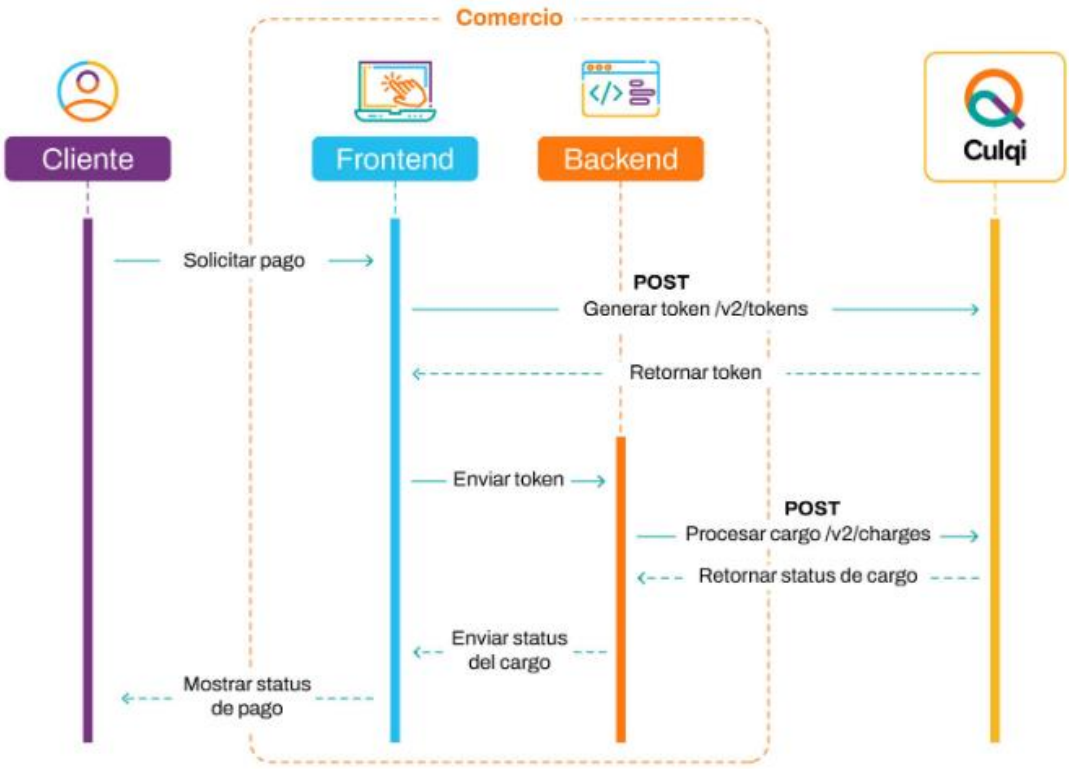
Tokenización de Tarjetas

1. Contexto

Las pasarelas de pagos guardan las tarjetas de crédito en una bóveda encriptada (encriptación en reposo) para evitar que la información sensible se pueda filtrar o que pueda ser interceptada en otro proceso del sistema.

El proceso de tokenización funciona enviando los datos de la tarjeta al tokenizador, este valida y guarda la información en la BD encriptada y devuelve un ID (token) como llave del registro el cual puede ser usado luego en los distintos procesos de culqi.

En el siguiente gráfico puedes ver donde se usa este API de tokenización en el proceso de autorización de una tarjeta de crédito.



2. Requerimiento técnico

Esta prueba técnica debe desplegarse en Lambda functions de AWS, por lo tanto, no puedes usar un framework de nodejs como express, deberá crear su propia estructura

de carpetas, aplicando patrones de diseño y siguiendo las buenas prácticas de AWS para el uso de Lambdas.

https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html

Las tecnologías a aplicar son las siguientes:

Tecnologías:

- **Backend**: TypeScript
- **BD relacional**: PostgreSql
- **BD no relacional**: Redis
- **Test**: Jest

El package.json del proyecto debe tener 2 comandos:

1. Comando para compilar TypeScript y generar el build de la aplicación que debe exponer los métodos que serán utilizados en los lambda functions.

2. Comando para ejecutar los test de la aplicación en un entorno local.

Puedes usar serverless para la configuración del lambda tanto a nivel local como en entornos posteriores.

