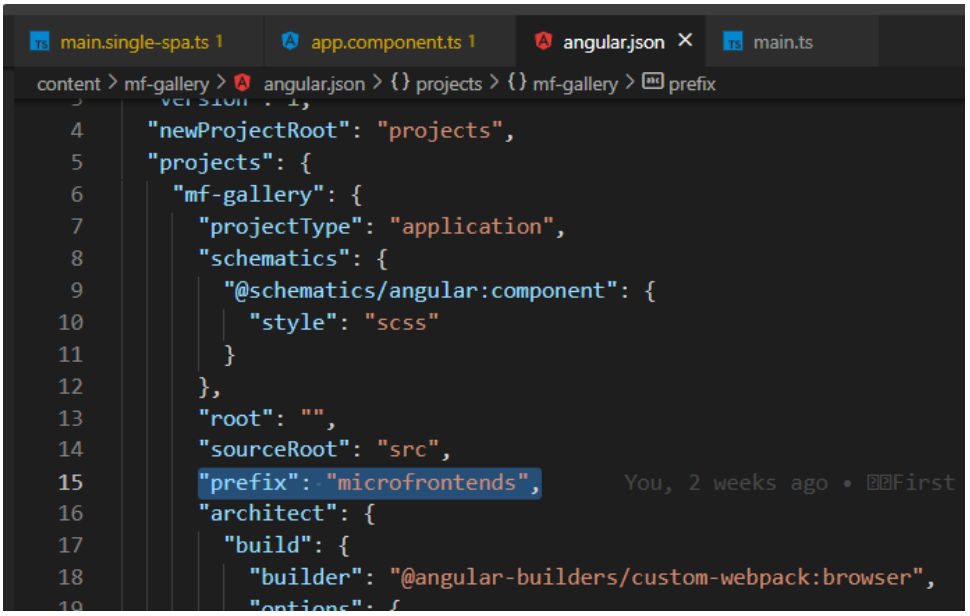


Figura 9

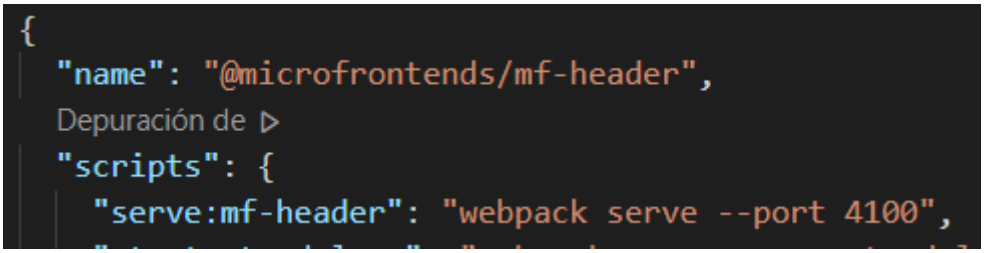
Configuración prefijo de microfrontend tipo Angular



b. En el caso de React solo es necesario asegurarse de que el nombre en el package.json se cree con el prefijo que utilizaremos.

Figura 10

Configuración prefijo de microfrontend tipo Angular



c. Para Vue es un poco más complejo, ya que se requieren varias configuraciones:
   i. Se debe copiar la configuración en el main.js que provee single spa para este caso.

Figura 11

Configuración prefijo de microfrontend tipo Vue

```
import { BootstrapVue, IconsPlugin } from 'bootstrap-vue'
import 'bootstrap/dist/css/bootstrap.css'
import 'bootstrap-vue/dist/bootstrap-vue.css'

// Make BootstrapVue available throughout your project
Vue.use(BootstrapVue)
// Optionally install the BootstrapVue icon components plugin
Vue.use(IconsPlugin)

Vue.config.productionTip = false;

const vueLifecycles = singleSpaVue({
  Vue,
  appOptions: {      You, 2 weeks ago • First commit - base v
    render(h) {
      return h(App);
    },
    router,
  },
});

export const bootstrap = vueLifecycles.bootstrap;
export const mount = vueLifecycles.mount;
export const unmount = vueLifecycles.unmount;
```

ii. Es importante cambiar la configuración para generar el bundle con nombre estáticos, así será posible reconocerlos desde el root config con el mismo nombre.

Figura 12

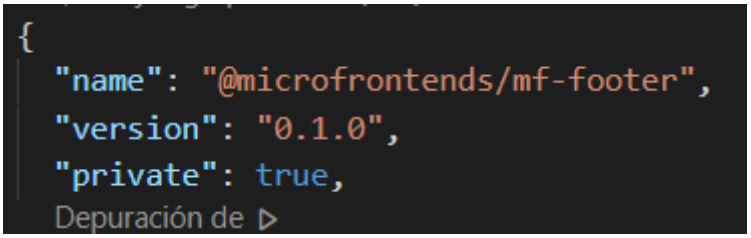
Configuración bundle estático de microfrontend tipo Vue

```
package.json mf-header | package.json mf-footer | vue.config.js
mf-footer > vue.config.js > <unknown> > configureWebpack > output > chu
You, 4 days ago | 1 author (You)
1  // vue.config.js
2  module.exports = {
3      configureWebpack: {
4          output: {
5              filename: '[name].js',
6              chunkFilename: '[name].[contenthash].js',
7          },
8      },
9  }
```

iii. Al igual que en react y en angular se debe manejar el prefijo, en este caso es necesario validarlo en el package.json.

Figura 13

Configuración bundle estático de microfrontend tipo Vue



4. Después de crear los microfrontends es importante asignarles puertos diferentes y registrar en el index.ejs del contenedor sus nombres y puertos para que pueda reconocerlos. Es importante agregar a este cambien las rutas del reac y react-dom si se utilizaran mf en React como es el caso.

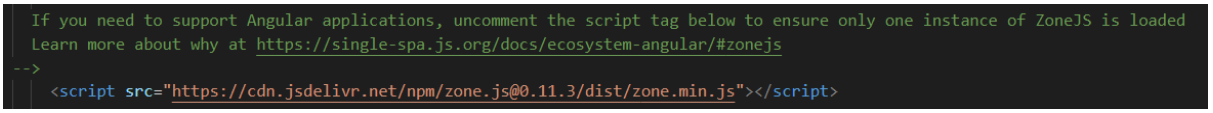
Figura 14

Configuración bundle estático de microfrontend tipo Vue

5. Para angular es necesario descomentar la línea que permite activar el zone.js en el index.ejs

Figura 15

Zone.js para angular



6. Después de correr los mf, es posible validar que cada ruta genere correctamente los archivos.

Figura 16

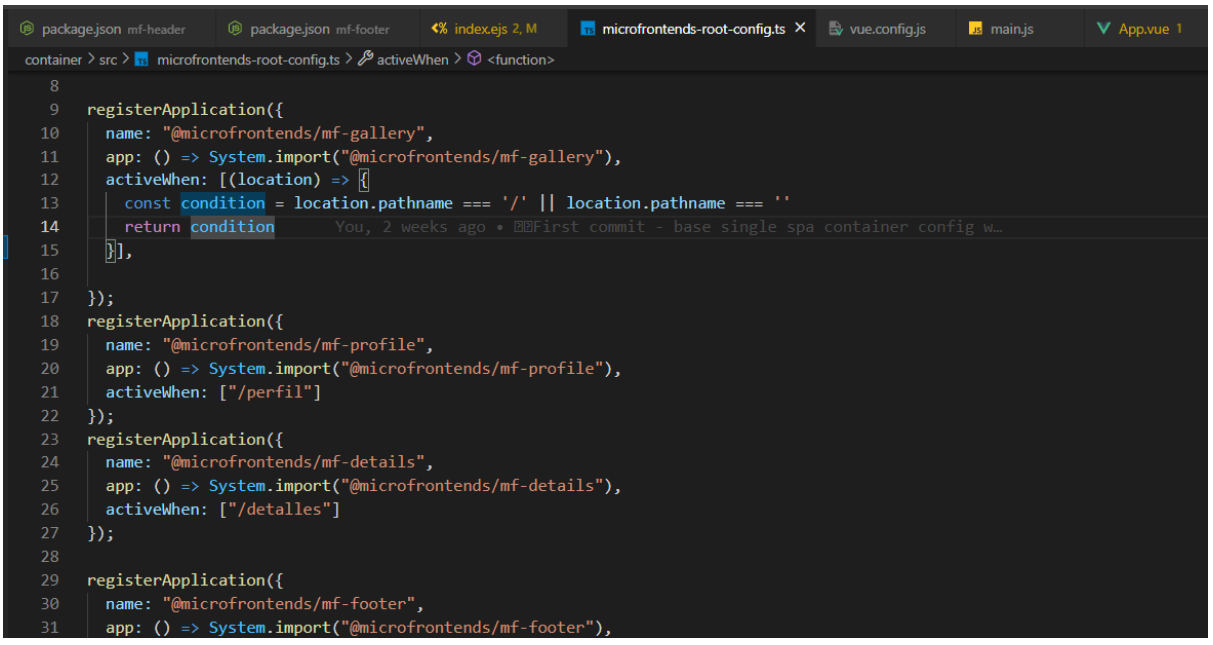
Archivo principal de un microfrontend

```
(function webpackUniversalModuleDefinition(root, factory) {
	if(typeof exports === 'object' && typeof module === 'object')
		module.exports = factory();
	else if(typeof define === 'function' && define.amd)
		define([], factory);
	else {
		var a = factory();
		for(var i in a) (typeof exports === 'object' ? exports : root)[i] = a[i];
	}
})((typeof self !== 'undefined' ? self : this), function() {
return /******/ (function(modules) { // webpackBootstrap
/******/ 	function hotDisposeChunk(chunkId) {
/******/ 		delete installedChunks[chunkId];
/******/ 	}
/******/ 	var parentHotUpdateCallback = (typeof self !== 'undefined' ? self : this)["webpackHotUpdate"];
/******/ 	(typeof self !== 'undefined' ? self : this)["webpackHotUpdate"] = // eslint-disable-next-line no-unused-vars
/******/ 	function webpackHotUpdateCallback(chunkId, moreModules) {
/******/ 		hotAddUpdateChunk(chunkId, moreModules);
/******/ 		if (parentHotUpdateCallback) parentHotUpdateCallback(chunkId, moreModules);
/******/ 	} ;
/******/
/******/ 	// eslint-disable-next-line no-unused-vars
```

7. Por último se deben configurar las rutas, donde se utilizaran los microfrontends. Esta se realiza en el archivo microfrontends-root-config.ts, el cual se encuentra en el contenedor.

