BankLink API

Simulación de un sistema bancario con transferencias interbancarias en ASP.NET Core 9

Agenda de Hoy

- **Introducción:** ¿Qué es BankLink y qué problema resuelve?
- **Arquitectura:** Stack tecnológico y diseño del sistema.
- **Funcionalidades:** Autenticación, cuentas y transferencias.
- **Aspectos Técnicos:** Idempotencia, Resiliencia y Transacciones.
- **Demostración en Vivo:** Flujo de la API en acción.
- **Desafíos y Soluciones:** Problemas comunes y cómo se resolvieron.
- **Conclusión:** Resumen y mejoras futuras.

¿Qué es BankLink?

Propósito

BankLink es una API REST que simula un sistema bancario completo. Permite gestionar clientes, cuentas, y realizar transferencias tanto locales como con bancos externos.

El objetivo es construir un servicio **robusto, resiliente** y **seguro**.

Tecnología Principal

- ASP.NET Core 9.0 con C#
- Entity Framework Core 9
- JWT Authentication
- ASP.NET Core Identity
- Polly (Resiliencia)
- FluentValidation

Arquitectura del Proyecto

Stack y Arquitectura

Stack Tecnológico

- ASP.NET Core 9 Web API
- Entity Framework Core 9
- JWT Authentication
- FluentValidation
- Polly (Resiliencia)
- SQL Server

Arquitectura en Capas

La aplicación sigue una arquitectura limpia donde se separan responsabilidades:

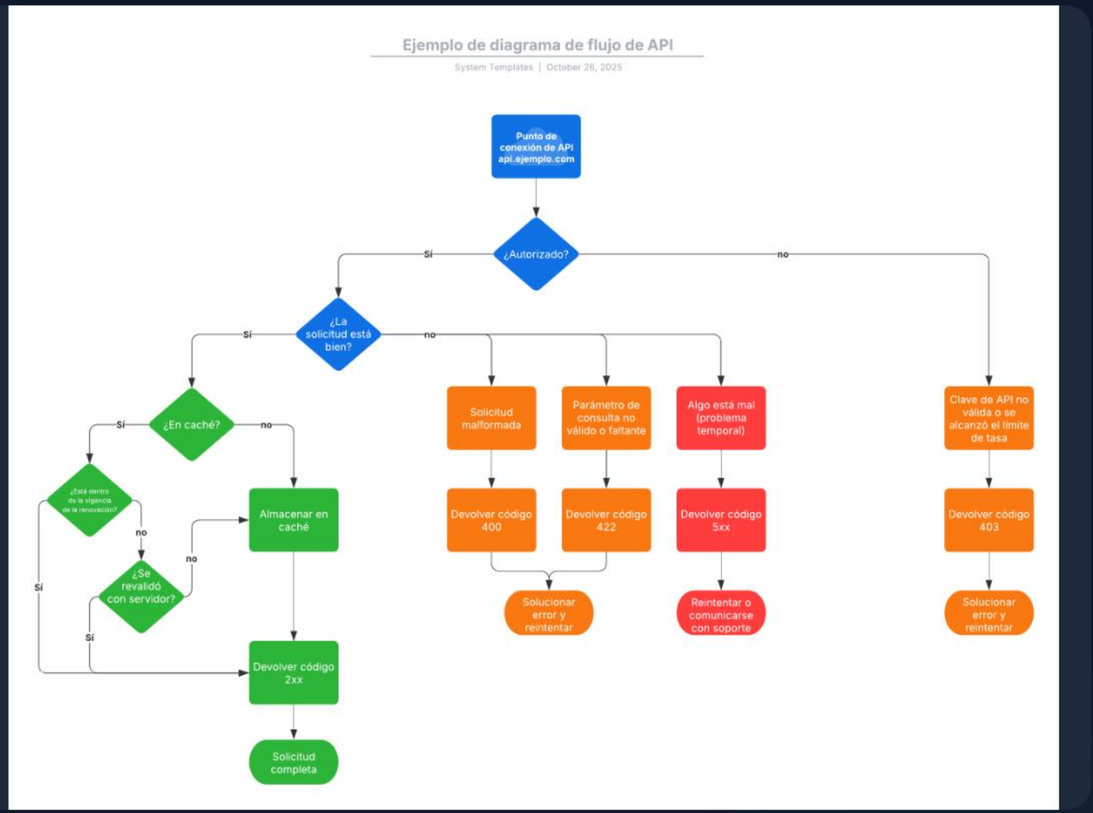
- **Controllers:** Exponen los endpoints REST.
- **Domain:** Contiene las entidades del modelo de negocio.
- **Data:** Maneja la persistencia con Entity Framework Core.
- **Services:** Encapsula la lógica de negocio e integración.
- **DTOs:** Separan la representación externa de las entidades.

