Arquitectura Front

Menu

- Primeros pasos
- Integración continua
- Diagrama CI/CD
- Flujo Git
- Creación y manejo de librerías
- Convención para el consumo de servicios
- Esquema para el consumo de servicios
- Esquema de seguridad
- Versionado
- Directorio por ambientes
- Estilos
- Pantallas de parametros del sistema
- Actualización de Angular

MicroFrontend

- Configuración MicroFrontend

Librerías

- Configuración librería UI Sicof
- Configuración librería de encriptación
- Configuración limpieza caché por despliegue

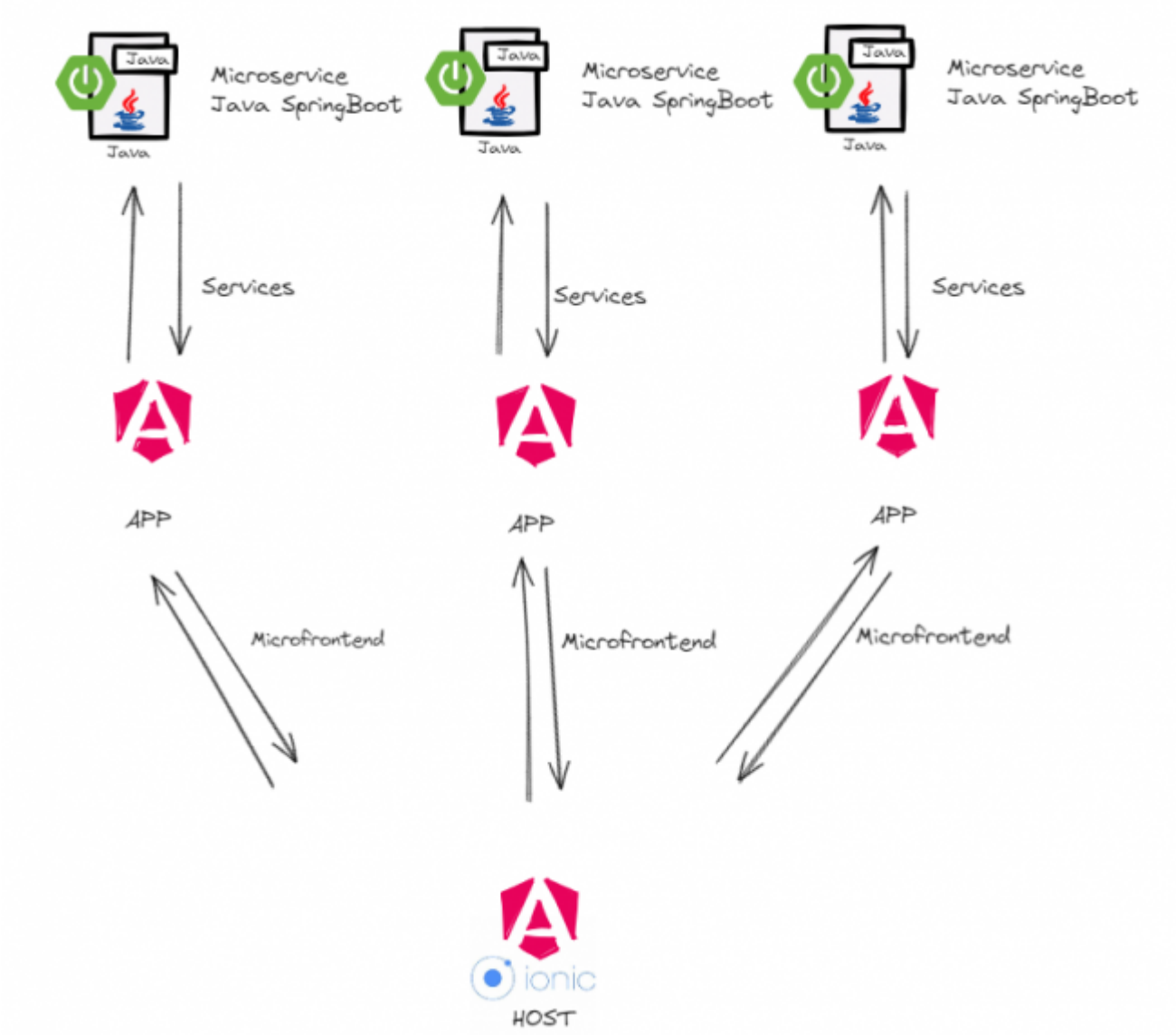
Clases Reutilizables

- Base component

Repositorios

- Convenciones para crear repositorios en GitLab
- Nombrado de ramas en git y convenciones para commits

