gRPC

✅ Serwis płatności (PaymentService) z gRPC

Opis:

Mikroserwis `PaymentService`, który przyjmuje żądania płatności od `Ordering` i zwraca status płatności (np. "Accepted", "Declined", "Pending").

Komunikacja gRPC:

`Ordering → PaymentService` wywołuje metodę `ProcessPayment`.

📄 1. `payment.proto`

```
syntax = "proto3";

option csharp_namespace = "PaymentService.Grpc";

package payments;

// Status płatności
enum PaymentStatus {
    PAYMENT_STATUS_UNSPECIFIED = 0;
    ACCEPTED = 1;
    DECLINED = 2;
}

// Żądanie płatności
message PaymentRequest {
    string order_id = 1;
    string user_id = 2;
    double amount = 3;
}

// Odpowiedź płatności
message PaymentResponse {
    PaymentStatus status = 1;
    string reason = 2;
}

// Serwis płatności
```

```
service Payment {
    rpc Process(PaymentRequest) returns (PaymentResponse);
}
```

Korzyści:

- Szybka, binarna komunikacja.
- Kontrakt jasno zdefiniowany w `.proto`.
- Możliwość rozwoju (np. strumieniowanie do monitoringu transakcji).

✅ 3. Użycie w implementacji serwera:

```
public class PaymentServiceImplementation : Payment.PaymentBase
{
    public override Task<PaymentResponse> Process(PaymentRequest request,
ServerCallContext context)
    {
        var status = request.Amount < 1000 ? PaymentStatus.Accepted :
PaymentStatus.Declined;

        return Task.FromResult(new PaymentResponse
        {
            Status = status,
            Reason = status == PaymentStatus.Declined ? "Limit exceeded" : ""
        });
    }
}
```

✅ Scenariusz: **Strumieniowe przetwarzanie płatności**

🔸 Serwis: `PaymentService`

Będzie przyjmować żądania płatności i w czasie rzeczywistym wysyłać statusy etapu przetwarzania (np. „Processing”, „Verification”, „Accepted”).

📄 `payment.proto`

```
syntax = "proto3";

option csharp_namespace = "Payments.Grpc";
```

```
package payments;

service PaymentService {
  rpc ProcessPayment (PaymentRequest) returns (stream PaymentStatusResponse);
}

// Żądanie płatności
message PaymentRequest {
  string orderId = 1;
  double amount = 2;
  string paymentMethod = 3;
}

// Enum opisujący możliwe statusy
enum PaymentStage {
  STAGE_UNSPECIFIED = 0;
  PROCESSING = 1;
  VERIFICATION = 2;
  ACCEPTED = 3;
  DECLINED = 4;
}

// Odpowiedź streamowana przez serwis
message PaymentStatusResponse {
  PaymentStage stage = 1;
  string description = 2;
  string transactionId = 3;
}
```

1. `PaymentServiceImpl` (implementacja serwera)

```
using Grpc.Core;
using Payments.Grpc;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

public class PaymentServiceImpl : PaymentService.PaymentServiceBase
{
    // Główna metoda ProcessPayment – streamuje status płatności
    public override async Task ProcessPayment(PaymentRequest request,
IServerStreamWriter<PaymentStatusResponse> responseStream, ServerCallContext
context)
```

```
    {
        var transactionId = Guid.NewGuid().ToString();

        // Etap 1: Przetwarzanie
        await responseStream.WriteAsync(new PaymentStatusResponse
        {
            Stage = PaymentStage.Processing,
            Description = "Processing your payment.",
            TransactionId = transactionId
        });

        // Krótkie opóźnienie
        await Task.Delay(1000);

        // Etap 2: Weryfikacja
        await responseStream.WriteAsync(new PaymentStatusResponse
        {
            Stage = PaymentStage.Verification,
            Description = "Verifying payment method.",
            TransactionId = transactionId
        });

        // Krótkie opóźnienie
        await Task.Delay(1000);

        // Etap 3: Akceptacja / Odrzucenie
        var stage = request.Amount > 1000 ? PaymentStage.Declined :
PaymentStage.Accepted;
        var statusMessage = stage == PaymentStage.Accepted ? "Payment
accepted!" : "Payment declined due to limit.";

        await responseStream.WriteAsync(new PaymentStatusResponse
        {
            Stage = stage,
            Description = statusMessage,
            TransactionId = transactionId
        });
    }
}
```

🧠 Dlaczego streaming?

Dzięki `stream PaymentStatus`, klient (np. `Ordering`) otrzymuje kolejne etapy przetwarzania płatności w czasie rzeczywistym, bez potrzeby pollingowania.

🧩 Implementacja C#

🔹 Serwer (PaymentService)

```
public class PaymentService : payments.PaymentService.PaymentServiceBase
{
    public override async Task ProcessPayment(PaymentRequest request,
IServerStreamWriter<PaymentStatus> responseStream, ServerCallContext context)
    {
        string transactionId = Guid.NewGuid().ToString();

        await responseStream.WriteAsync(new PaymentStatus
        {
            Stage = "Processing",
            Description = "Rozpoczynanie przetwarzania płatności",
            TransactionId = transactionId
        });

        await Task.Delay(1000);
        await responseStream.WriteAsync(new PaymentStatus
        {
            Stage = "Verification",
            Description = "Weryfikacja 3D Secure w toku",
            TransactionId = transactionId
        });

        await Task.Delay(1500);
        await responseStream.WriteAsync(new PaymentStatus
        {
            Stage = "Accepted",
            Description = "Płatność zaakceptowana",
            TransactionId = transactionId
        });
    }
}
```

🔹 Klient (Ordering)

```
var channel = GrpcChannel.ForAddress("https://localhost:5001");
var client = new PaymentService.PaymentServiceClient(channel);

var paymentRequest = new PaymentRequest
{
```

```
    OrderId = "ORD123",
    Amount = 199.99,
    PaymentMethod = "Card"
};

using var call = client.ProcessPayment(paymentRequest);

await foreach (var status in call.ResponseStream.ReadAllAsync())
{
    Console.WriteLine($"Etap: {status.Stage} | Opis: {status.Description}");
}
```