Figura 17

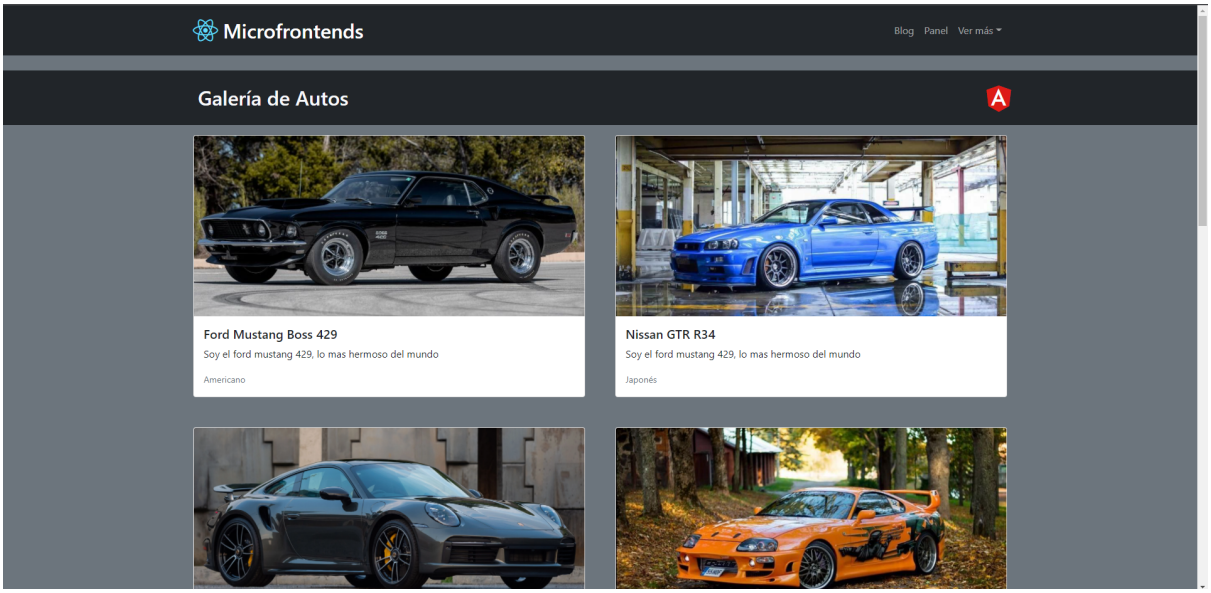
Rutas de los microfrontends



8. Luego de aplicar estilos y funcionalidades queda de la siguiente forma.

Figura 18

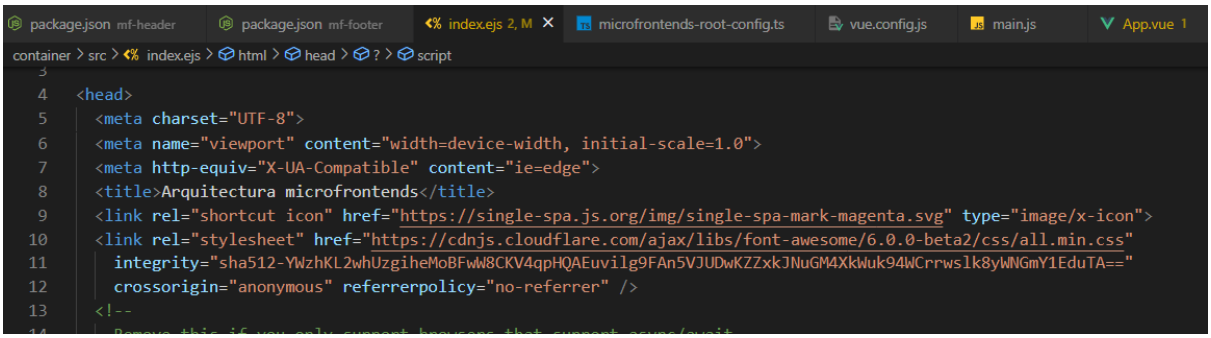
Microfrontends corriendo en local



9. Como adicional se puede incluir un sistema de diseño en un cdn, si este es compartido se puede integrar en el contenedor. En este caso se utiliza bootstrap como sistema de diseño, el cual se encuentra en el index.ejs como cdn.

Figura 19

Sistema de diseño global



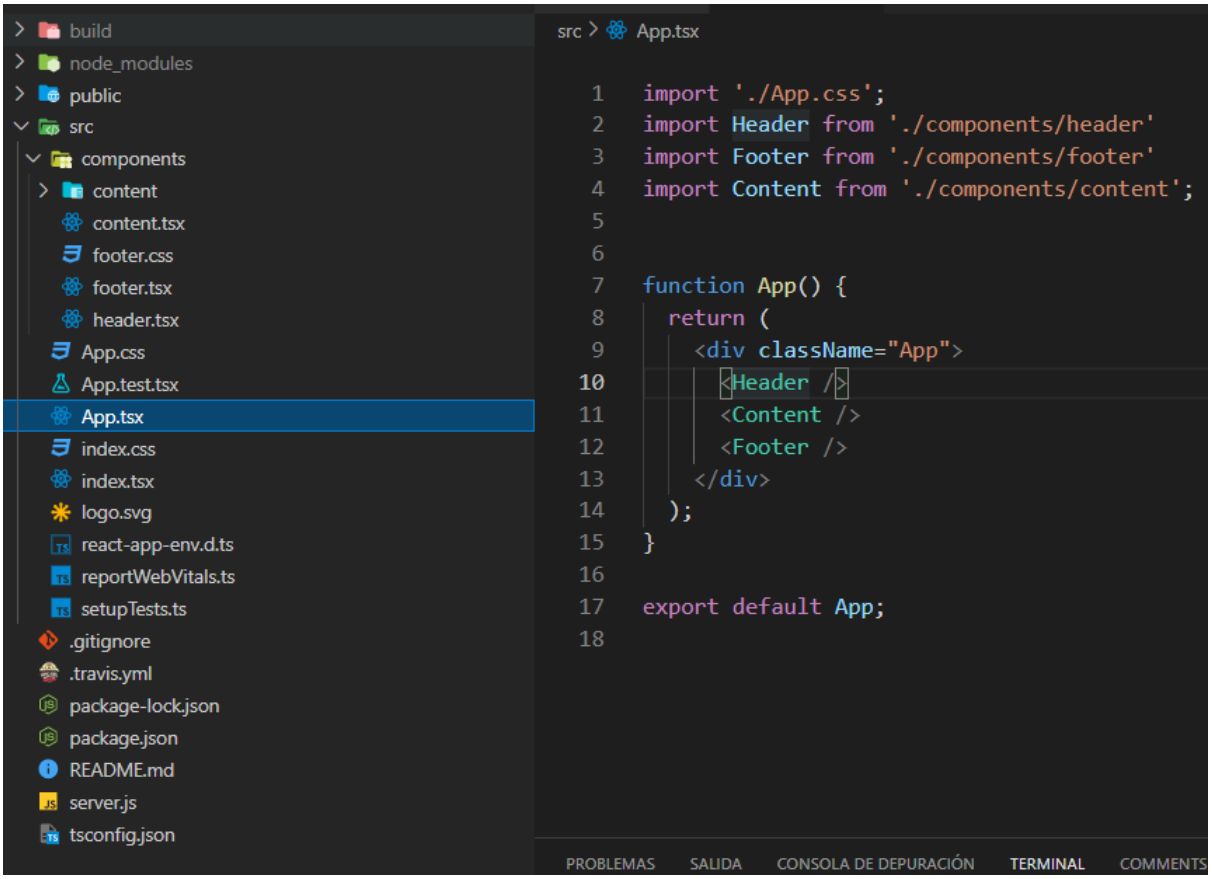
Frontend Blog - Path

El frontend como path es una alternativa para realizar microfrontends, ya que este puede comunicarse con los demás por medio de múltiples medios, como pueden serlo cookies, local storage o session storage.

Este tipo de microfronted no es más que un proyecto sencillo, el cual se integrará a los demás por medio de la infraestructura, en este caso se utilizara cloudfront, el CDN de AWS que también permite realizar estas integraciones a partir de comportamientos.

Figura 20

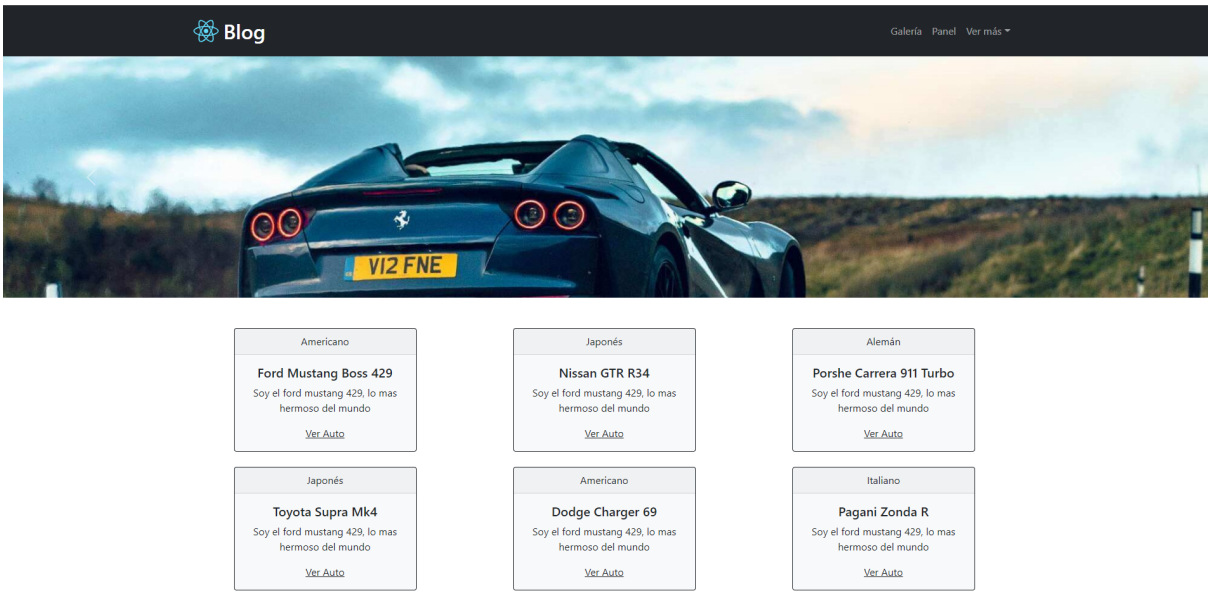
Arquitectura del blog



Este se puede construir como se requiera, no hay configuraciones adicionales que se deban tener en cuenta, como en el caso de single spa.

Figura 21

Blog



Frontend Panel - Subdominio

Otra alternativa son los subdominios, los cuales están mucho más aislados, ya que no comparten directamente los mismos medios como en el caso del path. Sin embargo, si es posible comunicar un subdominio con el dominio a través de algo llamado Cross-origin, del cual se hablará más adelante.

El panel será un proyecto en Angular sencillo, no requiere configuraciones adicionales para el despliegue.