Diagrama de Flujo de la API

Flujo de Petición

El flujo de una petición sigue un patrón claro y desacoplado:

1. Cliente HTTP (con Token JWT)
2. Endpoint del Controller (API REST)
3. Middleware de Validación (FluentValidation)
4. Capa de Servicio (Lógica de Negocio)
5. DbContext (Entity Framework Core)
6. Base de Datos SQL Server



Funcionalidades Principales

Autenticación y Autorización

Autenticación y Autorización

- Autenticación basada en **JWT** (JSON Web Tokens).
- Roles: **Admin** y **User**.
- Contraseñas hasheadas con **ASP.NET Core Identity**.
- Tokens JWT con expiración (60 min) y claims (ID, email, roles).
- Endpoints: `/register`, `/login`, `/me`.

```
// Program.cs - Configuración JWT
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
  .AddJwtBearer(o => {
    o.TokenValidationParameters =
      new TokenValidationParameters {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey =
          new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtKey)),
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidIssuer = jwtIssuer,
        ValidAudience = jwtAudience
    };
  });
```

Gestión de Clientes y Cuentas

Clientes y Cuentas

- CRUD completo de Clientes (con DNI único).
- Tipos de Cuentas: Ahorro o Corriente.
- Estados: Activa o Inactiva.
- Historial de movimientos por cuenta.
- Depósitos y retiros con **transacciones atómicas**.

```
// Transacción atómica en depósitos
await using var tx = await _db.Database
    .BeginTransactionAsync();

a.Balance += dto.Monto;
_db.Movements.Add(new Movement {
  AccountId = id,
  Type = MovementType.Deposito,
  Amount = dto.Monto
});

await _db.SaveChangesAsync();
await tx.CommitAsync(); // ✅ Todo o nada
```

Transferencias Interbancarias: Salientes

Flujo de Transferencia Saliente

`POST /api/Transferencias/enviar`

1. Validar **IdempotencyKey** (evita duplicados).
2. Verificar cuenta origen y saldo.
3. Verificar banco externo.
4. Iniciar **transacción** en BD.
5. Debitar monto localmente.
6. Enviar **HTTP POST** al banco externo.
7. **Si éxito:** Commit.
8. **Si falla:** Rollback.

Payload de Ejemplo

```
{
  "originAccountId": 1,
  "destinationBankCode": "BANCO_NACION",
  "destinationAccountNumber": "0110599520000001234567",
  "amount": 1500.50,
  "concept": "Pago de factura",
  "idempotencyKey": "unique-key-12345"
}
```

Transferencias Entrantes y Bancos

Transferencias Entrantes

`POST /api/Transferencias/recibir`

- Valida **X-Api-Key** del banco origen en el header.
- Verifica cuenta destino.
- Valida idempotencia.
- Acredita monto y registra el movimiento.

Registro de Bancos Externos

`POST /api/BancosExternos`

```
{
  "name": "Banco Nación",
  "code": "BANCO_NACION",
  "baseUrl": "https://api.banconacion.com",
  "transferEndpoint": "/api/transferencias/recibir",
  "apiKey": "secret-key-banco-nacion"
}
```

Aspectos Técnicos Destacados

Aspecto Técnico: Idempotencia

Idempotencia

Garantiza que una transferencia no se procese dos veces, incluso si el cliente reenvía la misma petición (ej: por un error de red).

