

Micro-frontends

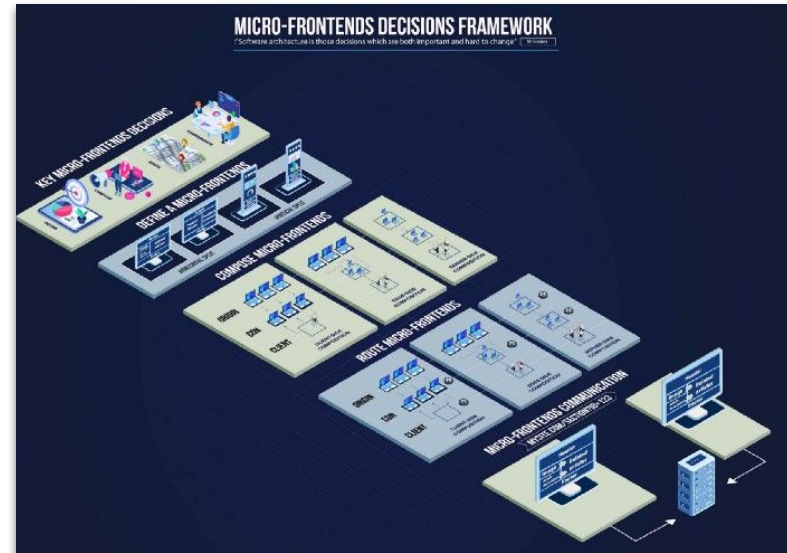


Nombres:

Diego Villa García UO277188

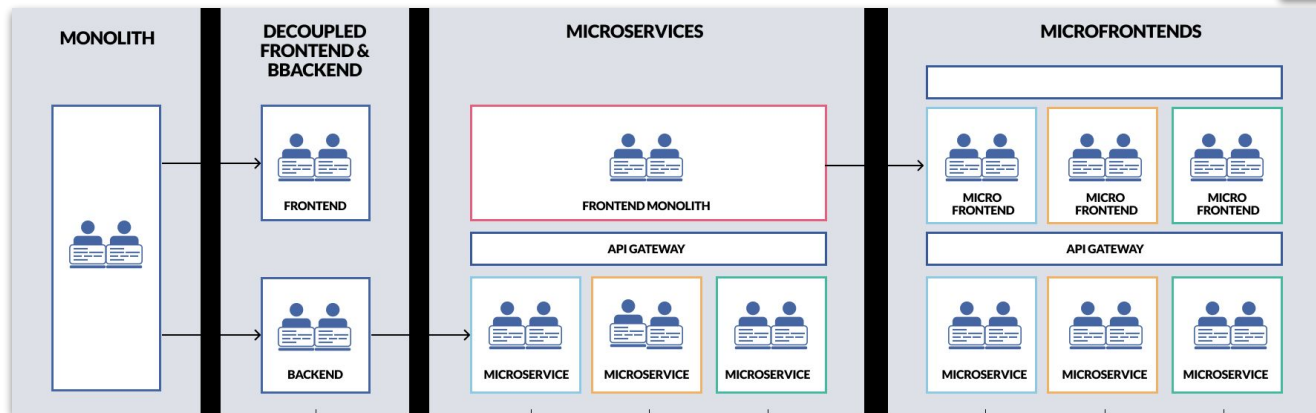
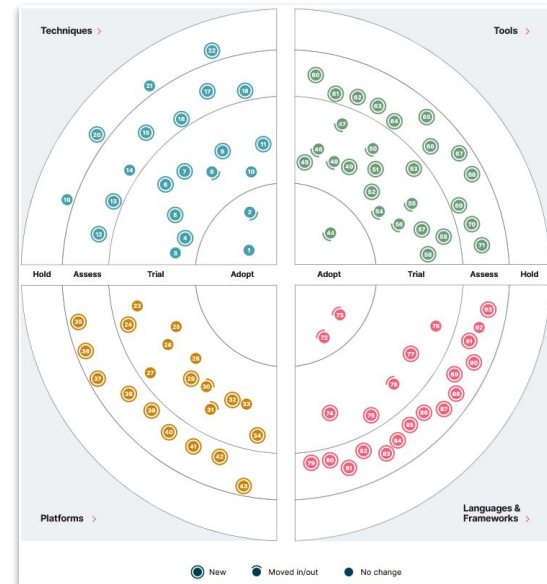
Juan Mera Menéndez UO277406