Figura 22

Arquitectura del panel

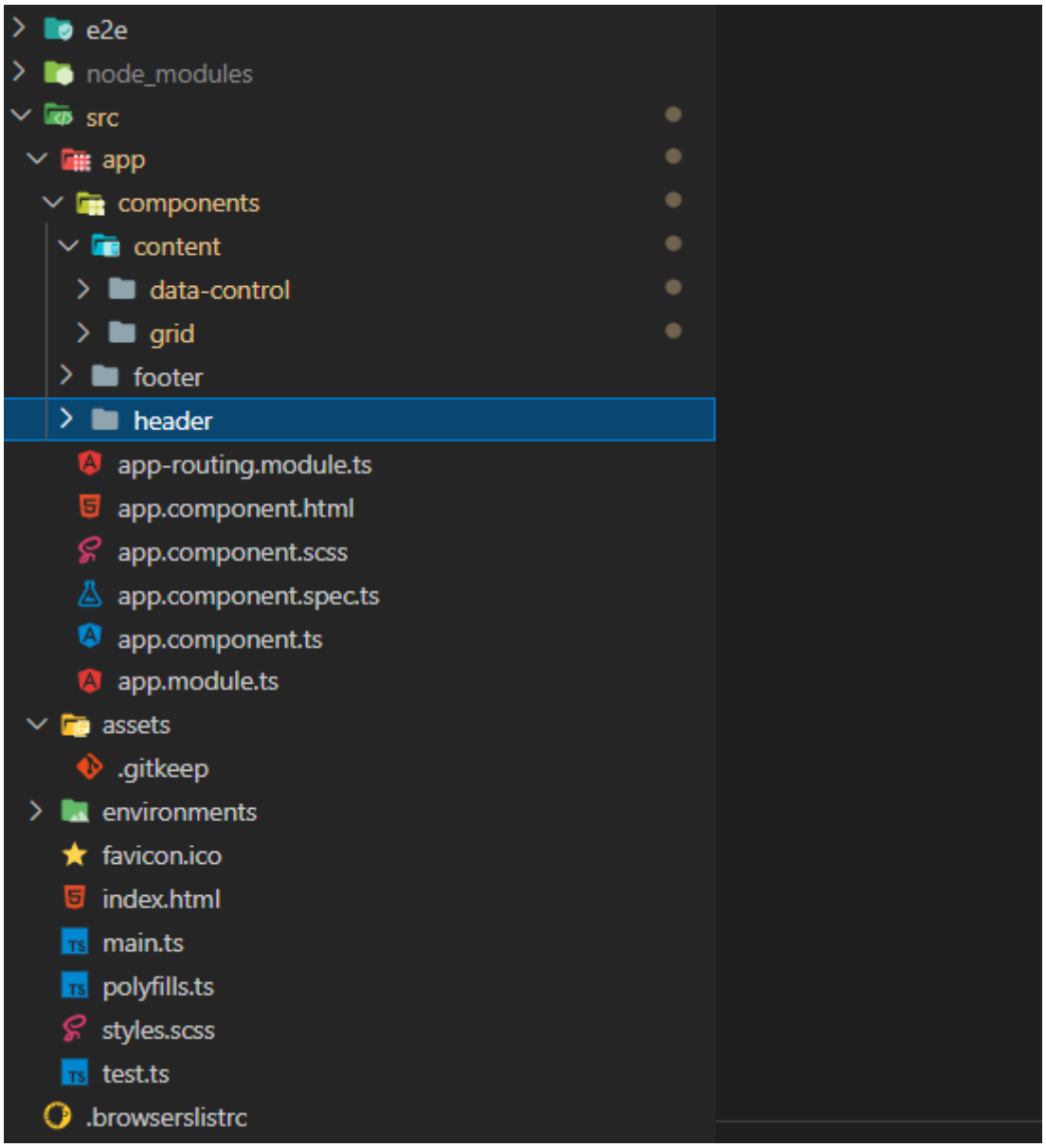
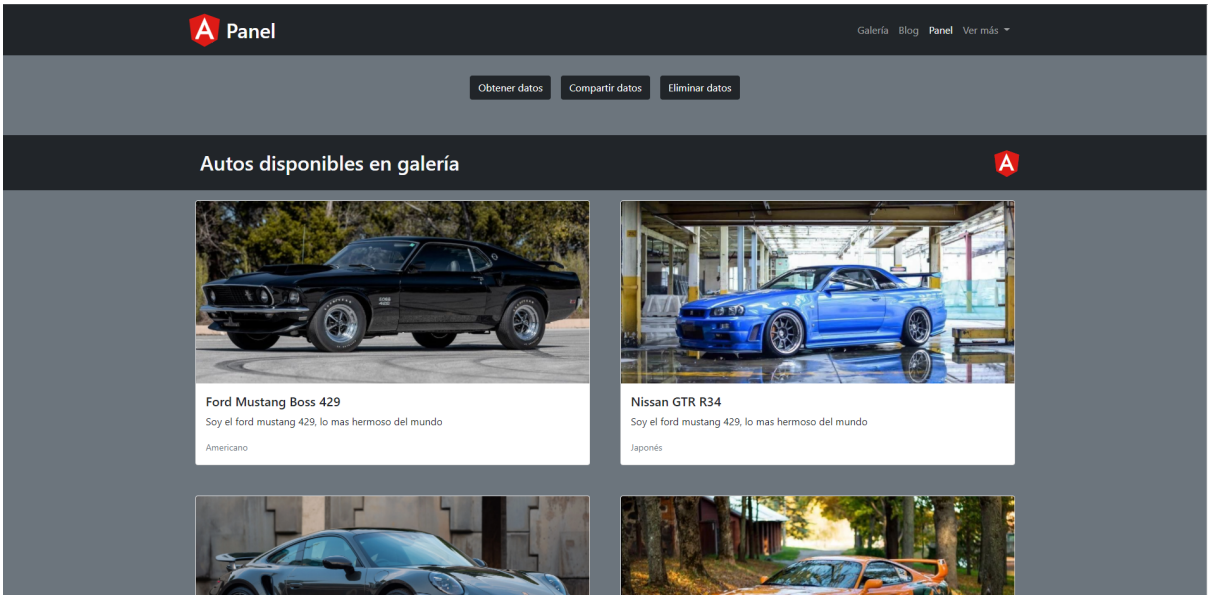


Figura 23

Panel



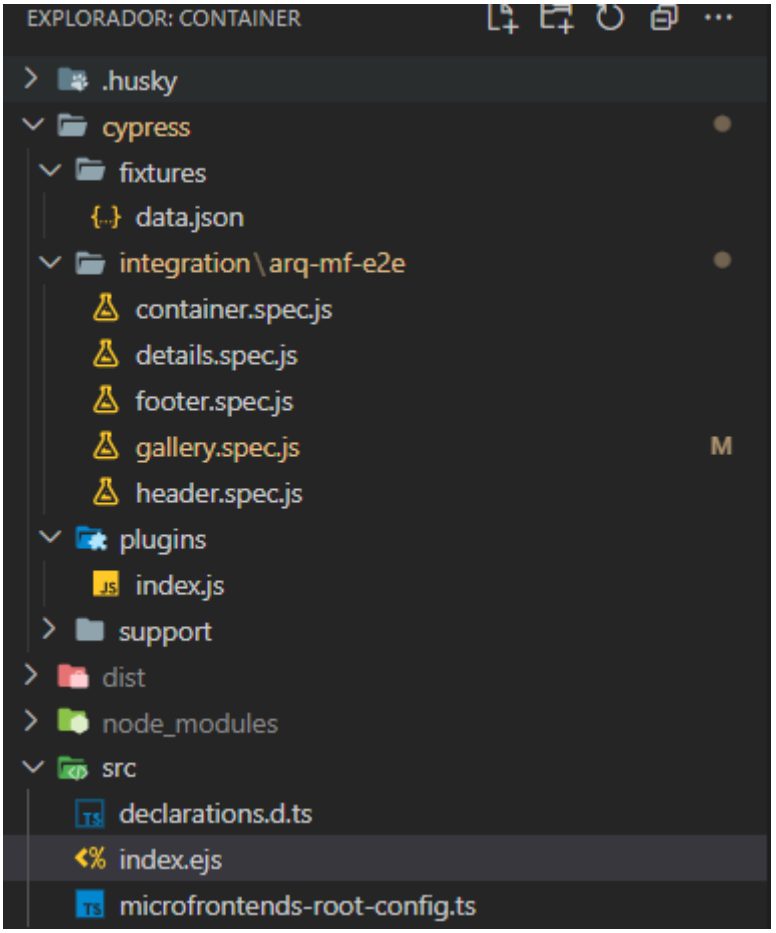
Pruebas

Las pruebas son fundamentales en cualquier sistema y poseen la misma importancia, ya que también hacen parte del código. En el caso de los microfrontends las pruebas no cambian mucho, como cada microfrontend es un proyecto aparte, cada uno será responsable de sus pruebas unitarias. Para las pruebas de integración y pruebas e2e, si hay una pequeña variación; estas se deben realizar en el contenedor, ya que es quien se encargará de orquestar los microfrontends.

Es importante resaltar que las pruebas de integración en una arquitectura de microfrontends tienen mucho más peso, ya que permiten saber que todos los microfronteds se ejecutan correctamente.

En este caso se utiliza cypress, un framework para testing fácil de manejar, este se instala en las dependencias de desarrollo en el contenedor y permite probar tanto las integraciones como las funcionalidades e2e.

Figura 24
Pruebas



CI/CD - Integración y Despliegue Continuo

Los microfrontends poseen dos grandes retos y el despliegue es uno de ellos, ya que se requiere realizar múltiples integraciones y no generar dependencias en le proceso, para ellos se utiliza el servicio de travis ci, el cual permite automatizar las partes del desarrollo de software relacionadas con la implementación, pruebas y construcción del proyecto, lo que facilita la integración y despliegue continuo.

Para ello será necesario configurar en cada microfrontend un archivo llamado .travis.yml, el cual posee la siguiente estructura en todos (figura 25), la única diferencia será la ubicación del bundle y los scripts para compilar, sin embargo estos scripts se pueden renombrar en el package.json; pero la ubicación en algunos casos sí será importante, ya que al desplegar cloudfront buscará los archivos en rutas puntuales.

Figura 25

.Yml de travis

.travis.yml

```yaml
language: node_js
node_js:
  - node
env:
  global:
    # include $HOME/.local/bin for `aws`
    - PATH=$HOME/.local/bin:$PATH
before_install:
  - npm run test
  - pyenv global 3.7.1
  - pip install -U pip
  - pip install awscli
script:
  - npm run build:webpack
deploy:
  provider: s3
  access_key_id: "$AWS_ACCESS_KEY_ID"
  secret_access_key: "$AWS_SECRET_ACCESS_KEY"
  bucket: "arq-microfrontends-single-spa-lion"
  region: "us-west-2"
  cache-control: "max-age=31536000"
  acl: "public_read"
  local_dir: dist
  skip_cleanup: true
  on:
    branch: master
after_deploy:
  - chmod +x after_deploy.sh
  - "./after_deploy.sh"
```

En la configuración es importante dejar las credenciales como variables de ambiente por seguridad, para ello sus valores inician con signo $, estas es posible agregarlas en el dashboard de travis.