ARQUITECTURA CLIENTE-SERVIDOR DISTRIBUIDO CERTIFICADO PARA MICROFRONTENDS



Acercad de:

1. **Estructura del Proyecto Angular**:

    1. La raíz del proyecto, tendrá la configuración de Angular CLI y otros archivos de configuración necesarios.

    1. La carpeta `src` contendrá todos los archivos fuente de la aplicación Angular.

2. **Módulos de Funcionalidad**:

    1. La aplicación está organizada en módulos de Angular que representen las principales funcionalidades o características de la aplicación. Cada módulo puede corresponder a un microfrontend potencial en el futuro.

    1. Se utiliza el enrutador de Angular para gestionar la navegación entre los diferentes módulos y componentes, sin embargo, en la aplicación se implemente SPA para la navegación por pestaña levantando un componente cada vez que se requiera.

3. **Componentes Reutilizables**:

1. Los componentes son preparados para ser reutilizables que puedan ser compartidos entre los diferentes módulos de la aplicación. Estos componentes pueden incluir elementos de interfaz de usuario comunes, como barras de navegación, barras laterales, formularios, etc.

1. Estos componentes son en su mayoría independientes y desacoplados, lo que facilitará su reutilización en diferentes partes de la aplicación y su posible migración a microfrontends en el futuro.

4. **Servicios para la Comunicación con el Servidor**:

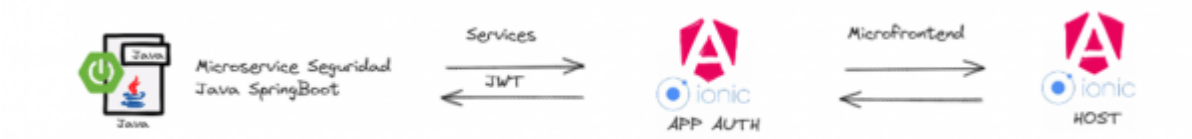
1. Se implementan consumos de servicios en Angular para la comunicación con el servidor y la gestión de datos. Estos servicios pueden encapsular las llamadas a APIs proporcionadas por los microservicios en el backend.

1. Si se decide introducir un BFF en el futuro, puedes adaptar estos servicios para comunicarse con el BFF en lugar de acceder directamente a los microservicios.

5. **Capa de Presentación y Lógica de Negocio**:

1. Utiliza los componentes de Angular para implementar la interfaz de usuario y la lógica de presentación de la aplicación.

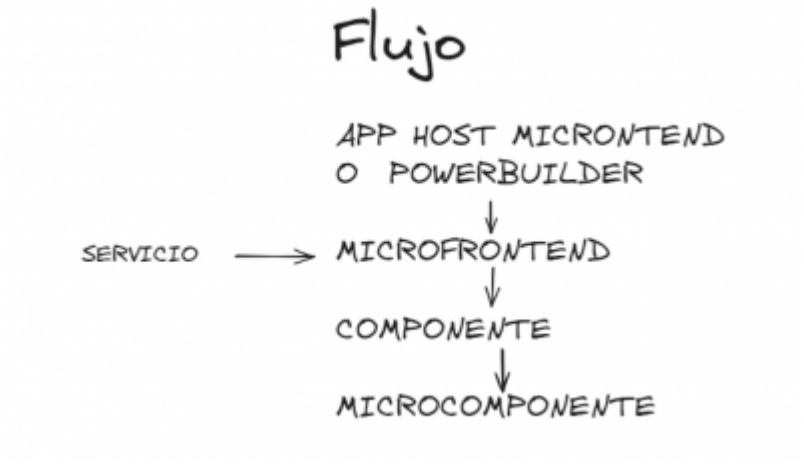
1. Mantén la lógica de negocio separada de los componentes de presentación, siguiendo los principios de separación de preocupaciones (SoC). Esto facilitará la escalabilidad y la evolución de La aplicación.



