Challenge Técnico Backend (.NET)

Gracias por tu interés en sumarte al equipo. Esta prueba busca evaluar tus habilidades de backend de forma realista y concreta. ¡Éxitos!

Objetivo

Desarrollar una API RESTful que permita registrar pagos realizados por clientes, guardarlos en una base de datos y exponer algunos endpoints para obtener información relevante. Además de la funcionalidad base, deberás considerar escenarios de concurrencia, idempotencia y aplicación de buenas prácticas de diseño.

Entidades

Debés modelar y persistir las siguientes entidades:

Cliente

- Id (GUID)
- Nombre
- Email
- FechaAlta

Medio de Pago

- Id (int)
- Nombre
- Descripcion
- Code

Ejemplo: efectivo, cheque, tarjeta.

Moneda

- Id (int)
- Nombre
- Descripcion
- ISOCode

Ejemplo: pesos argentinos, pesos uruguayos, dólares.

Pago

- Id (GUID)
- ClienteId
- Monto
- FechaPago
- MedioPagoId
- MonedaId
- Source
- ExternalReference

A continuación, se detallan los endpoints que deberá exponer la API, junto con los formatos de request/response esperados.

Endpoints requeridos

Método	Endpoint	Descripción
POST	/pagos	Registra un nuevo pago
GET	/pagos	Consulta un pago por referencia

POST: /pagos

Request (ejemplo):

```
{
  "cliente": {
    "nombre": "Juan Perez",
    "email": "juan@megatlon.com.ar"
  },
  "monto": 3000000000,
  "fechaPago": "2025-07-23T14:00:00",
  "medioPagoCode": "CASH",
  "monedaISOCode": "ARS",
  "externalReference": "operacion12345",
  "source": "megatlon"
}
```

Response: id de pago en propiedad “pagoId”.

Condición de negocio: por necesidades de Megatlon, solo deben aceptarse pagos cuyo monto sea mayor o igual a 3000000000, independientemente de la moneda y el medio de pago.

GET: /pagos

Request: queryString con

- Source
- ExternalReference

Response:

- Nombre del cliente asociado al pago
- Email del cliente asociado al pago
- Monto del pago.
- Nombre del medio de pago
- Nombre de la moneda

ApiResponse:

Para los responses implementar ApiResponse con las siguientes propiedades:

- success (bool)
- message (string)
- data (object)
- errors (array): array con N motivos de error

Ejemplos:

```
{
  "success": true,
  "message": "Pago registrado correctamente",
  "data": {
    "pagoId": "e158b1e8-8911-4ffb-9174-bb60adb6e601"
  },
  "errors": null
}
```

```
{
  "success": false,
  "message": "No se pudo registrar el pago",
  "data": null,
  "errors": ["Error1, error2, error3"]
}
```

Logging

Cada vez que se realice un request a un endpoint, se debe dejar un registro en un archivo de texto en la ruta relativa logs/pagos.txt.

El archivo debe contener:

- Fecha y hora de la operación
- El request completo
- El response generado

Debe ser legible y organizado. El archivo debe generarse automáticamente si no existe.

Consideraciones

- Usar SQLite como base de datos, con Entity Framework Core y migraciones (code-first)
- Incluir el archivo app.db generado en el repositorio
- Seed mínimo: al menos tres monedas y tres medios de pago.
- Dejar una migración inicial generada.
- No hay limitaciones respecto al uso de librerías externas.
- **Idempotencia**: si llegan dos requests con el mismo Source y ExternalReference, la API debe devolver siempre el mismo resultado (no duplicar pagos).
- **Reglas de negocio configurables**: cada medio de pago debe poder tener restricciones distintas (por ejemplo: límite de monto, lista de **BINs de tarjeta**, lista de **medios de pago** permitidos/denegados, o lista de **monedas** habilitadas). Estas reglas deben poder combinarse entre sí de manera flexible.
- **Validaciones claras**: las validaciones deben devolver un mensaje de error claro para mostrar al cliente y a su vez loggear información técnica del mismo.

Nice to have (no excluyentes)

- Incluir al menos un test unitario de alguna parte de la lógica
- Documentar uno o más endpoints

Entrega

- Subí tu solución a un repositorio Git (público o privado) y compartí el link.
- Asegúrate de que el proyecto sea fácilmente ejecutable e incluya un README.md con instrucciones claras si se necesita algún paso adicional.

Tip

Buscamos claridad, funcionalidad y una solución razonada.
La estructura, las validaciones y la forma en que organices el código también son parte de la evaluación.