**Evidencias de prueba del reto**

En primera instancia se detalla los elementos tecnológicos que se solicitó usar y posteriormente las pruebas realizadas.

En el reto se han solicitado los siguientes puntos a tener en cuenta:

1. Arquitectura de microservicios

Las evidencias se encuentran en el repositorio de fuentes del proyecto

<https://github.com/JorgeAlamo/challenge-java-03.git>

Para microservicios construidos se ha considerado la siguiente estructura de paquetes:

* config, para configurar la seguridad y autenticación mediante Keycloak, así como la configuración de cache para el API UX según lo indicado en los requerimientos de caso.
* core/exception, para mapear las excepciones que se puedan producir al autenticar, consumir los endpoints, etc.
* entities, mapea las clases correspondientes a las tablas (SQL) o documentos (No SQL) en las bases de datos.
* handlers, para manejar los métodos en los servicios que se expondrán.
* repositories, para el acceso a datos ya sea un base de datos, otros servicios o almacenamiento en cache, según corresponda a los solicitado en el reto.
* routers, exponde los métodos de los servicios mediante REST.
* services, se maneja las validaciones y reglas indicadas como requerimiento.

Captura de pantalla de un celular

Descripción generada automáticamente

1. Programación en reactiva con WebFlux.

Las evidencias se encuentran en el repositorio de fuentes del proyecto

<https://github.com/JorgeAlamo/challenge-java-03.git>

Se ha utilizado Spring Web Flux para las implementaciones, evidencia de esto es el uso de elementos asíncronos flux y mono, manejo de los servicios mediante routerFunctions, r2dbc, reactiveMongoRepository, etc.

1. Spring Security con tokens JWT y oAuth2 con Keycloak.

Las evidencias se encuentran en el repositorio de fuentes del proyecto

<https://github.com/JorgeAlamo/challenge-java-03.git>

Las 3 APIs solicitadas (2 BS y 1 UX) se han configurado para la autenticación mediante un token oauth obtenido mediante el endpoint de keycloak, previas configuraciones realizadas en la herramienta. Adicionalmente se implementó la autenticación en el servicio de Spring Cloud Gateway.

1. Uso de BD Sql y NoSQL

Se ha utilizado una base de datos relaciona en MySQL y una base de datos no relacional en MongoDB, ambas contenerizadas.

1. Contenerización de microservicios.

Cada uno de los servicios principales, APIs (2 BS y 1 UX), config server, eureka server, gateway cuentan con un archivo Dockerfile utilizado para contenerizarlos en el desarrollo y las imágenes se encuentran en un repositorio público de DockerHub.

**ajorge2893/config-server** (servicio de Spring Cloud Config Server)

**ajorge2893/eureka-server** (servicio de Spring Cloud Eureka)

**ajorge2893/servicing-payment** (servicios de API BS Service Payment)

**ajorge2893/favorites** (servicios de API BS Favorites)

**ajorge2893/payments** (servicios de API UX Payments)

**ajorge2893/api-gateway** (servicio de Spring Cloud Gateway)

El resto de los servicios se generaron con los siguientes comandos en local y en el despliegue en AKS se encuentran en los archivos manifest en el repositorio de fuentes.

* docker run -d -p 3306:3306 --name mysql-db -e MYSQL\_DATABASE=servicedb -e MYSQL\_ROOT\_HOST=% -e MYSQL\_ROOT\_PASSWORD=root mysql/mysql-server
* docker run -d -p 9999:8080 --name keycloak-auth -e KEYCLOAK\_ADMIN=admin -e KEYCLOAK\_ADMIN\_PASSWORD=admin quay.io/keycloak/keycloak:17.0.0 start-dev
* docker run -d -p 27017:27017 --name mongo-db mongo
* docker run -d -p 6379:6379 --name redis-cache redis

1. Uso de Config Server y Eureka

Las evidencias se encuentran en el repositorio de fuentes del proyecto

<https://github.com/JorgeAlamo/challenge-java-03.git>

El servicio de config-server manejar el siguiente repositorio con los archivos de configuración para los servicios:

<https://github.com/JorgeAlamo/repo-config-server.git>

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Respecto al servicio de Eureka se ha configurado para tener como clientes a las APIs implementadas y servicio de gateway.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. Spring Cloud Gateway (opcional)

Las evidencias se encuentran en el repositorio de fuentes del proyecto

<https://github.com/JorgeAlamo/challenge-java-03.git>

Se han configurado las 3 APIs implementadas para ser consumidas a través del gateway, el cual también solicita autenticación mediante Keycloak.