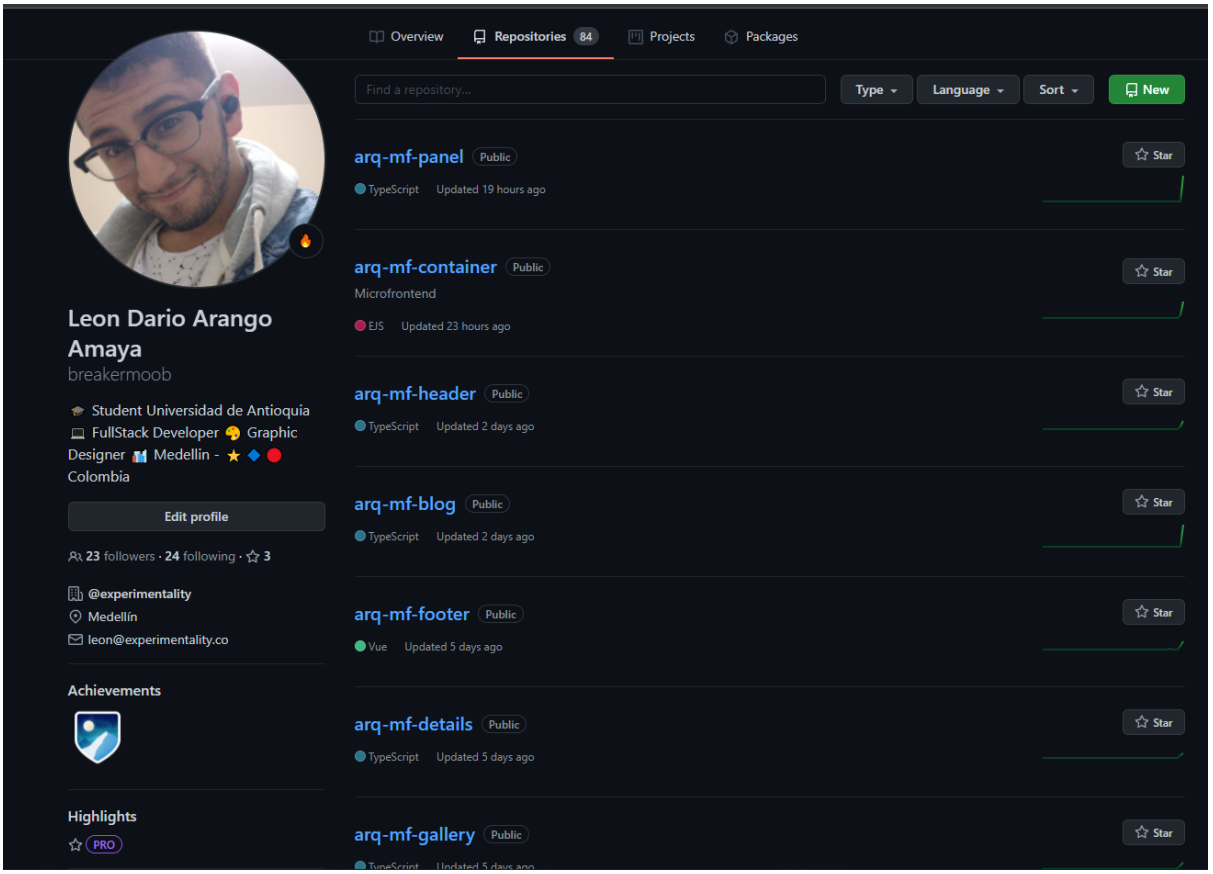
Control de versiones

El control de versiones es una gran ventaja en los microfrontends y esta arquitectura ayuda en gran medida a reducir la cantidad de conflictos que se pueden generar en el código,

por ello se crearon múltiples repositorios en github, cada microfrontend tendrá uno (figura 26).

Figura 26

Repositorios Github

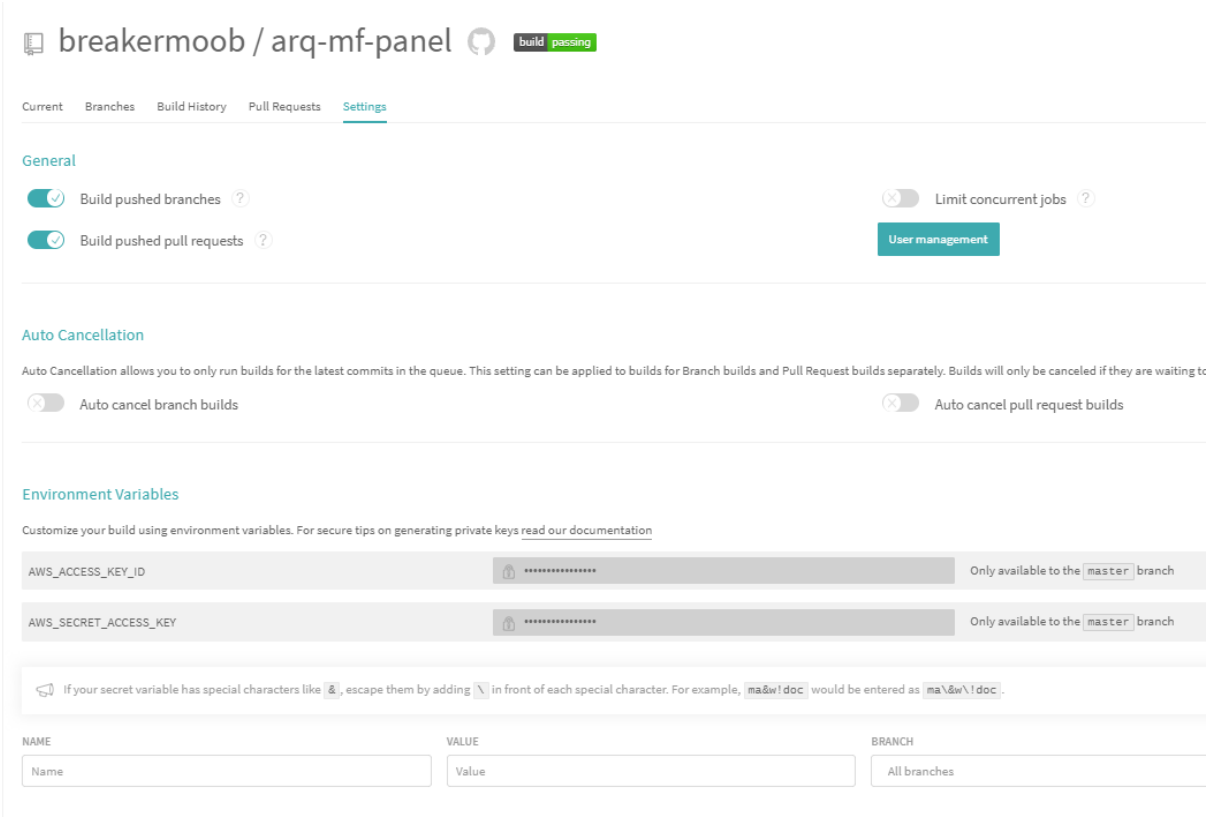


Integración con Travis CI

El dashboard de travis es sencillo de manejar, solo es necesario ingresar a https://travis-ci.org/ e iniciar sesión con github en este caso, el automaticamente reconocera los repositorios que se han creado y al momento de realizar cambios este los detectará automáticamente, si el repositorio posee el archivo de configuración, procederá con la ejecución que se ha configurado en este. En este punto es importante tener en cuenta si se requiere desplegar y sobre que ramas se realizarán los despliegues. Por ello se configuran las variables en la vista de configuraciones de cada repositorio (figura 27). En esta es posible configurar las variables que sean necesarias e indicar cual rama desplegará con estas.

Figura 27

Configuración de variables en Travis Dashboard



Luego de configurar estas y subir todos los cambios a los repositorios, este ejecutara el pipeline e indicara cuando este listo con color verde, mientras este en proceso será amarillo y si ocurre algun problema durante su ejecución, se detendrá, enseñara en la consola los logs y permanecerá en color rojo.

Despliegue en Amazon Web Services (AWS)

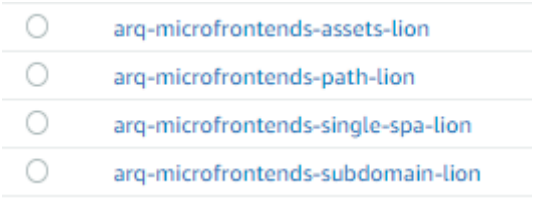
Para este despliegue se utilizaron múltiples servicios de AWS, los cuales se explicarán a continuación.

Simple Storage Service (S3)

Este es uno de los mejores servicios que ofrece AWS y se utiliza tanto para almacenar contenido como sitios web estáticos. En este caso ambos.

Se utilizaron cuatro buckets, para manejar las imágenes, el sistema principal con single spa, el subdominio en angular y el microfrontend como path del dominio.

Figura 28
Buckets S3



Cada uno de estos se debe configurar con las políticas necesarias para su uso, en este caso poseen permisos de lectura, ya que es necesario que cloudfront pueda acceder a estos.

Algo importante es que s3 entrega sus recursos de dos formas, como REST o cómo website, por ello en algunos casos cloudfront puede bloquear estos recursos (Michael, 2015), una forma fácil de manejar esto es exponer el sitio como alojamiento de sitios web estáticos (figura 29)

Figura 29

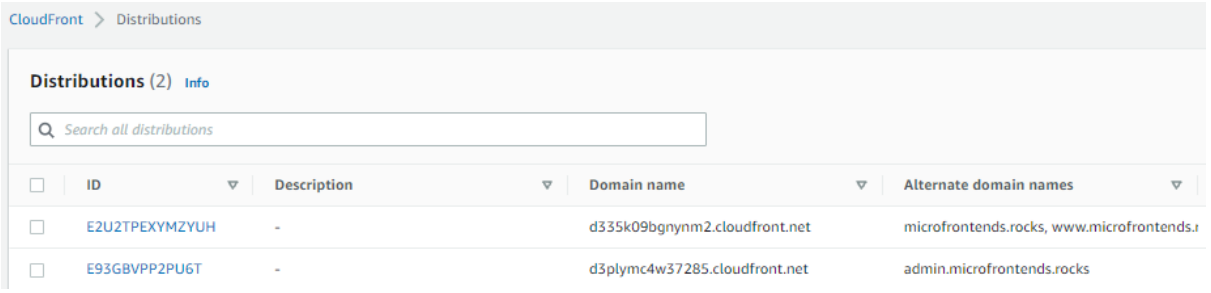
Alojamiento de estáticos

CloudFront

Este es el Content Delivery Network (CDN) de AWS, el cual permite distribuir contenido con baja latencia y alta velocidad a nivel mundial. En este caso se utiliza para conectar los microfrontends acompañado de Route 53. Para ello se crearon dos distribuciones con algunos comportamientos. (Figura 30)

Figura 30

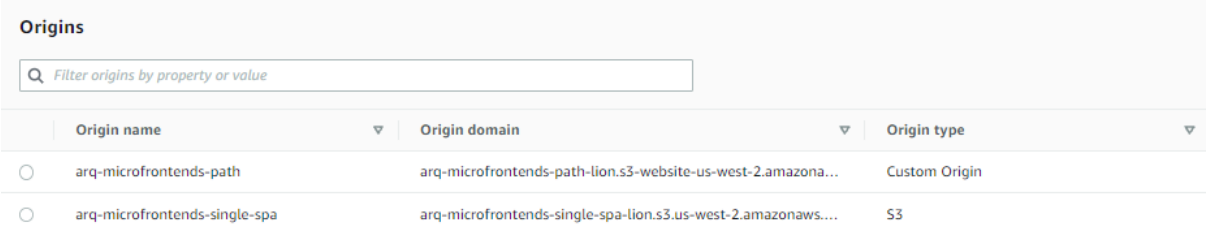
Distribuciones de cloudfront



Una de las distribuciones se encargará de manejar el dominio principal https://www.microfrontends.rocks y todo lo que contenga, la otra manejara el subdominio https://admin.microforntends.rocks .