Héctor Martín Gutiérrez UO239198



Explicación general y origen

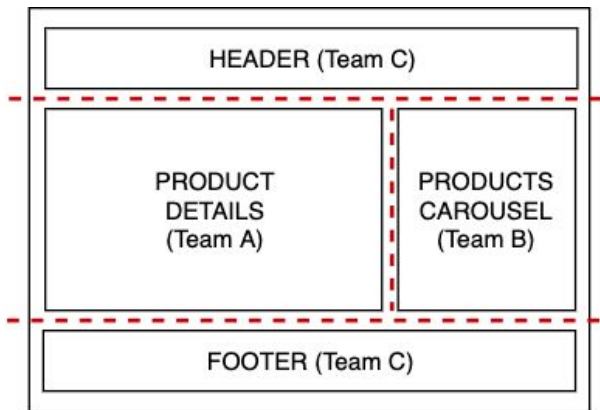
Primera mención en el **Technology Radar** de 2016.
Es la aplicación de los microservicios al frontend.



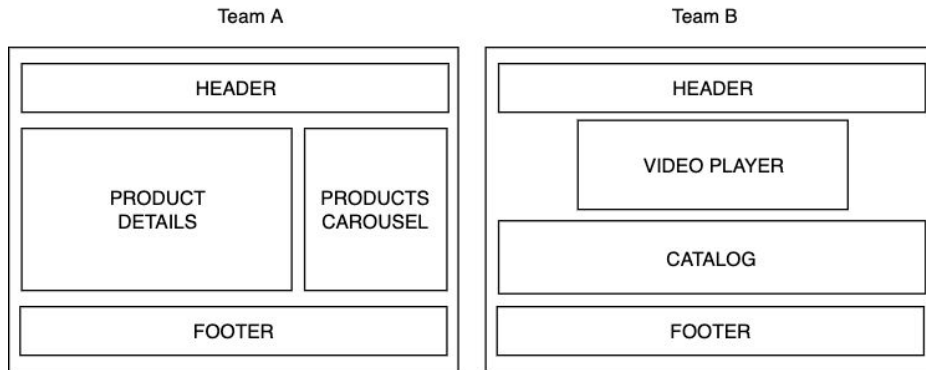


Directrices para diseñar una arquitectura de micro-frontends

¿Cómo se dividirán las vistas?