Se usa una **IdempotencyKey** única enviada por el cliente para cada transacción.

```
// 1. Cliente envía clave única
"idempotencyKey": "TRX-2025-11-10-001"

// 2. Sistema verifica si ya existe
var idemPrevio = await _db.Movements
  .Where(m => m.IdempotencyKey == dto.IdempotencyKey)
  .FirstOrDefaultAsync();

// 3. Si existe, retorna el resultado anterior
if (idemPrevio is not null)
  return Ok(new TransferResultDto(/* ... */));
```

Aspecto Técnico: Resiliencia con Polly

Retry Policy (Reintentos)

Maneja fallos temporales (Errores 5xx, Timeout 408) al llamar a bancos externos.

Realiza **3 reintentos** automáticos con **Backoff Exponencial** (espera 2s, 4s, y 8s).

Circuit Breaker (Cortocircuito)

Después de **5 fallos consecutivos**, el circuito se "abre" y bloquea nuevas peticiones por 30 segundos.

Esto evita saturar un servicio externo que está caído.

Aspecto Técnico: Transacciones Atómicas

Transacciones Atómicas

Garantiza la **consistencia** de los datos. Si cualquier paso de la transferencia falla (ej: la llamada al banco externo), toda la operación se revierte (Rollback).

O se completa TODO, o no se completa NADA.

```
await using var tx = await _db.Database
    .BeginTransactionAsync();
try {
  // 1. Debitar cuenta local
  account.Balance -= dto.Amount;
  await _db.SaveChangesAsync();

  // 2. Llamar banco externo
  var response = await _externalBankService
    .SendTransferAsync(...);

  if (!response.Success) {
    await tx.RollbackAsync(); // ⚠️ Revertir TODO
  }

  await tx.CommitAsync(); // ✅ Confirmar TODO
} catch {
  await tx.RollbackAsync(); // ⚠️ Revertir en excepción
}
```

Aspecto Técnico: Validación Multi-Capa

1. Data Annotations

Validaciones rápidas a nivel de DTO (Required, StringLength, Range).

2. FluentValidation

Reglas más complejas y limpias, separadas de los DTOs (ej: "Monto entre 0 y 1,000,000").

3. Lógica de Negocio

Validaciones en el controlador o servicio (ej: "Saldo insuficiente", "Cuenta inactiva").

Demostración en Vivo

Se mostrará el flujo completo usando Swagger y/o Postman.

Desafíos y Soluciones

Desafíos y Soluciones (1/2)

Desafío 1: Consistencia

Problema: Debitar localmente pero que falle la llamada al banco externo. El dinero se "pierde".

Solución: Transacciones Atómicas (`BeginTransactionAsync`) con `Rollback` en caso de fallo.

Desafío 2: Duplicados

Problema: El cliente reenvía la misma transferencia por un error de red o impaciencia.

Solución: Idempotencia. Usar una `IdempotencyKey` única para identificar y rechazar transacciones duplicadas.

Desafíos y Soluciones (2/2)

Desafío 3: Bancos Caídos

Problema: El API del banco externo no responde, está lento, o da error 500.

Solución: Resiliencia con **Polly** (Políticas de Retry + Circuit Breaker) para reintentar de forma inteligente.

Desafío 4: Seguridad de Keys

Problema: Almacenar API Keys de bancos externos en texto plano en la base de datos.

Solución Futura: Usar **Azure Key Vault** o .NET Data Protection API para encriptar/gestionar secretos.

Conclusión y Mejoras Futuras

Conclusión

- BankLink es una API robusta y resiliente.
- Implementa patrones clave de sistemas distribuidos (Idempotencia, Transacciones).
- Usa ASP.NET Core 9 de forma moderna y limpia.
- Demuestra un sistema bancario funcional y seguro.

Mejoras Futuras

- Encriptar API Keys (Azure Key Vault).
- Implementar un "Mock" de banco externo para testing de integración.
- Añadir WebSockets para notificaciones en tiempo real.
- Optimizar consultas de EF Core con proyecciones.

¿Preguntas?

Gracias por su atención.

Image Sources

https://lucid.app/systemTemplates/thumb/8f8335cf-2706-4451-b24e-2e12c7a2885c/0/124/NULL/2400/true?clipToPage=false

Source: lucid.co