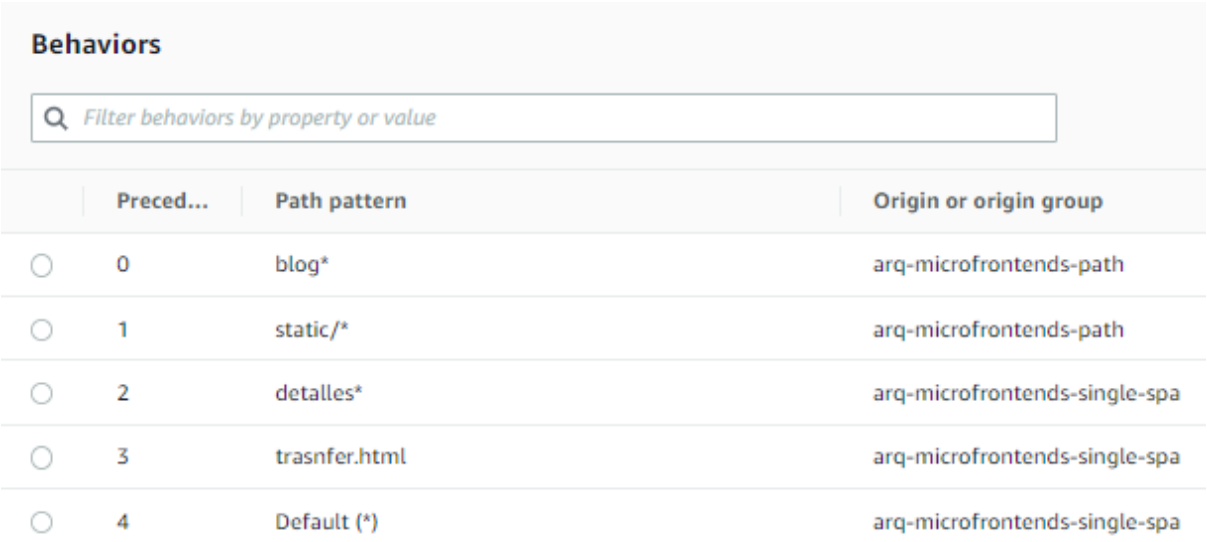
En el caso del dominio principal es necesario crear dos orígenes, uno para el sistema con single spa y otro para el path del blog (figura 31).

Figura 31
Orígenes



Y para el caso de los comportamientos tendremos los siguientes (figura 32), donde el blog será el path en el cual cloudfront encontrará este tipo de microfrontend y static son todos los recursos del mismo. Y los demás estarán asociados a las rutas de single spa. El **transfer.html** será explicado en la parte de comunicaciones entre microfrontends.

Figura 32
Comportamientos

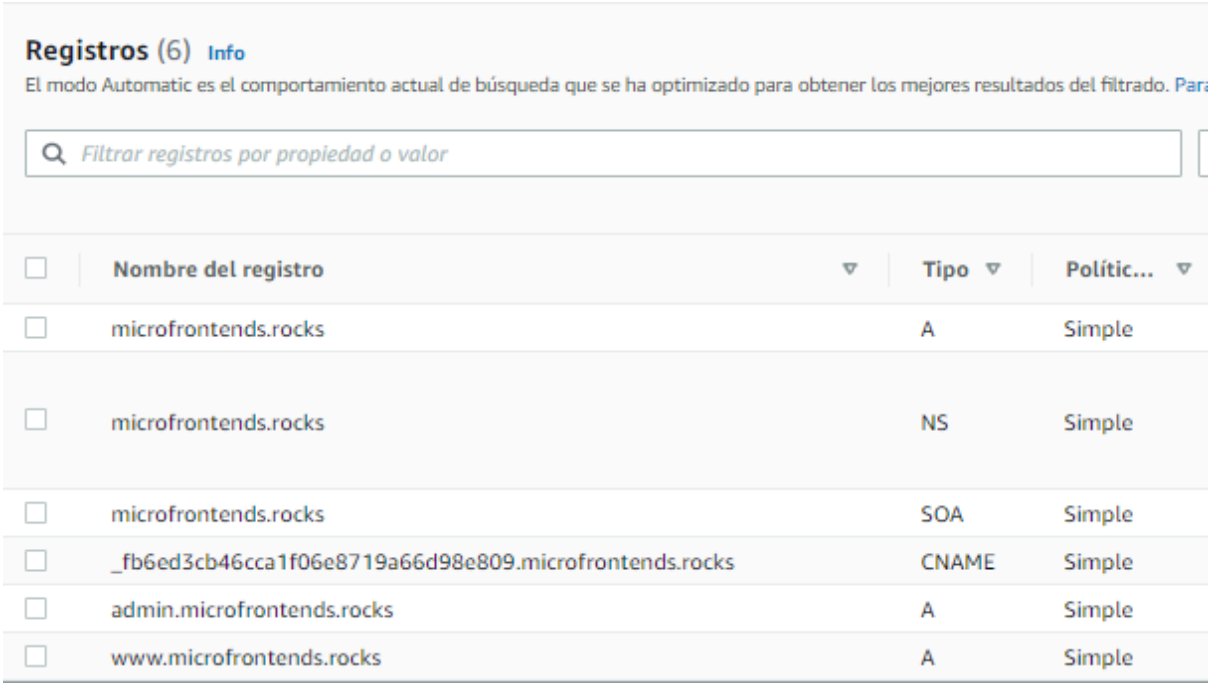


Route 53

Este es el servicio de Domain Name Server (DNS) de AWS, el cual se utilizará para dirigir el tráfico del dominio microfrontends y que pueda llegar a las distribuciones de cloudfront según sea el caso, esto será un dominio alternativo en cloudfront. En este se debe crear una zona alojada, la cual posee las siguientes configuraciones. (Figura 33)

Figura 33

Registro de la zona alojada



El dominio debe registrarse en cloudfront como alternativo y para ello es importante registrarlo en AWS, así que al momento de registrar las distribuciones de cloudfront se deberá enlazar con el proveedor de dominio, por medio de un CNAME clave valor.

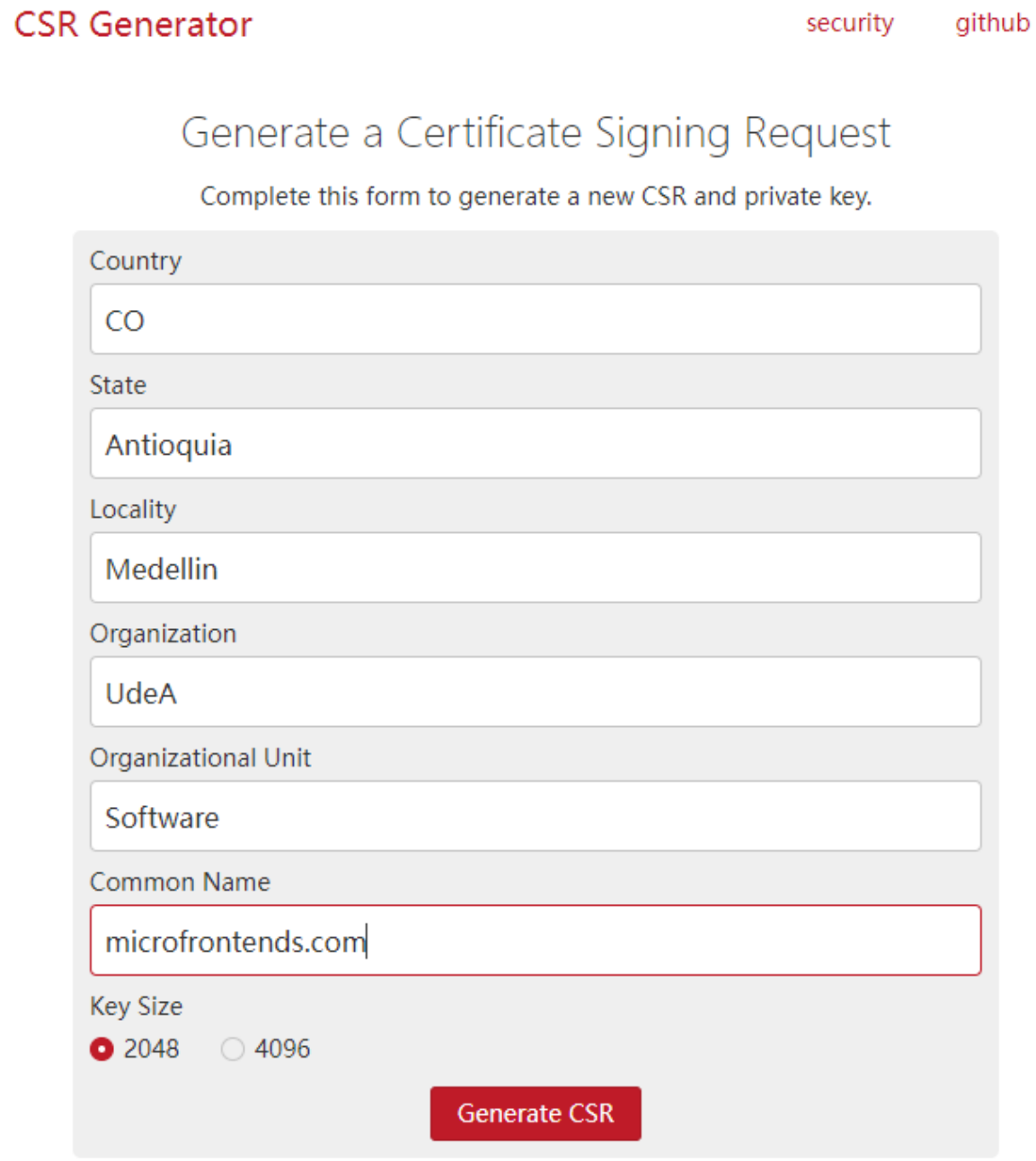
Adquisición del dominio

En este caso se utilizó un dominio de Name.com, la cual provee el dominio y el certificado SSL, para poder tener un sitio más seguro bajo el protocolo HTTPs, para ello se utilizó el pack de github education que ofrece un dominio gratis en .rocks.