1. Despliegue y orquestación en Kubernetes.

Se ha desplegado un componente AKS en Azure para realizar el despliegue, los archivos manifest correspondientes a las 3 apis, gateway, config-server, eureka-server, 2 bases de datos, redis cache y keycloak se encuentran en el repositorio con las fuentes (carpeta 07 - manisfest).

<https://github.com/JorgeAlamo/challenge-java-03.git>

**Pruebas realizadas**

He de mencionar que las validaciones indicadas en el documento corresponden al despliegue final en AKS.

1. Generación de token de autenticación con endpoint de Keycloak.

En el manual de implementación se indica los pasos para levantar en AKS un deployment y service de keycloak y exponerlo mediante una ip publica para realizar la configuración de realm, client y user necesarios para generar los tokens.

Texto

Descripción generada automáticamente

1. Consumir endpoints de las APIs BS y UX con token oauth

Se indica en la siguiente imagen la ip pública del gateway y el puerto que se utilizó para realizar las pruebas.

Imagen que contiene Calendario

Descripción generada automáticamente

Para el API BS Service Payment v1, endpoint de listado de servicios

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Para el API BS Service Payment v1, endpoint de listado de servicios por canal (canal bi: banca por internet, bm: banca móvil)

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Para el API BS Service Payment v1, endpoint de listado de transacciones

Texto

Descripción generada automáticamente

Para el API BS Favorites v1, endpoint de listado de favoritos

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Para el API UX Payments V1, endpoint de registro de transacción

Texto

Descripción generada automáticamente

1. Validaciones y pruebas de requerimientos del reto

Para API BS Service Payment v1

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Se valida que, para realizar una transacción, entre los campos a enviar se encuentran el código de servicio, número de suministro y monto a pagar.

Captura de pantalla con letras y números

Descripción generada automáticamente

Se valida que el servicio servicing-payment en AKS esta escalado a 2 instancias.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

Se valida que el API de pago de servicios cuenta con un endpoint para listar los servicios por canal.

Texto

Descripción generada automáticamente

Para API BS Favorites v1

Texto

Descripción generada automáticamente con confianza media

Se valida que, para registrar un favorito, entre los campos a enviar se encuentran nombre, tipo de favorito y código de servicio.

Captura de pantalla con letras y números

Descripción generada automáticamente

Para API UX Payments v1

Texto

Descripción generada automáticamente

Cumple con el primer requerimiento, endpoint de listado de servicios por canal (canal bi: banca por internet, bm: banca móvil). Para obtener el listado se consume desde el API UX al API BS de pago de servicios

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Se valida que el código de servicio se encuentre en la cache, caso contrario se consume desde el API UX el API BS de pago de servicios en busca del código y de no encontrarlo retorna un mensaje de error. Se cumple con el segundo requerimiento.

Captura de pantalla de un celular

Descripción generada automáticamente

En el log del API UX al registrar una transacción se realiza un llamado a redis cache con la función findAll para recuperar el listado de servicios y buscar el código, al recuperar el listado buscará el código con la función findByChannel y si no lo encuentra en cache, ejecutará el update de la cache consultando el listado al API BS de pago de servicios. Imagen que contiene exterior, tabla, grande, computadora

Descripción generada automáticamente

En el log se evidencia que en los registros resaltados, se hace el update del listado de servicios en redis cache, y en un periodo de 5 min, las consultas para listado de servicios se obtiene de la cache, posteriormente se consulta el API BS de pago de servicios y se actualiza la cache

Imagen que contiene Texto

Descripción generada automáticamente

En caso el registro de la transacción falle en el llamado al API BS de pago de servicios desde el API UX, se realizarán 2 intentos de reconexión cada 2 seg. Cumple con el 3er requisito.

Texto

Descripción generada automáticamente

En caso el registro de la transacción sea exitoso, se llama al API BS de favoritos para registrar como favorito la transacción. Se verifica en el log que posterior a ejecutar el método save de registro de la transacción, se ejecuta el método save para el registro del favorito. Cumple con el 4to requerimeinto.

Imagen que contiene exterior, tabla, grande, computadora

Descripción generada automáticamente

Texto

Descripción generada automáticamente