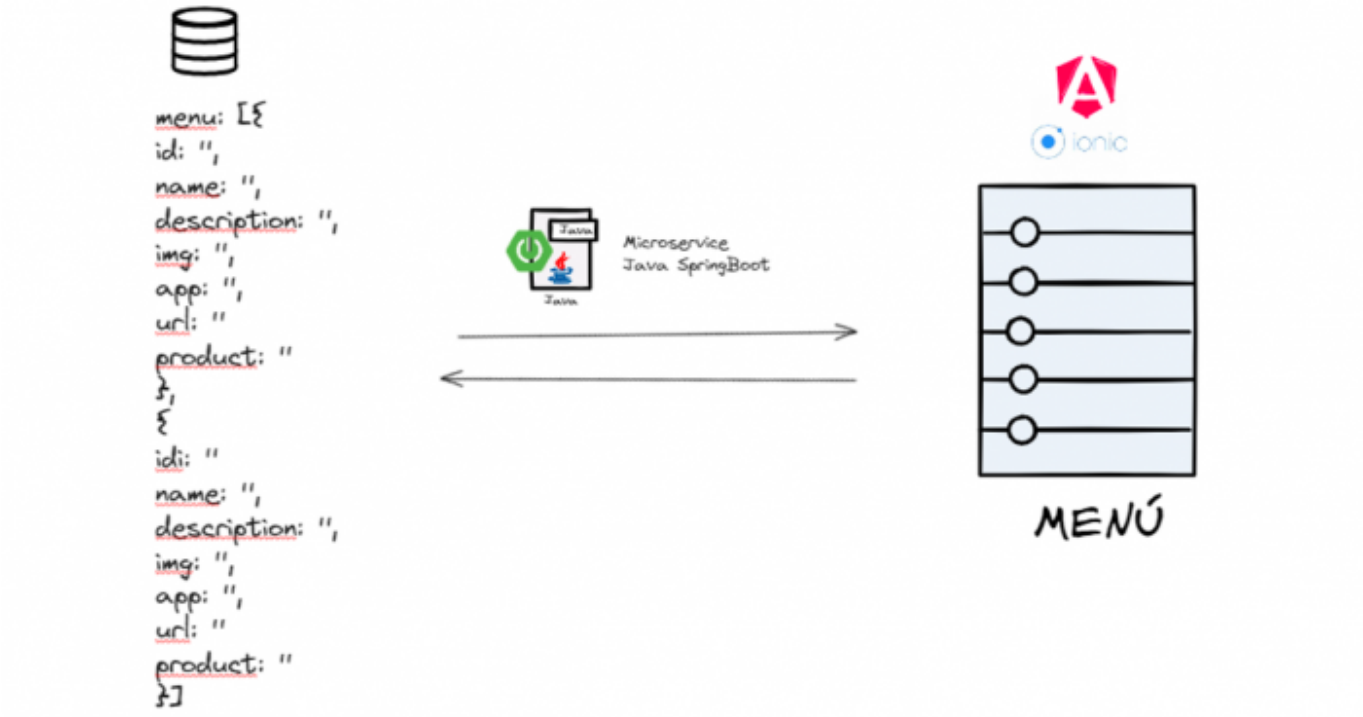
ARQUETIPO MFSICOF

```
|- public
  |- css
  |- images
  |- favicon
|- src/
  |- app/
    |- layout/
      |- components
        |- layout-dashboard
        |- layout-form
      |- layout.routes.ts
    |- pages
      |- components
      |- models
      |- services
      |- pages.routes.ts
    |- shared/
      |- components
      |- models
      |- modules
```