Para hacer uso de este solo es necesario adquirirlo y seguir los pasos que la plataforma indica, en el momento que solicite un certificado CSR, solo se requiere ingresar en un generado de estos como lo es: https://csrgenerator.com/ allí se llenan los datos (Figura 34)

Figura 34

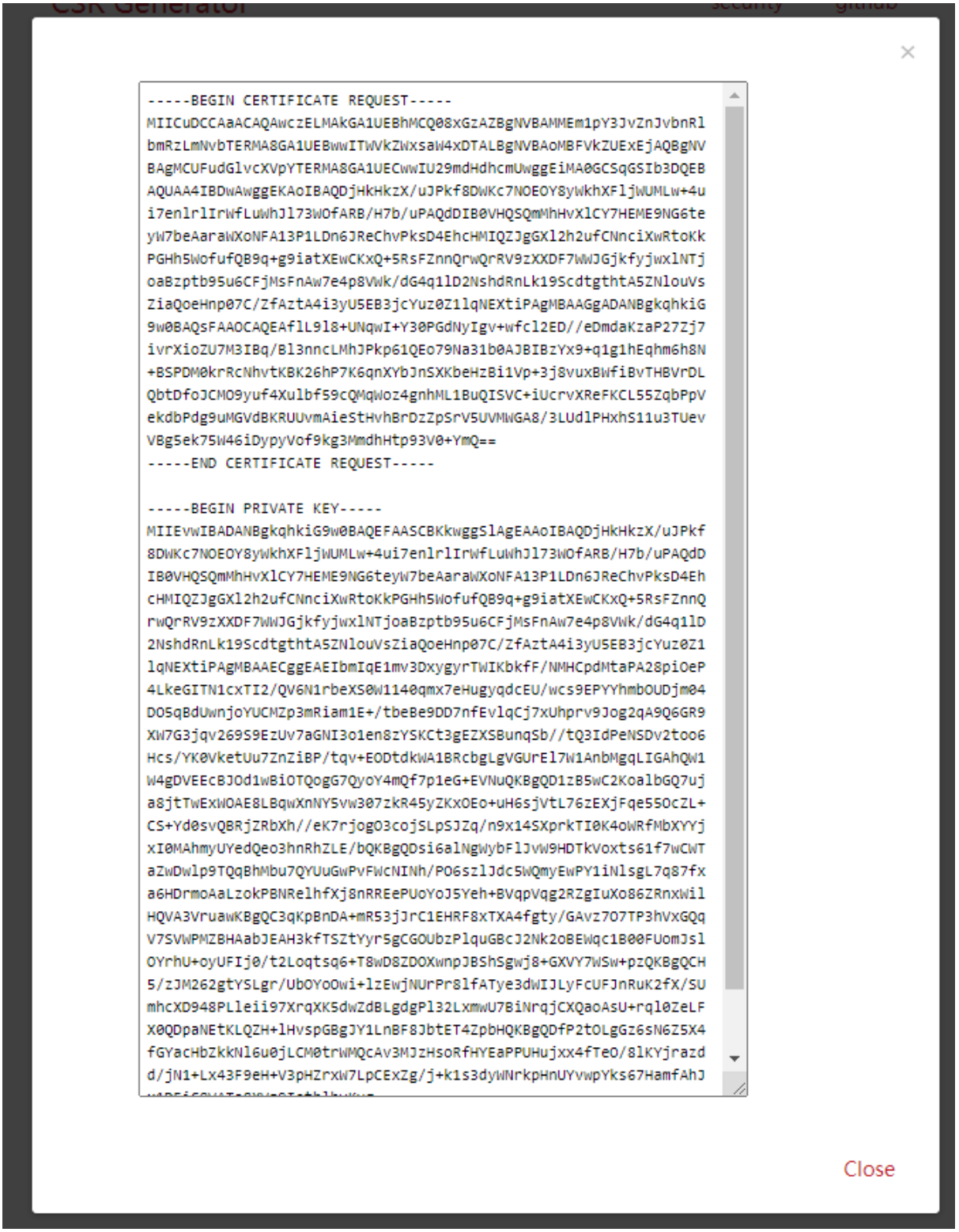
Datos requeridos en un CSR



El cual generará el CSR y la clave privada de este (figura 35).

Figura 35

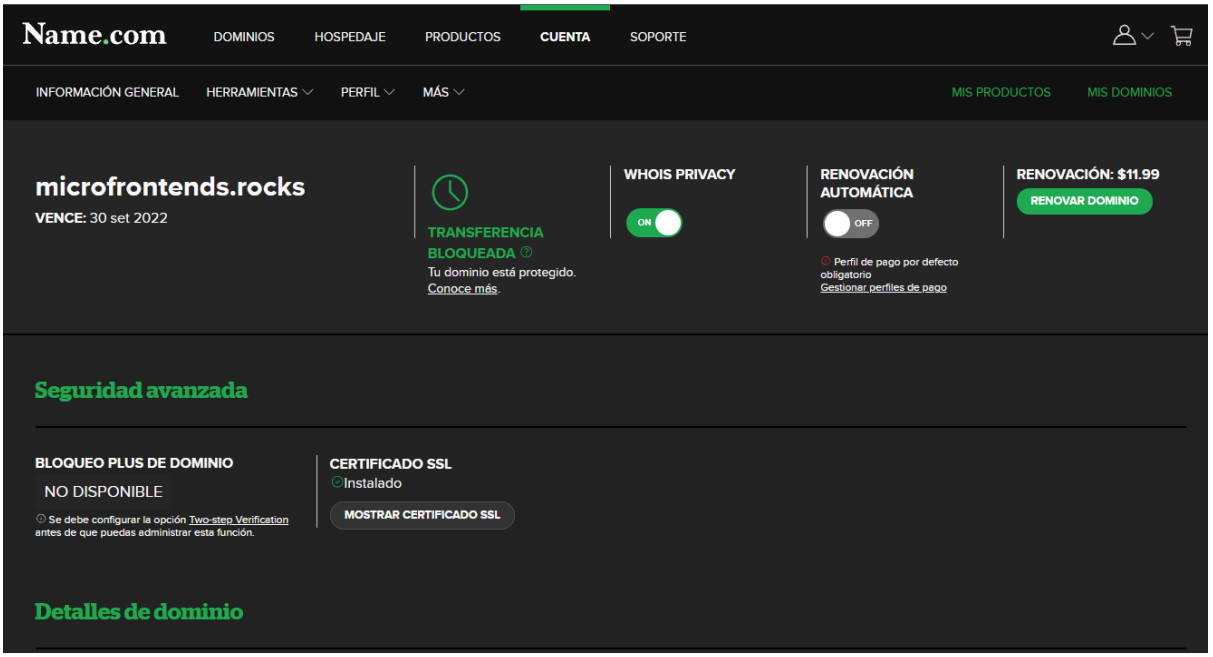
Datos generados



La clave privada es importante conservarla, ya que esta puede ser necesaria para validar la autenticidad del certificado. Luego de registrar el CSR en Name.com quedará listo el certificado para ser utilizado, (figura 36)

Figura 36

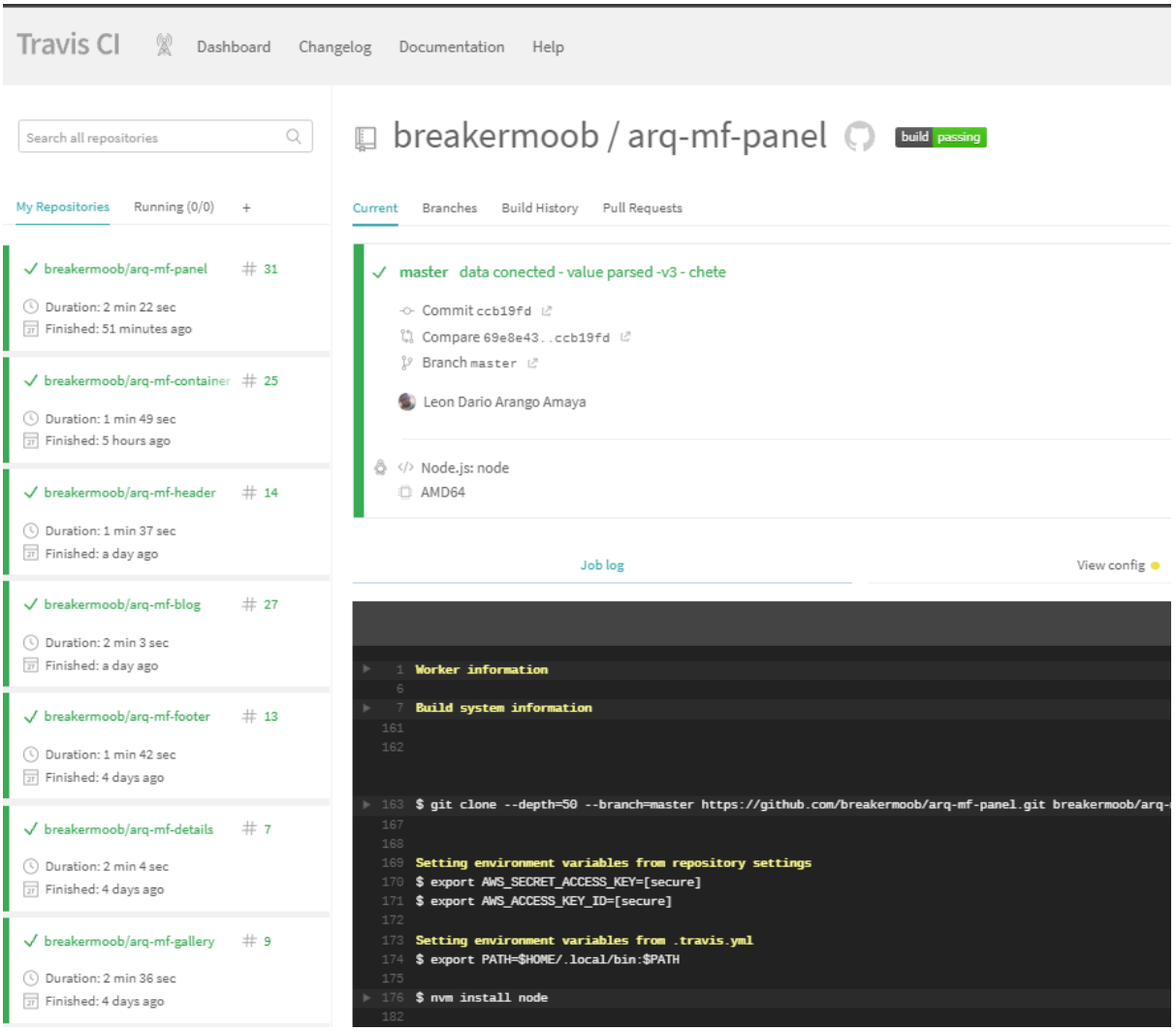
Dominio con SSL



Luego de realizar todo esto ya los despliegues se completaron correctamente y podremos observarlos en el dominio o subdominio según sea el caso. Además en el dashboard de travis aparecerá en verde. (Figura 37)

