Micro-frontends

Hugo Hevia García (UO277553)
Andrea Auñón Antúnez (UO277876)
Luis Manuel González Baizán (UO269502)

Introducción

A la hora de desarrollar una aplicación se suele crear una aplicación de navegador potente y rica en características, también conocida como “single page app”, que se asienta sobre una arquitectura de microservicios, es decir una capa frontend y un backend dividido en distintos servicios.

Esto provoca que el frontend, normalmente desarrollado por un solo equipo, vaya creciendo haciéndolo mucho más difícil de mantener.

Para solucionar este problema surgen los micro-frontends.

¿Que son los micro-frontends?

El término Micro Frontends apareció por primera vez en ThoughtWorks Technology Radar a finales de 2016. Este término designa un estilo o arquitectura, es decir, una forma de construir aplicaciones web, que se basa en ver a las aplicaciones como una composición de características que son propiedad de equipos independientes. Cada equipo tiene un área de negocio de la que está encarga y que desarrolla todas sus características end-to-end, desde la base de datos hasta la interfaz de usuario.

Debido a esta división los equipos de desarrollo no se separan por tecnología, sino que son equipos transversales con todos los perfiles necesarios para desarrollar la funcionalidad completa.

Es muy importante tener claro que no son una solución a un problema tecnológico sino una solución a un problema organizativo que pretende facilitar el desarrollo de aplicaciones complejas, sobre todo a nivel de coordinación de los diferentes equipos ofreciéndoles mayor autonomía focalizando la distribución del trabajo en una distribución funcional y no tecnológica.

Conceptos clave:

- **Independencia tecnológica**: Cada equipo debe poder elegir y actualizar su parte sin tener que coordinar con otros equipos.
- **Separación de equipos**: Te permite tener un foco más específico en cada equipo y poder hacer una gestión más específica y detallada dirigida al objetivo.
- **Establecer prefijos por equipos**: Hay ciertos recursos que deberán de ser compartidos entre equipos y cada equipo ha de tener una nomenclatura específica para evitar mezclar recursos.
- **Diseño web resiliente**: Cada equipo trabajará en una sección aislada del sistema y esto ayudará a solucionar los problemas más rápido y ser adaptativo.
- **Usar eventos nativos del navegador**: La mejor opción es usar los eventos nativos del navegador que permiten comunicarse entre equipos, en el caso de que no sea suficiente sé tratará de mantener una API común lo más simple posible.

Ventajas

- Bases de código más pequeñas y mantenibles: son más fáciles de trabajar por desarrolladores y evitamos el acoplamiento involuntario entre componentes y empujamos a que sea un frontend más explícito.

- **Mejoras incrementales:** Podemos ir poco a poco desacoplando y estrangulando sistemas monolito. Además de que es asumible por la empresa, ya que no es una mega tarea de reescribir todo el código, nos ayuda a experimentar con nuevas soluciones de manera más aislada que antes. Además, reduce la deuda técnica

- **Despliegue independiente:** Al hacer cada despliegue independiente, reduce el alcance lo que provoca una reducción del riesgo asociado. Cada micro-frontend tendrá su propio despliegue por lo que se pude desplegar un cada uno nada más esté listo si necesidad de esperar al resto.

- **Equipos independientes:** Cada equipo trabaja sobre una sección del producto diferente por lo que tiene independencia lo que los permite ser más efectivos y rápidos.

- Funcionalidad más escalable: Levantar un nuevo microfrontend es más barato que levantar un monolito.

Desventajas

- **El fragmento más lento determina el tiempo de respuesta de toda la página**: Una posible solución es guardar los fragmentos en caché. Para los más costosos o de difícil producción sería buena idea excluirlos del procesamiento inicial generándolos de manera asíncrona en el navegador.

- **Obligación de definir los puntos de integración y las interfaces entre los diferentes equipos**: Genera un problema de coordinación de equipos y de integración de los componentes del front-end resultante. Es importante recordar que cada propósito de negocio ha de estar aislado. También hemos de tener en cuenta que cada equipo tiene unos objetivos y unos retos diferentes pero que han de estar acordados entre todos. Es muy importante que los contratos entre todas las partes y la API estén muy bien definidos para que la comunicación sea lo más fluida posible.

- **No es válida para equipos o proyectos pequeños**: Es aplicada en grandes proyectos con diferentes equipos distribuidos y que cuentan con una gran infraestructura. Por este motivo este sistema es utilizado por las compañías más grandes, ya que son las que le sacan el mayor partido posible.

- **Posibles problemas de fragmentación**: Si fragmentas demasiado tu sistema es muy posible que al final te quedes con fragmentos que no aporten valor a tu objetivo. Es importante que haya un motivo para la división. Además, los paquetes generados de forma independiente pueden causar duplicidad de dependencias, aumentando el tamaño en bytes que enviaremos a nuestros usuarios. Una posible solución es crear una biblioteca de dependencias y decidir cuáles serán genéricas.

- **Diferencias entre entornos:** Los micro frontend tienen la capacidad de poder desplegarse en entornos propios para el desarrollo y probarse de forma independiente. Es posible que el desarrollo

no se comporte igual por separado, que integrado en conjunto y desplegado en un entorno de producción. Para ello debemos probar en entornos que sean lo más parecidos a producción y hacer pruebas manuales y automatizadas para detectar problemas.

Pilares de su arquitectura

1.Definición: Lo primero que se debe hacer es CÓMO identificamos un micro-frontend, ¿es división horizontal o vertical? DIVIDIR HORIZONTALMENTE significaría identificar micro-frontends dentro de la misma vista, por lo tanto, varios equipos se encargan de la composición de la página coordinándose para el resultado final presentado a un usuario. DIVISIÓN VERTICAL asignaría una vista específica, o un grupo de vistas, a un equipo, lo que le permitiría dominar un área específica de la aplicación.

2.Composición: Hay tres tipos de composición.

2.1. Del lado del cliente: Se puede usar simplemente usando markups o JavaScript. La mayoria de librerías que habilitan esta composición usan fragmentos en App Shell y además proporcionan CLI.

2.2. Del lado del borde: Aborda el problema de la escalabilidad, ya que esta composición está diseñada para entornos informáticos sin servidor.

2.3. Del lado del servidor: Es más complejo de usar que los anteriores, además, hay que hacer un buen diseño ya que no tiene librerías o frameworks externos que ayuden en la escalabilidad o disponibilidad software.

3.Comunicación: La decisión final es encontrar una buena forma de comunicación entre micro-frontends teniendo en cuenta que son unidades independientes que deben estar completamente desacopladas entre sí.
Cuando decidimos tener múltiples micro-frontends en la misma página, debemos decidir cómo un micro-frontend debe notificar cuando ocurre un evento o una interacción del usuario para coordinar cambios en otros micro-frontends.
En este caso, podríamos usar eventos personalizados o una biblioteca de emisores de eventos para que nuestras micro-frontends simplemente emitan/envíen un evento y quien está escuchando reaccione a ese evento específico.
Sin embargo, ya sea que identifiquemos múltiples micro-frontends por página o consideremos un micro-frontend como una vista completa, debemos decidir cómo una vista intercambiaría datos con otra vista.
Podemos decidir utilizar una solución de almacenamiento web en la que cada vez que se carga una nueva vista se busca dentro del almacenamiento web para encontrar la información necesaria o un enrutamiento de URL.