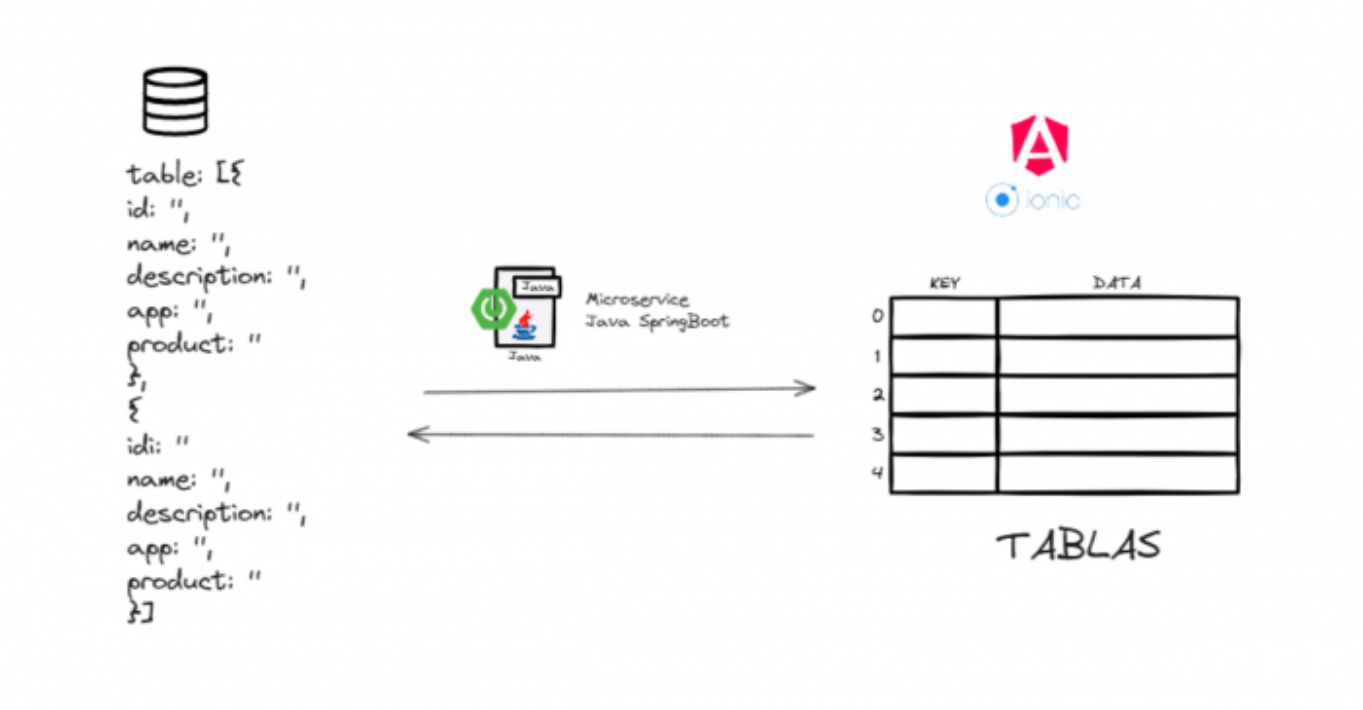
```
  |- app-component.css
  |- app-component.html
  |- app-component.ts
  |- app-config.ts
  |- app-routes.ts
|- environments/
  |- environment.dev.ts
  |- environment.prod.ts
  |- environment.qa.ts
  |- environment.ts
|- index.html
|- main.ts
|- styles.scss
```