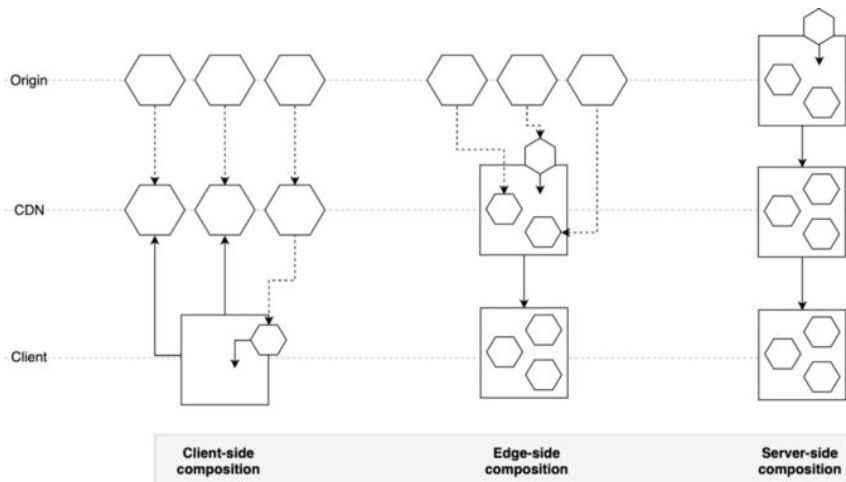
HORIZONTAL SPLIT



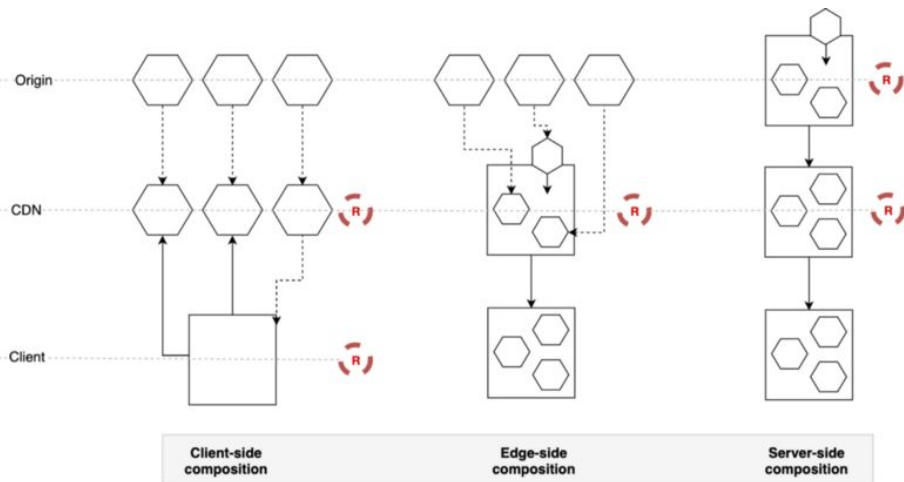
VERTICAL SPLIT

Directrices para diseñar una arquitectura de micro-frontends

¿Dónde se combinan las diferentes vistas?



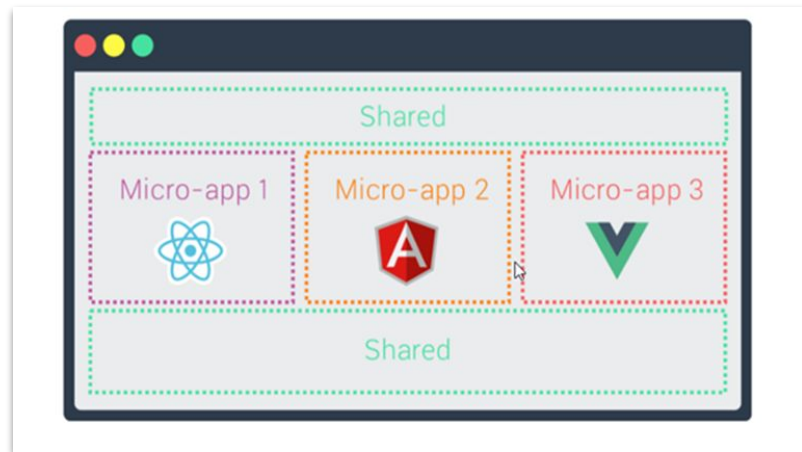
¿Dónde y cómo se realizará la lógica de enrutamiento?



Enfoques de integración

- Composición de plantillas del lado del servidor
- Integración en tiempo de compilación

```
{
  "name": "@feed-me/container",
  "version": "1.0.0",
  "description": "A food delivery web app",
  "dependencies": {
    "@feed-me/browse-restaurants": "^1.2.3",
    "@feed-me/order-food": "^4.5.6",
    "@feed-me/user-profile": "^7.8.9"
  }
}
```





Integración en tiempo de ejecución

- Con Iframes
- Con JavaScript

```
<body>
  <h1>Welcome to Feed me!</h1>

  <iframe id="micro-frontend-container"></iframe>


  <script type="text/javascript">
    const microFrontendsByRoute = {
      '/': 'https://browse.example.com/index.html',
      '/order-food': 'https://order.example.com/index.html',
      '/user-profile': 'https://profile.example.com/index.html',
    };


    const iframe = document.getElementById('micro-frontend-container');
    iframe.src = microFrontendsByRoute[window.location.pathname];
  </script>
```

```
<!-- Estos scripts no muestran nada inmediatamente -->
<!-- En su lugar, adjuntan funciones de punto de entrada a `window` -->
< script src = "https://browse.example.com/paquete.js" ></ script >
< script src = "https://order.example.com/bundle.js" ></ script >
< script src = "https://profile.example.com/bundle.js" ></ script >
```