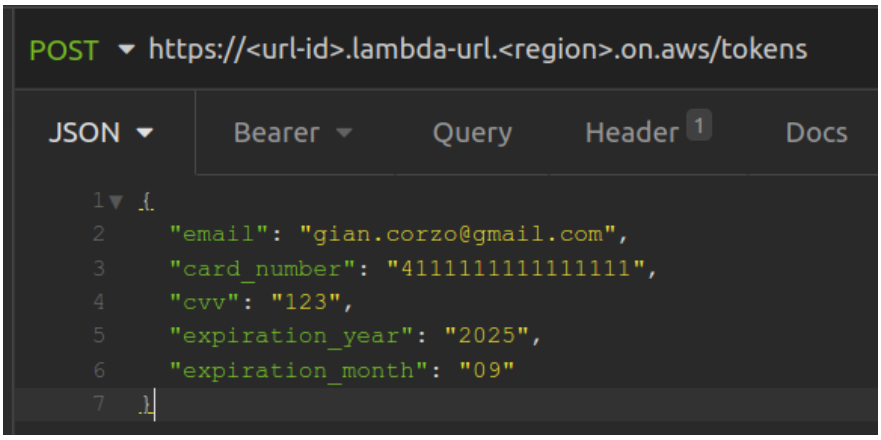
El proyecto deberá tener un README.MD con la descripción de los pasos a seguir para poder ejecutar el proyecto en un entorno local y así poder ejecutar los 2 comandos de npm.

3. Flujos de negocio

I. Creación de un token

Se debe crear un método para simular la tokenización de una tarjeta de crédito / débito, este método recibirá los siguientes parámetros.

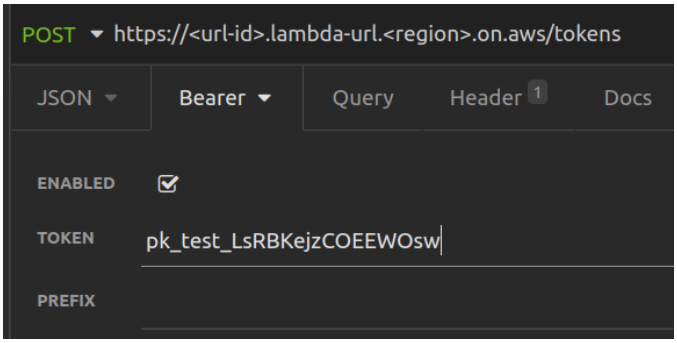
Body del request:



Parameter	Type	Restriction	Description
card_number	number	Length: 13…16	Utilizar el algoritmo de LUHN para garantizar que la tarjeta sea válida.
cvv	number	Length: 3…4	Visa / mastercard: 123 Amex: 4532
expiration_month	string	Length: 1..2	Del 1 a 12
expiration_year	string	Length: 4	Año actual máximo 5 años
email	string	Length: 5..100	Garantizar que sean email válidos utilizando los siguientes dominios “gmail.com”, “hotmail.com”, “yahoo.es”.

Si no logra pasar las validaciones, se debe interrumpir el proceso y arrojar una excepción.

Headers del request:
Las pk son llaves que permiten identificar a los comercios dentro del API, asumir por simplicidad que siempre vamos a usar la misma llave en todos los request pero el sistema debe validar la presencia de la misma y hacer las validaciones necesarias (exista, formato, etc)



Con los datos recibidos por petición deberá almacenar todos los datos de la petición más el token auto generado en una base de datos no relacional, con una expiración de 15 min (la expiración debe ser automática).

El token generado debe tener el siguiente formato (un random de 16 carácteres, donde utiliza números, letras minúsculas, letras mayúsculas y éste sea único enviado en la respuesta, ejemplo: “0ae8dW2FpEAZlxlz”)

II. Obtener datos de tarjeta

Se debe crear un método para obtener los datos de la tarjeta según los siguientes parámetros.

a. token
   i. Type: String
   ii. Length: 16

Obtener los datos de la tarjeta almacenados en el paso anterior, según el token. Este endpoint sólo debe funcionar para los PK válidos

La respuesta del método debe retornar solo los datos de la tarjeta (sin CVV) en caso el elemento ya no esté presente (porque expiró) debe retornar una respuesta coherente.

4. Consideraciones

Seguir todas las buenas prácticas (que pase el linter), el código debe ser legible y bien estructurado.