Ejemplo de flujo entre api-componentes



Ejemplo menú



Ejemplo tablas

Arquitectura integración con Back, PowerBuilder y CI/CD

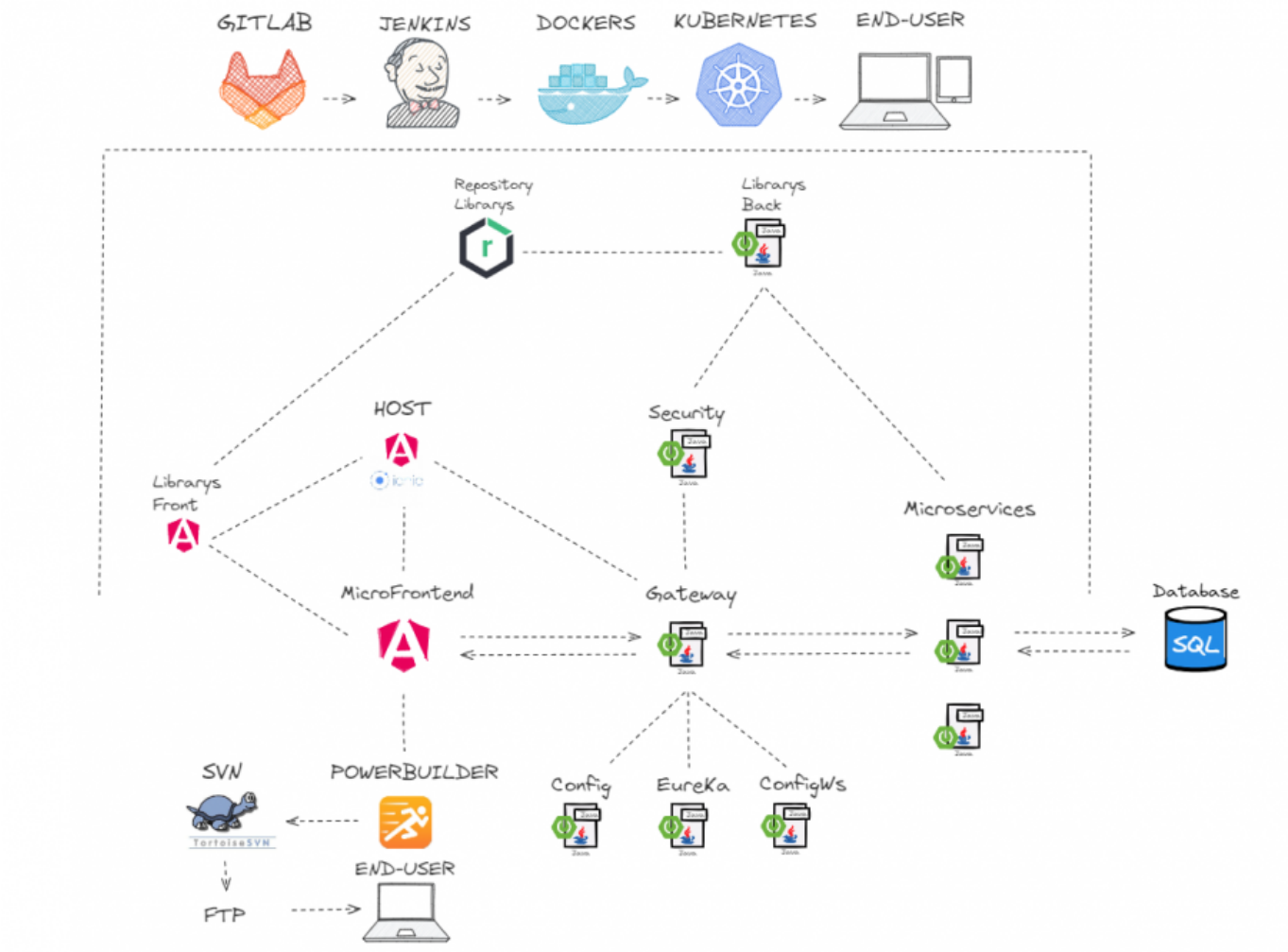
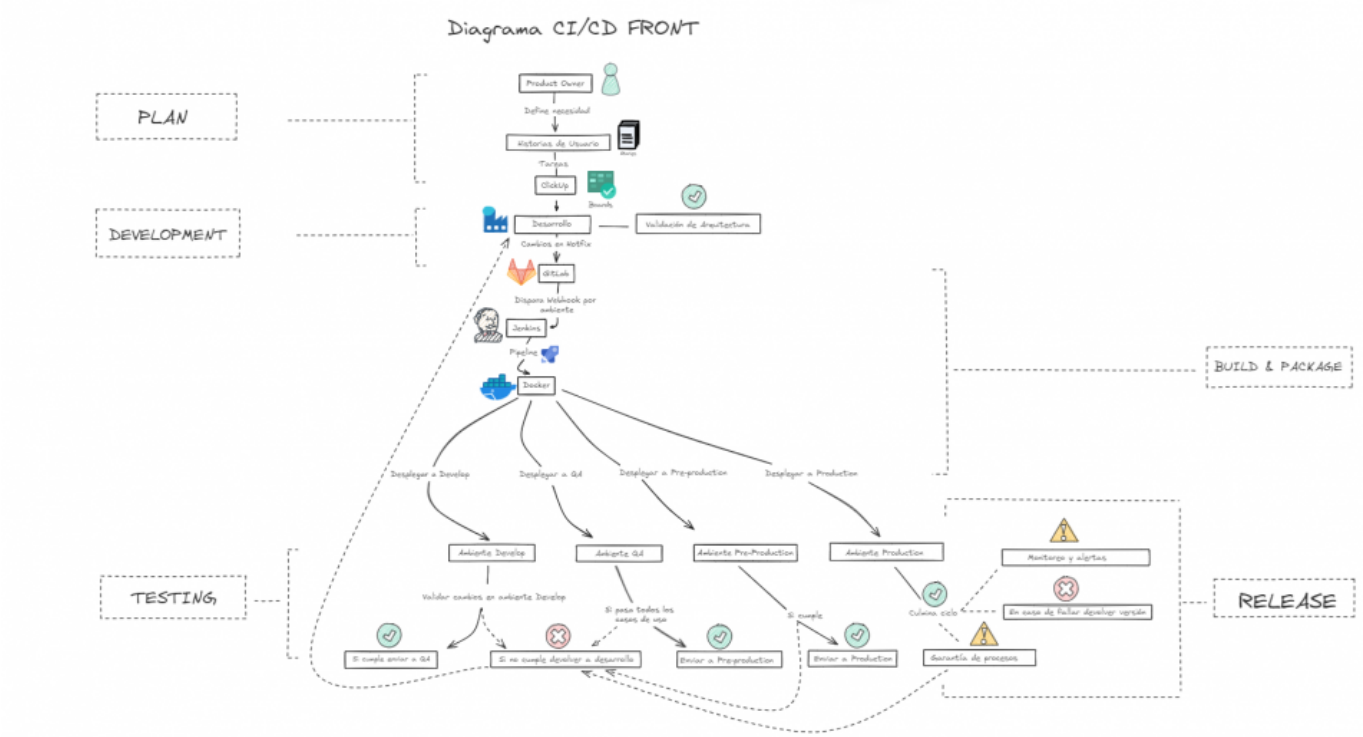