Figura 37

Despliegue exitoso de los microfrontends



Comunicaciones en los microfrontends

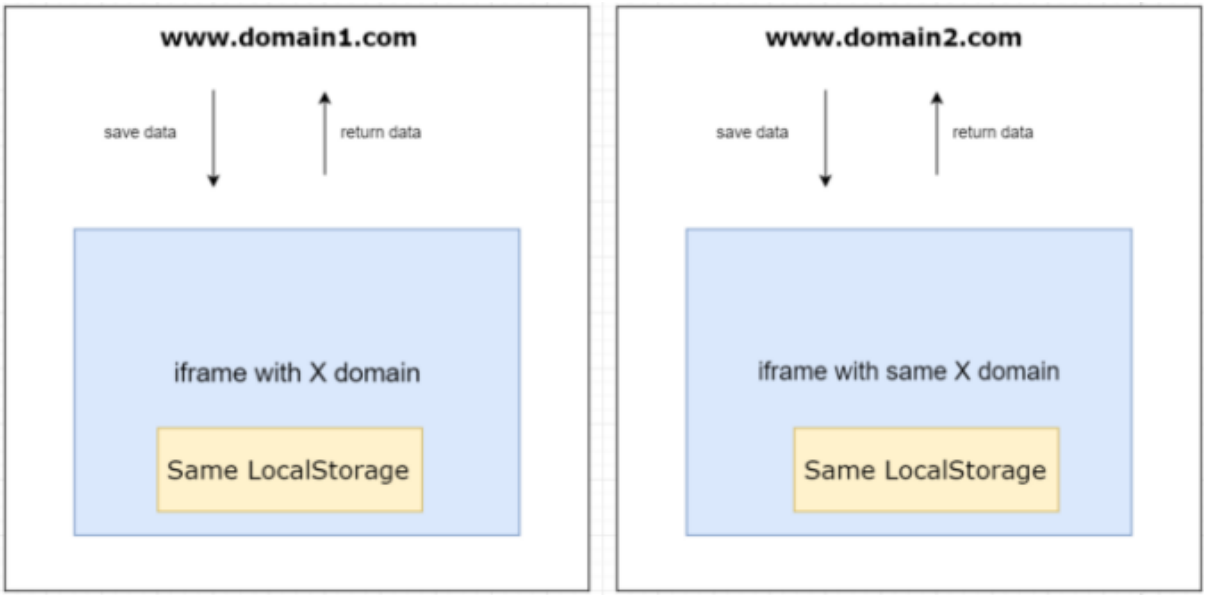
Se dejó este apartado luego del despliegue, ya que es necesario tener muy claro como es este para abordar las comunicaciones.

Anteriormente se habló sobre los medios en el cliente que sirven para almacenar y transferir datos; para este caso utilizaremos unos de ellos, el local storage. Se utiliza este, ya que posee una buena cantidad de almacenamiento y persistencia de los datos. Este en el caso del sitio principal y el blog funcionan bastante bien, ya que la comunicación, al estar asociados al mismo dominio www.microfrontends.rocks/ y www.microfrontends.rocks/blog poseen un medio directo para compartir información, realmente no importa que tan diferentes sean estos sistemas, si poseen el mismo dominio ambos pueden compartir el local storage. Por otro lado, el subdominio no comparte un medio directo con el dominio y para ello se utilizará una estrategia de cross-origin, la cual permite compartir información utilizando un iframe para embeber parte del dominio.

Como se puede observar en la siguiente imagen (Figura 38), el dominio1 y el dominio2 pueden compartir su local storage utilizando iframes, sin embargo no es solo esto, ya que de ser así sería bastante inseguro, así que hay algunas condiciones que se deben tener en cuenta.

Figura 37

Conexión de dominios por medio de un iframe



Lo primero es que el sitio que contiene el iframe, será quien solicite los datos utilizando la función de javascript postMessage (*Window.postMessage(), 2021*), el sitio embebido en el iframe podrá responde al mismo origen que le ha solicitado los datos, siempre y cuando se esté escuchando el evento, para ello se utiliza un listener de javascript con el evento message. Es importante aclarar que no es necesario embeber todo el sitio en el iframe, ya que esto puede generar una carga innecesaria, para ello se utilizara un archivo de tipo html que pertenezca al dominio que queremos comunicar, en este caso el trasnfer.html (Figura 39). Este no es más que un html con un script que permite escuchar los eventos y responder a ellos según sea el caso, es posible realizar un crud sobre el local storage y hasta más si se requiere. De hecho esta es una forma bastante particular de realizar microfrontends, ya que no requiere tecnologías complejas.

Figura 39

Conexión de dominios por medio de un iframe

```
<script>
  const dominios = [
    "https://www.microfrontends.rocks",
    "https://admin.microfrontends.rocks",
  ]
  window.addEventListener("message", messageHandler, false);

  function messageHandler(event) {
    if (!dominios.includes(event.origin))
      return;
    const { action, key, value } = event.data
    if (action == 'save') {
      console.log(key);
      window.localStorage.setItem(key, JSON.stringify(value))
    } else if (action == 'get') {
      event.source.postMessage({
        action: 'returnData',
        key: 'returnData',
        value: window.localStorage.getItem(key)
      }, '*')
    } else if (action == 'delete') {
      window.localStorage.removeItem(key);
    }
  }
</script>
```

También será necesario añadir el escuchador de eventos en el dominio que contiene el iframe para capturar las respuestas (figura 40).

Figura 40

Escuchador de eventos

```
  const iframe = document.getElementById('ifr')
  window.addEventListener("message", this.messageHandler, false);

}

messageHandler(event) {
  const { action, key, value } = event.data;
  if (action == 'returnData') {
    window.localStorage.setItem(key, value);
  }
}
```

Ya se ha hablado de los diferentes tipos de microfrontends y cómo implementar cada uno de ellos, pero como se ha mencionado anteriormente, existen otras formas de reutilizar componentes, ya sea con librerías para múltiples componentes o web components para uno solo.

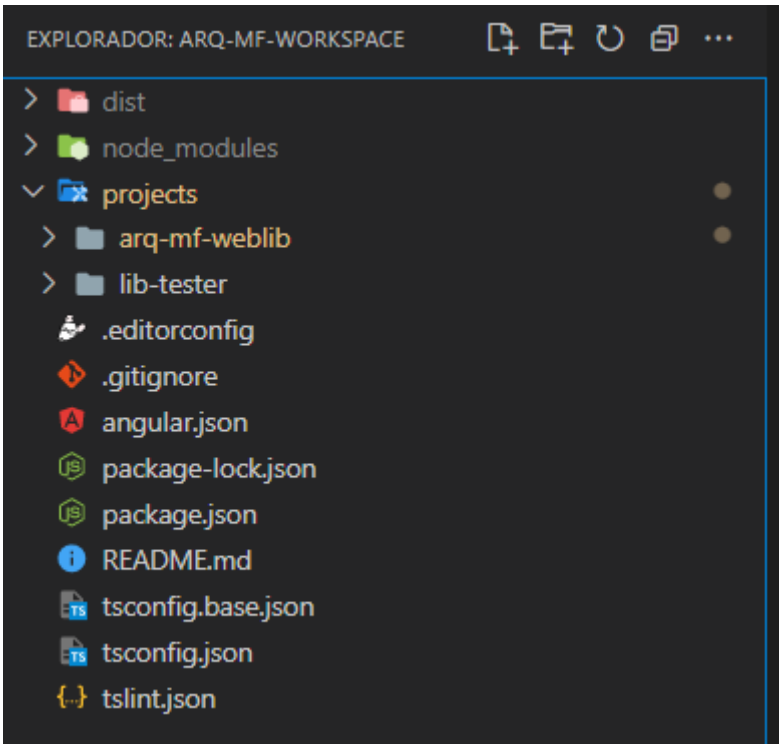
Librería

Se utilizará las librerías de Angular, para general la librería de componentes, esta no es más que un grupo de componentes que se pueden reutilizar en múltiples proyectos según sea el caso, para ello se utiliza la documentación oficial como guía (Angular, 2019).

Inicialmente se crea un workspace con el comando "ng new my-workspace --create-application=false" el cual permite generar un ambiente de trabajo para manejar tanto librerías y aplicaciones, luego se debe crear una librería con "ng generate library my-lib" y una aplicación con "ng new my-application". Al finalizar se obtendrá un proyecto como en la siguiente imagen(figura 41)

Figura 41

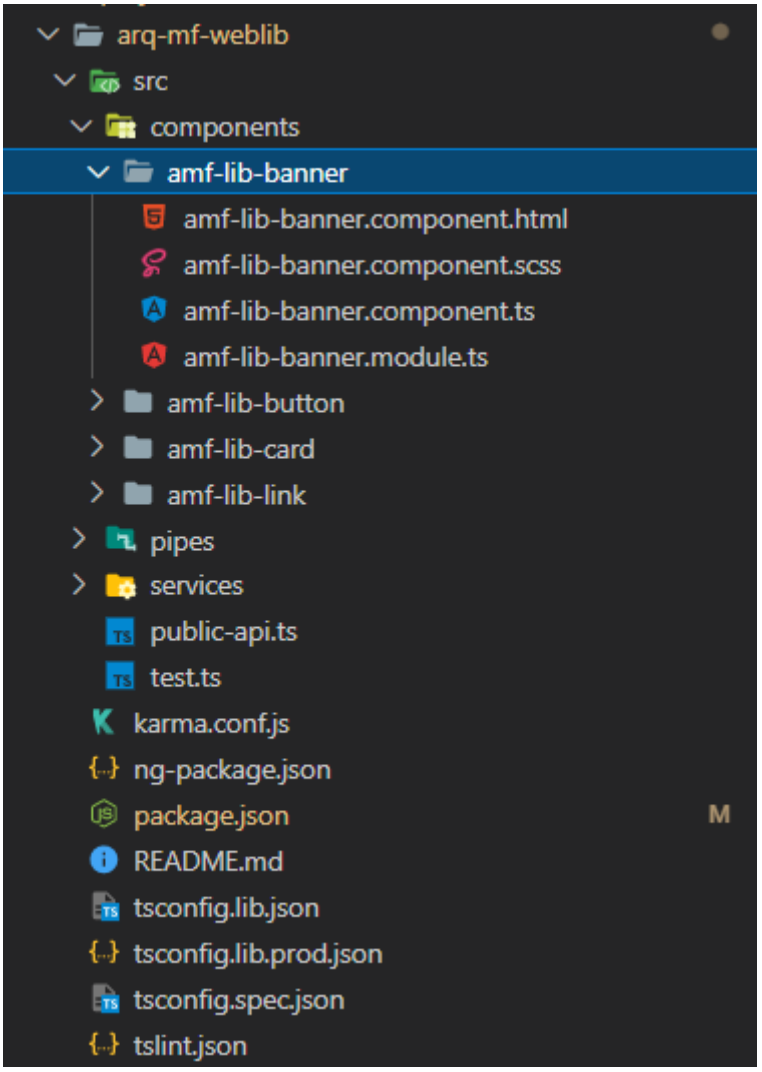
Proyecto Librería



Luego se implementan cada uno de los componentes, se pueden agrupar por módulos o generar un módulo por componente para no generar cargas innecesarias en el código de cada proyecto donde se implemente la librería.

