CKO Payment Gateway assignment Documentation

Solution summary

- The application is built using
  - C# and Asp.NET Core Framework targeting .NET 6 SDK.
  - Visual Studio 2022.
- The application exposes two endpoints
  - POST “api/payments” to submit new payment request.
  - GET “api/Payments/{payment-id}“ to retrieve previous payment details.

Solution Structure

- CKO.PaymentGateway.Api : Asp.NET Core project and exposes payment endpoints (it is the startup project).
- CKO.PaymentGateway.Application : represents application layer. It contains requests and their handlers.
- CKO.PaymentGateway.BankSimulator : represents fake implementation to “Acquiring Bank” using in-memory data list.
- CKO.PaymentGateway.Persistence : provides in-memory Repository implementation with required data operations.
- CKO.PaymentGateway.Domain: stores definitions for domain entities (payment & merchant).
- CKO.PaymentGateway.Application.Tests : unit-test project to application handlers
  - It uses XUnit , Moq and AutoFixture Nugets.
- CKO.PaymentGateway.Api.IntegrationTests: integration-test project for API project
  - It uses XUnit , Moq and AutoFixture Nugets.

Assumptions

- The application uses in-memory data store.
  - When starting the application, payment data store is empty.
  - Inserted Payments records are stored in memory as a List.
- The incoming payment requests are validated against internal hardcoded list of merchants and cards’ details.
  - List of accepted cards are defined at “<solution-root>\CKO.PaymentGateway.BankSimulator\Factories\FakePaymentProcessingServiceFactory.cs”.
  - List of accepted Merchants are defined at “<solution-root>\CKO.PaymentGateway.Persistence\Factories\MerchantRepositoryFactory.cs”.

- To test “successful scenario”, please do the following in single session
    - For all requests, please specify following header
        - “Content-Type” : “application/json”.
    - Submit valid POST request.
    - Retrieve payment id from response body.
    - Submit valid GET request using payment id (retrieved in previous step).
- No authentication required.

How to run the application

- Run pre-requisites
    - The .NET 6 SDK should be installed on target machine.
    - The target machine should be connected to the internet so nuget packages can be restored successfully.
- Method 1: using Visual studio 2022
    - Open the solution in Visual Studio.
    - Assure that “CKO.PaymentGateway.Api” project is selected as startup project.
    - Press CTRL+F5.
- Method 2: using terminal
    - Navigate to solution root folder.
    - Run following command “dotnet run --project .\CKO.PaymentGateway.Api\CKO.PaymentGateway.Api.csproj”

Areas of improvement

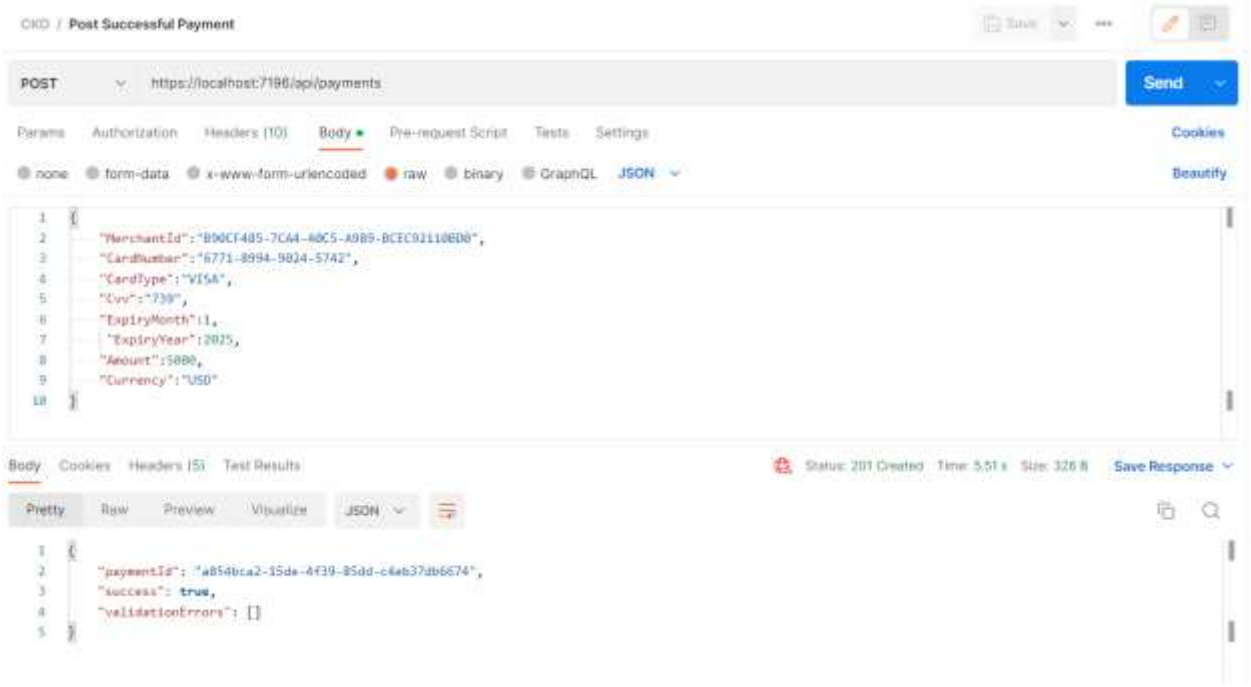
- Using data store.
- In domain
    - In “Payment” entity, Currency can be defined as enum.
- In application
    - In “CreatePaymentCommandValidator.cs” – submitted Currency value should be validated against list of supported currencies.
    - Supported list of currencies can be retrieved from external data store.
- API
    - Adding authentication.
    - Assuring that only authenticated merchants can retrieve their own payments data.
    - Adding api versioning

    - Configure Rate limiting using middleware or delegate this functionality to API gateway.
    - Use Redis for Caching to store supported card types or supported currencies.
    - Adding Swagger in non-production environments.
    - Deploy API as Docker container.
- Bank Simulator
    - It can be replaced with “client” which communicates with external API and support retries.
    - This “client” normally is developed as Nuget to be re-used easily from multiple projects.
- Testing
    - Adding unit-test projects for other existing projects.
    - Write missing unit & integration tests.

Cloud technologies

- Using data store which is scalable (supports horizontal scaling and sharding) like AWS Dynamo DB or Azure Cosmos DB.
- Logging to Azure Application Insights or AWS CloudWatch.
    - To enhance team ability to analyze logs, detect issues’ root cause.
    - Monitor the system and give team ability to setup alerts when fatal issues happen.
- Using API Gateway like Azure APIM or AWS API gateway
    - This will help in enforcing security, API monitoring, apply caching policies, apply API throttling, ability to change internal system APIs without affecting System clients.
- Deploy API as Azure functions or AWS lambda to enhance system scalability.
- Using “infrastructure as code” to automate creating system infrastructure like Terraform or Pulumi
- Using Redis or AWS Memory-DB for caching .
- Communication flow with external “Bank” can be modified
    - When the system receives payment request from a merchant
        - System replies with acknowledgement to merchant and put a message in a queue (using Azure service bus or AWS SQS)
        - System has a dedicated Background service(s) to process the messages and communicate with external bank system (backend services can be defined as azure function or AWS lambda)
        - Background service(s) update data store with payment operation results.
        - System defines an endpoint to be used by merchant to enquire about payment status.

Example of submitting successful Payment Operation



Example of successful payment GET operation