Diagrama CI/CD para despliegues automatizados bajo la arquitectura y la filosofía GIT seleccionadas.

Basado en las necesidades de la empresa y las experiencias previas, donde no se quería una filosofía de despliegue lenta que genere más trabajo a los desarrolladores y al equipo de integración, lo cual sería un cuello de botella muy grande para toda la operación, aunado a esto no se quería sacrificar calidad, por tal motivo se planteó el debate entre los Arquitectos y Senior del equipo de migración sobre cuál sería la mejor estrategia de despliegue analizando para ello filosofías como gitflow, gitlab flow y github flow. Esto dio como resultado el siguiente diagrama CI/CD, el cual fue llevado a la practica en el Front y ha resultado en una experiencia exitosa.

Para ampliar el tema se recomienda leer los siguientes artículos;

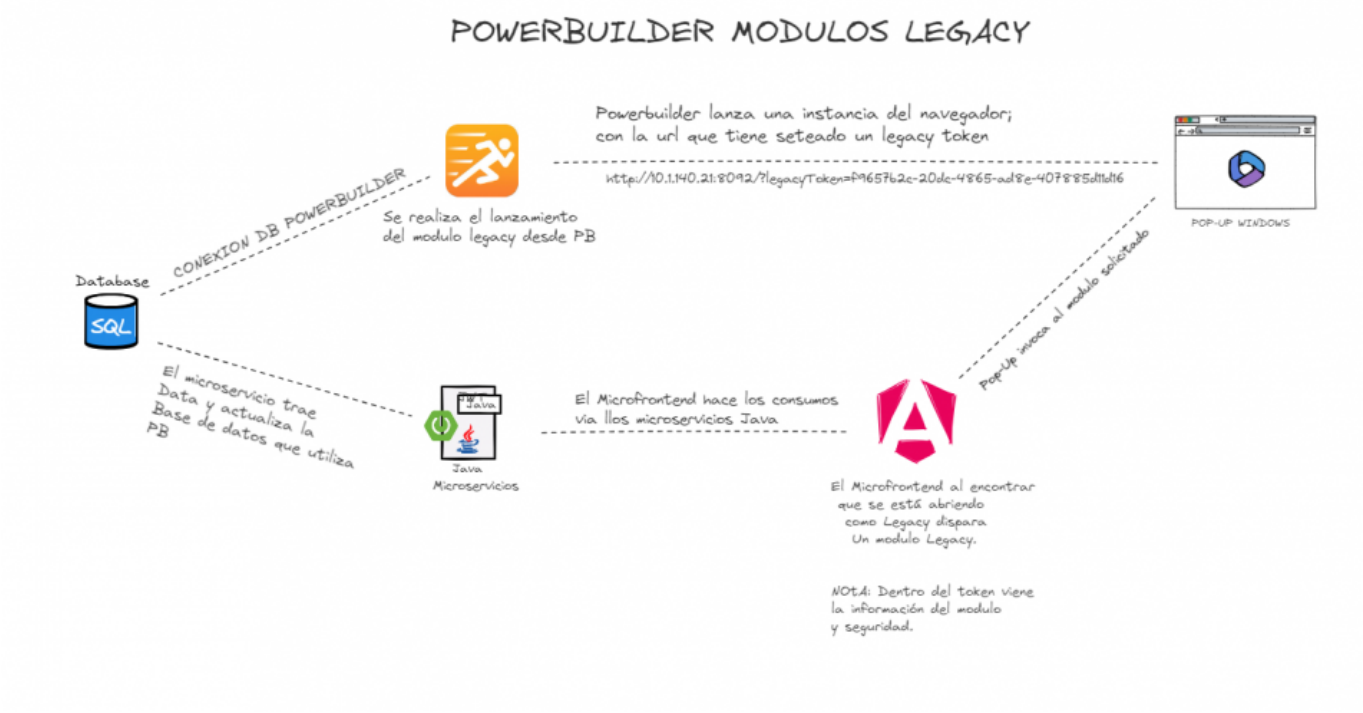
- Integración continua
- Flujo Git
- Versionado



MODULOS LEGACY POWERBUILDER

Para dar soporte a las herramientas que existen actualmente en la empresa se definió una ruta de modulos legacy, la cual irán sustituyendo los modulos actualmente existente en powerbuilder, para que esto sea posible dicha integración powerbuilder realiza lanzamientos de aplicaciones a un navegador externo con una url, esta url tiene seteado un legacytoken el cual contiene la información de seguridad y el módulo que se requiere levantar. Una vez se abra la Pop-up con el modulo el consumo y la comunicación con la base de dato la realiza vía los microservicios.

Podemos conocer un poco más de como se comporta la api-legacy de powerbuilder para Back en el siguiente articulo;
http://10.1.20.89/doku.php?id=ada:howto:sicoferp:factory:new-migracion-sicoferp:apilegacy:apipb



TECNOLOGIAS

- ANGULAR
- IONIC
- Typescript
- Prime NG
- Boopstrap
- CSS3
- HTML
- Rest
- JWT

VENTAJAS DE ARQUITECTURA CLIENTE-SERVIDOR DISTRIBUIDO CERTIFICADO PARA MICROFRONTENDS

- ESCALABLE: Permite escalar el desarrollo de manera eficiente.
- Flexible: Permite utilizar diferente frameworks y nuevas tecnologías.
- Despliegue independientes: Cada microfront se puede desplegar sin afectar los otros microfront, una app caída no afecta a la otra.
- Ideal para microservicios.
- Ideal para trabajar con externos.

CALIDAD

La calidad de software es muy importante para la migración por tal motivo se está siguiendo las siguientes normativas y prácticas;

- ISO/IEC 25010:2011; Funcional, Eficiente, Compatible, Usable, Confiable, Seguro, Mantenible y Portable.
- ISO/IEC 5055:2021; Seguro, Confiable, Eficiente y Mantenible.
- Principio Solid; Robusto, Flexible, Mantenible, Extensible. (Responsabilidad Única,

Abierto/Cerrado, Substitución de objetos, Interfaces segregadas, Inversión de dependencias)

- Clean code; Leíble, entendible, modificable y mantenible.
- POO; Modularidad, Reutilización, Mantenibilidad, Flexibilidad (Encapsulación, Abstracción, Herencia, Polimorfismo)

←Regresar