Figura 42

Componentes de la librería



Sin embargo es importante exportar los componentes y para ello se deben exportan en cada módulo y en el archivo public-api.ts como muestra la imagen (Figura 43)

Figura 43

Componentes de la librería - exports

public-api.ts

projects > arq-mf-weblib > src > public-api.ts

You, 2 hours ago | 1 author (You)

```
/*
 * Public API Surface of amf-lib
 */

export * from './components/amf-lib-button/amf-lib-button.component';
export * from './components/amf-lib-button/amf-lib-button.module';
export * from './components/amf-lib-link/amf-lib-link.component';
export * from './components/amf-lib-link/amf-lib-link.module';
export * from './components/amf-lib-banner/amf-lib-banner.component';
export * from './components/amf-lib-banner/amf-lib-banner.module';
export * from './components/amf-lib-card/amf-lib-card.component';
export * from './components/amf-lib-card/amf-lib-card.module';
```

Luego de esto ya se puede publicar en npm o generar el build y enlazarla con la aplicación que se ha generado para probar los componentes.En este caso se crearon cuatro (botón, enlace, banner y card) (figura 44) Los cuales se utilizaran en los microfrontends.

Figura 44

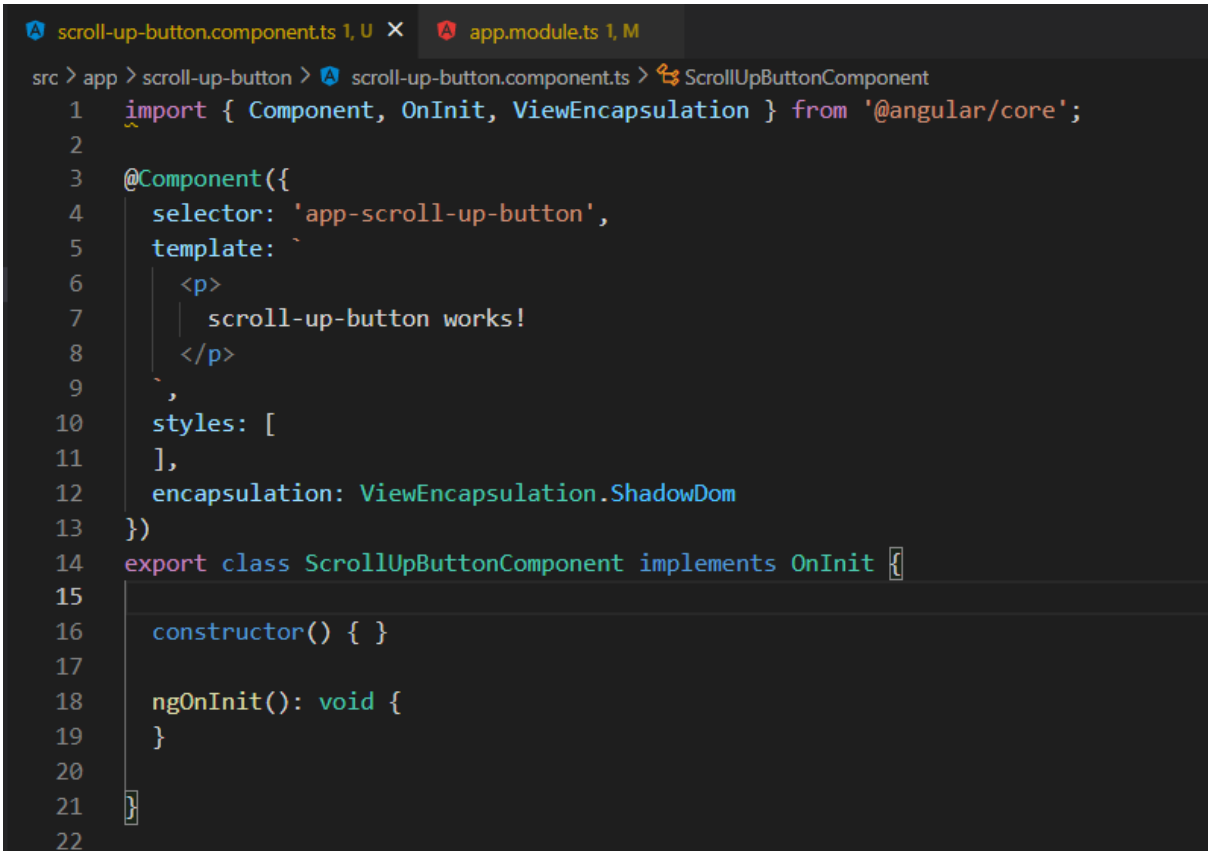
Componentes de la librería implementados

Web Component

Es posible realizar los web components desde cero, sin embargo existen tecnologías que facilitan la implementación como Angular, el cual posee un módulo especial para estos @angular/elements. Para crear web components solo es necesario crear un proyecto de angular y añadir el módulo con “ng add @angular/elements”. Además los componentes deben poseer encapsulación nativa o en las versiones más recientes de angular ShadowDom. (Figura 45)

Figura 45

Código base del web component



Luego de implementarlo es necesario realizar una configuración para que este componente sea el punto de entrada de la aplicación. (figura 46)

Figura 46

Configuración para definir el web component

```
You, seconds ago | 1 author (You)
@NgModule({
  declarations: [
    AppComponent,
    ScrollUpButtonComponent
  ],
  imports: [
    BrowserModule
  ],
  entryComponents: [ScrollUpButtonComponent],
})
export class AppModule {
  constructor(
    private injector: Injector
  ) {
    const imagenEspacio = createCustomElement(ScrollUpButtonComponent, { injector });
    customElements.define('arq-mf-scroll-up-button', imagenEspacio);
  }
  ngDoBootstrap() { }
}
```

Finalmente, para generar el bundle lo mejor es llevar todo a un solo archivo, se puede utilizar un comando como "(ng build --prod --output-hashing=none) -and (cat *.js > index.js)" para llevar todos los archivos de javascript a un index.js. Este se puede subir en un cdn para posteriormente importarlo y utilizarlo en múltiples los proyectos.

Conclusiones

Si preguntan ¿Es posible implementar una arquitectura en microfrontends y aprovechar todos los beneficios de ella en un proyecto? La respuesta desde el 2019 es si, tal y como se ha podido comprobar los microfrontends son una realidad, las arquitecturas en el frontend también son posibles, incluso es posible ir más allá gracias a la web moderna, sin embargo aún hay retos muy grandes que hacen el desarrollo de estos más complejo, hay tecnologías que son realmente útiles y en un futuro cercano pueden cambiar completamente la forma en la que se desarrolla el frontend.

Por otro lado, cuando hablamos de las tecnologías utilizadas es importante destacar a Single SPA, ya que es un framework que facilita mucho la implementación de una arquitectura en microfrontends, este framework, aunque aún le faltan cosas, es suficiente para satisfacer la necesidad básica de crearlos. AWS como nube es definitivamente la mejor opción, ya que permite desplegar frontends en S3 a un costo muy bajo y adicional a ello cuenta con otros servicios como Cloudfront y Route53 que facilitan el trabajo en gran medida. En cuanto a los frameworks de SPA como Vue, Angular y React funcionan muy bien

en conjunto a Single SPA y permite realizar la configuración de forma muy simple, si hablamos de otros como Svelte y Ember, aún hace falta madurar un poco más estas tecnologías, para que adaptarlas a Single SPA no sea algo complejo. En un futuro cercano pueden surgir muchas más tecnologías que faciliten el trabajo para implementar microfrontends, que puedan ayudar a resolver retos técnicos como las pruebas y la integración en menos tiempo.

El aporte que puede dejar un trabajo de este tipo es muy grande, ya que aporta al crecimiento técnico y personal, ayuda a tener más criterio para tomar difíciles decisiones y da una visión más amplia de lo que se puede llegar a construir conociendo más de este tipo de arquitecturas y tecnologías. Por ello vale la pena adentrarse más en el conocimiento.

Trabajo Futuro

Como se ha dicho anteriormente, existen retos en el desarrollo de este tipo de arquitecturas, los cuales aún se pueden automatizar más, como trabajo futuro se propone explorar más a fondo la comunicación e integración de microfrontends, teniendo en mente que son aguas profundas que pocos han explorado, aún no existen muchas definiciones en pruebas, integración y despliegue continuo o incluso en seguridad del lado del cliente en micro frontends.

Referencias

Geers, M. (2019, Agosto 12). *Micro Frontends - Extendiendo la idea de microservicio al desarrollo frontend*. Micro Frontends. https://micro-frontends-es.org/

Jackson, C. (2019, Junio 19). *Micro Frontends*. Martinfowler.Com. https://martinfowler.com/articles/micro-frontends.html

Verma, A. (2020, Junio 29). *Micro-Frontend using Web Components - The Startup*. Medium. https://medium.com/swlh/micro-frontend-using-web-components-e9faacfc101b

Web Components. (2021, Enero 30). MDN Web Docs. https://developer.mozilla.org/es/docs/Web/Web_Components

Discuss & share web components. (2016, Agosto 20). Webcomponents.Org. https://www.webcomponents.org/introduction

Using shadow DOM. (2019, Enero 12). MDN Web Docs. https://developer.mozilla.org/es/docs/Web/Web_Components/Using_shadow_DOM

Angular. (2019, Enero 17). Angular Libraries. https://angular.io/guide/libraries

Michael, M. (2015, Diciembre 3). *CloudFront + S3 Website: "The specified key does not exist."* Stack Overflow. https://stackoverflow.com/questions/34060394/cloudfront-s3-website-the-specified-key-does-not-exist-when-an-implicit-ind

Window.postMessage(). (2021, Septiembre 14). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage

Coffee Power. (2020, October 12). *Micro Frontends (Explicado) | Oswaldo Alvarez ft Edwin Torres #11* [Video]. YouTube. https://www.youtube.com/watch?v=EeoUkj3BoDY

Weincode. (2021, June 22). *Mircro frontends en Angular🌐Parte 2 - compartiendo info, enrutando, y otros cosas más* 😀 [Video]. YouTube. https://www.youtube.com/watch?v=eAg-bCL4Ob8