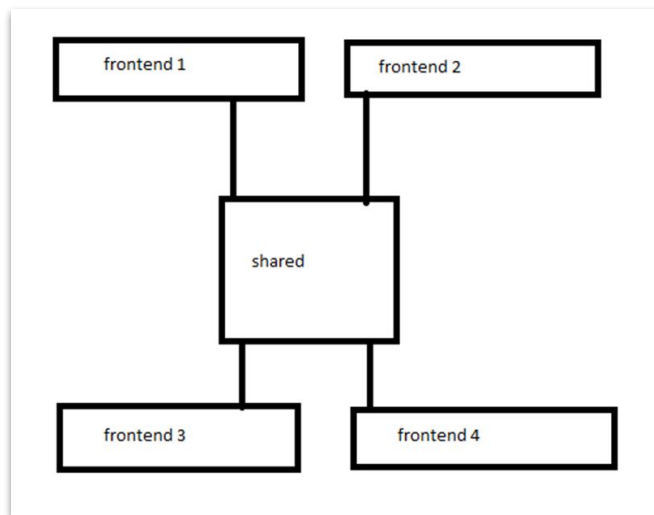


Estilismo

```
h1{  
  background-color:  blue;  
}
```

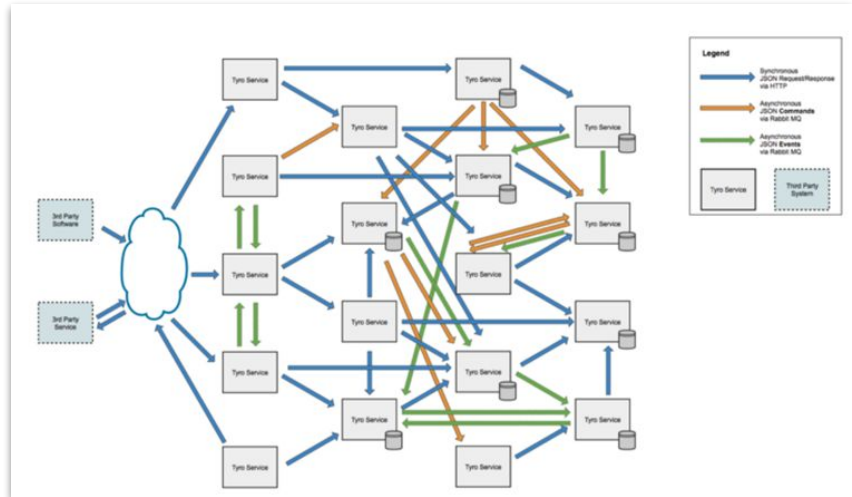
```
h1{  
  background-color:  red;  
}
```

Bibliotecas de componentes compartidos



Comunicación

- Entre los distintos frontends
- Con el backend
- preocupaciones transversales



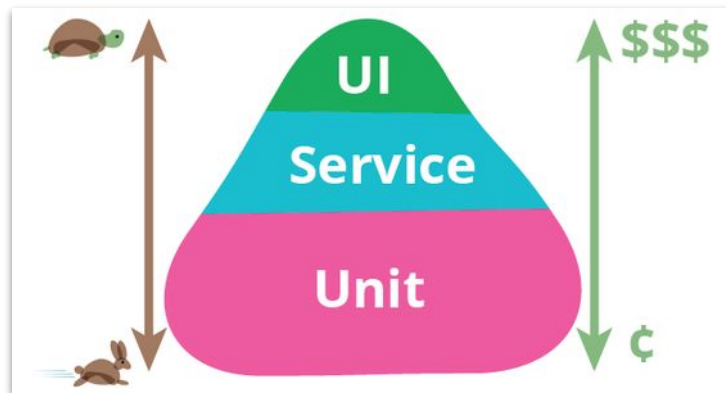


Test

- Las estrategias que se usan para probar interfaces monolíticas valen para **reproducirse en cada interfaz individual**.
- Cada micro Interfaz individual tendrá su **propio conjunto de pruebas** que garantice la calidad del código
- El problema está a la hora de **probar la integración** de las diversas micro interfaces con la aplicación contenedora

Objetivo principal: Validar la **integración** de las **interfaces** antes que la lógica interna de cada micro interfaz.

Las pruebas funcionales **solo** deben cubrir **aspectos que no se pueden probar en un nivel inferior** de la pirámide de prueba



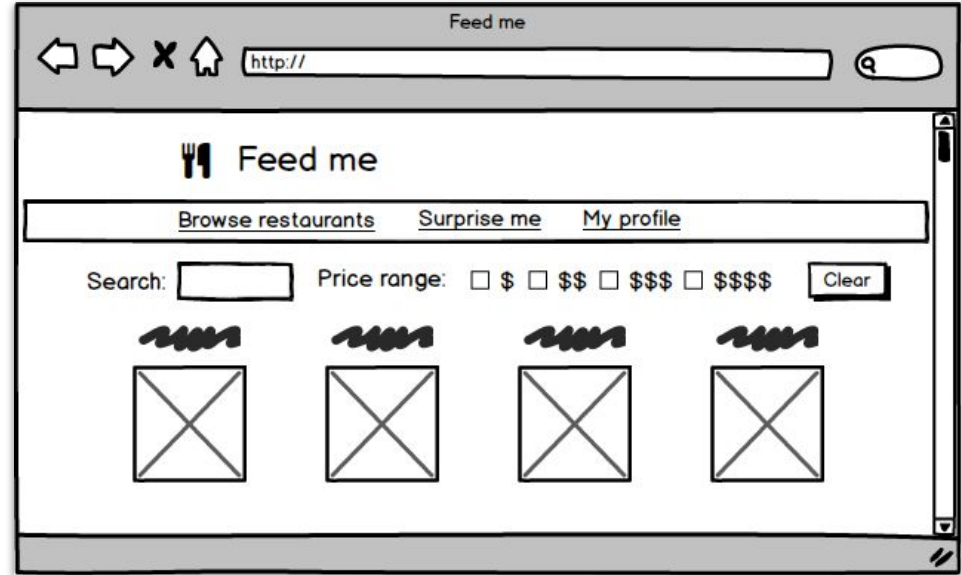
- **Pruebas unitarias** para cubrir su **lógica comercial** de bajo nivel y **lógica de representación**.
- **Pruebas funcionales** sólo para validar que la página se encuentra correctamente **ensamblada**.

