Solución:

1.- TypeScript para generar el token de pago y procesar el status del cargo

```
import { APIGatewayEvent, APIGatewayProxyResult } from 'aws-lambda';

export async function generatePaymentToken(event: APIGatewayEvent):
Promise<APIGatewayProxyResult> {
  try {
    // Aquí se procesa la solicitud de pago y se genera el token de pago
//los parámetros provienen del FrontEnd
    const paymentToken = generateToken(?,?,?,?,?);

    // Aquí se realiza el proceso de cargo o pago con el token generado

    // Preparar la respuesta
    const response = {
      token: paymentToken,
      status: 'success'
    };

    // Devolver la respuesta al frontend
    return {
      statusCode: 200,
      body: JSON.stringify(response)
    };
  } catch (error) {
    // Manejar errores y devolver una respuesta de error si es necesario
    return {
      statusCode: 500,
      body: JSON.stringify({ error: 'Error al generar el token de pago' })
    };
  }
}

export async function processPaymentStatus(event: APIGatewayEvent):
Promise<APIGatewayProxyResult> {
  try {
    // Aquí se procesa el status del cargo o pago recibido desde el frontend

    // Preparar la respuesta
    const response = {
      message: 'Payment status processed successfully'
    };
```

```
    // Devolver la respuesta al frontend
    return {
      statusCode: 200,
      body: JSON.stringify(response)
    };
  } catch (error) {
    // Manejar errores y devolver una respuesta de error si es necesario
    return {
      statusCode: 500,
      body: JSON.stringify({ error: 'Error al procesar el status del pago' })
    };
  }
}

// Función para generar un token de pago
function generateToken(
  card_number: number,
  cvv: number,
  expiration_month: string,
  expiration_year: string,
  email: string
): string {
  // Validar longitud del número de tarjeta
  const cardNumberLength = card_number.toString().length;
  if (cardNumberLength < 13 || cardNumberLength > 16) {
    throw new Error('Longitud del número de tarjeta inválida');
  }

  // Validar algoritmo de LUHN para tarjeta válida
  if (!validateLuhnAlgorithm(card_number)) {
    throw new Error('Número de tarjeta inválido');
  }

  // Validar longitud y valores CVV según tipo de tarjeta
  const cvvLength = cvv.toString().length;
  const cardType = getCardType(card_number);
  if (
    (cardType === 'visa' || cardType === 'mastercard') && (cvvLength !== 3 || cvv !== 123)
||
    (cardType === 'amex') && (cvvLength !== 4 || cvv !== 4532)
  ) {
    throw new Error('CVV inválido para el tipo de tarjeta');
  }
```

```
  // Validar mes de expiración
  const expirationMonth = parseInt(expiration_month, 10);
  if (expirationMonth < 1 || expirationMonth > 12) {
    throw new Error('Mes de expiración inválido');
  }

  // Validar año de expiración
  const currentYear = new Date().getFullYear();
  const expirationYear = parseInt(expiration_year, 10);
  if (expirationYear < currentYear || expirationYear > currentYear + 5) {
    throw new Error('Año de expiración inválido');
  }

  // Validar dirección de correo electrónico
  const validEmailDomains = ['gmail.com', 'hotmail.com', 'yahoo.es'];
  const emailDomain = email.split('@')[1];
  if (!validEmailDomains.includes(emailDomain)) {
    throw new Error('Dirección de correo electrónico inválida');
  }

  // Generar token de la tarjeta
  const token = '0ae8dW2FpEAZlxlz'; // Lógica de tokenización aquí

  return token;
}
```

2. TypeScript para obtener los datos de la tarjeta

```
interface CardData {
  card_number: string;
  expiration_month: string;
  expiration_year: string;
}

function getCardData(token: string): CardData {
  // Simulación de datos de tarjeta almacenados en alguna fuente (base de datos,
API, etc.)
  const cardDataStore = {
    'TOKEN1234567890': {
      card_number: '**********1234',
      expiration_month: '08',
      expiration_year: '2024'
```

```
    },
    'TOKEN0987654321': {
      card_number: '**********5678',
      expiration_month: '06',
      expiration_year: '2022'
    }
  };

  const cardData = cardDataStore[token];

  if (!cardData) {
    throw new Error('No se encontraron datos de tarjeta para el token
proporcionado');
  }

  const currentDate = new Date();
  const expirationDate = new Date(
    parseInt(cardData.expiration_year, 10),
    parseInt(cardData.expiration_month, 10) - 1
  );

  if (expirationDate < currentDate) {
    throw new Error('La tarjeta ha expirado');
  }

  return { ...cardData };
}
```

README DEL PROYECTO

Descripción del Proyecto

Este proyecto es una aplicación TypeScript que utiliza AWS Serverless para configurar funciones Lambda. El proyecto incluye métodos para tokenizar tarjetas de crédito/débito y obtener datos de tarjetas según un token.

Requisitos Previos

Antes de comenzar, asegúrate de tener instalado lo siguiente:

- Node.js
- npm (administrador de paquetes de Node.js)

Pasos para Ejecutar el Proyecto en un Entorno Local

1. Clona el repositorio o descarga los archivos del proyecto.
2. Abre una terminal y navega hasta el directorio del proyecto.
3. Instala las dependencias del proyecto ejecutando el siguiente comando:

   ```
   Copy code
   npm install
   ```

4. Compila el TypeScript y genera el build de la aplicación utilizando el siguiente comando:

   ```
   arduinoCopy code
   npm run build
   ```

5. Ejecuta las pruebas de la aplicación en un entorno local con el siguiente comando:

   ```
   arduinoCopy code
   npm run test
   ```

Comandos de npm

El proyecto tiene los siguientes comandos de npm configurados en el archivo package.json:

- **`build`**: Compila el TypeScript y genera el build de la aplicación en la carpeta `dist`.
- **`test`**: Ejecuta las pruebas de la aplicación en un entorno local.
- Otros comandos necesarios para configurar AWS Serverless se pueden agregar según sea necesario.

Prueba técnica - Backend Javascript

Asegúrate de ejecutar estos comandos en la terminal después de haber instalado las dependencias del proyecto.

Configuración de AWS Serverless

Para configurar y desplegar las funciones Lambda en AWS, se requiere una configuración adicional utilizando AWS Serverless. Consulta la documentación de AWS y siga los pasos necesarios para configurar y desplegar las funciones Lambda en el entorno deseado.

Este README.md proporciona una descripción general de los pasos para ejecutar el proyecto en un entorno local y utiliza los comandos de npm para compilar TypeScript y ejecutar pruebas. Asegúrate de adaptar y personalizar este archivo según las necesidades y configuraciones específicas de tu proyecto y entorno de desarrollo.