Ejemplo práctico

Clientes puedan **navegar y buscar** restaurantes. Los restaurantes deben poder **buscarse y filtrarse** por distintos atributos como: precio, tipo de comida etc

Cada restaurante necesitará su propia página que **muestre su menú** y permita que el cliente pueda elegir lo que quiere comer.

El usuario de la aplicación también podrá tener su propia página de perfil, con opiniones, restaurantes visitados, sus opciones de pago etc

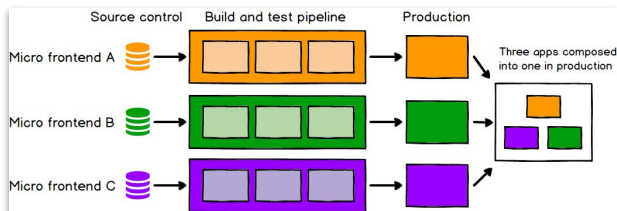


Toda esta funcionalidad perfectamente se podría **dividir en distintos equipos de trabajo** que trabajasen de manera independiente y finalmente, en conjunto hiciesen que todo funcionase.

Ventajas

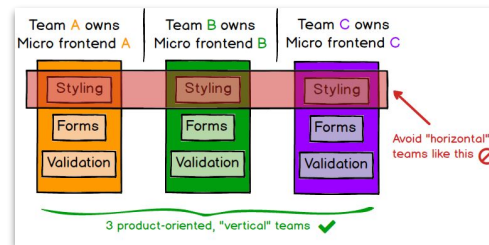
Despliegue independiente

Cada micro frontend se implementa en producción de **forma independiente**.
Formando finalmente un conjunto en el despliegue.



Equipos autónomos

Cada aplicación ha de ser propiedad de un sólo equipo. Se suele **dividir el producto** en función de lo que verán los usuarios finales.



Actualizaciones incrementales

Cada micro interfaz se puede **actualizar cuando tenga sentido**, en lugar de parar y modificar todo en conjunto.

Bases de código simples

El código fuente de cada micro interfaz individual será más pequeño que el de una única interfaz monolítica. También será más **simple y fácil de trabajar** con ellas.



Desventajas

Diferencias ambientales

Existen riesgos asociados si se desarrolla en un **entorno** que se comporte de manera **diferente al de producción**.

Tamaño de la carga útil

Los paquetes de JS creados de forma independiente pueden provocar la **duplicación de dependencias comunes**, aumentando así el número de bytes que hay que enviar en la red a los usuarios finales.

Complejidad operativa

Tener una arquitectura con micro frontends implica tener **más cosas que administrar**: más herramientas, más servidores, repositorios etc.



Conclusión

Con el paso de los años el código frontend va volviéndose más complejo y surge la necesidad de llevar a cabo arquitecturas más escalables.

Cuando se eligen micro frontends, por definición, se está optando por crear muchas cosas pequeñas en lugar de una cosa grande que trabajen en conjunto.

Deberíamos poder escalar la entrega de software a través de equipos independientes y autónomos.

Puede ser una buena opción de arquitectura, dependiendo del sistema a diseñar.



Bibliografía

Tanto información como imágenes obtenidos de las siguientes fuentes:

- Mezzalira, L. (22 de diciembre de 2019). *Micro-frontends decisions framework*.
<https://medium.com/@lucamezzalira/micro-frontends-decisions-framework-ebcd22256513>
- Jackson, C. (19 de junio de 2019). *Micro frontends*.
<https://www.martinfowler.com/articles/micro-frontends.html>



¡FIN!

¡Muchas gracias!