

Payment Gateway

Background

E-Commerce is experiencing exponential growth and merchants who sell their goods or services online need an easier way to collect money from their customers.

Processing a payment online involves multiple steps and entities:



Entity	Description
Shopper	Individual who is buying the product online
Merchant	The seller of the product. (Seller ex: Apple, Amazon etc.)
Payment Gateway	Responsible for validating requests, storing card information, forwarding payment requests and accepting payment responses to and from the acquiring bank
Acquiring Bank	Responsible for validating the card information and send the payment details to the appropriate 3 rd party organization for processing

Business Case

We are building a payment gateway, an API based application that allows a merchant to offer a way for their shoppers to pay for their product.

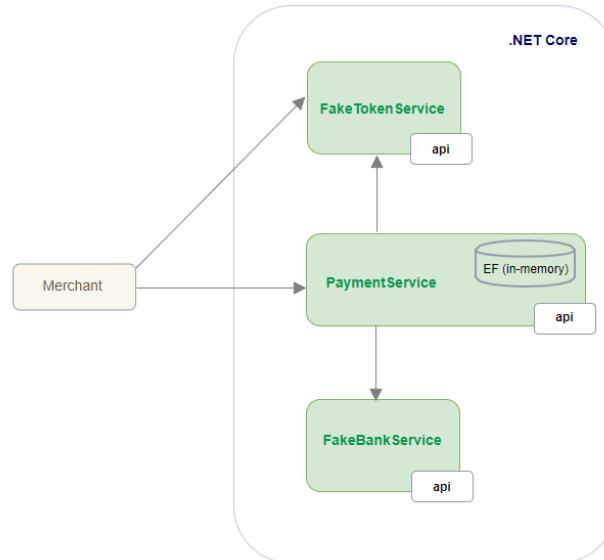
The acquiring bank component is simulate in order to allow the execution of a full test of the payment flow.

The *product requirements* for this initial phase are the following:

- **Process a payment** - A merchant should be able to process a payment through the payment gateway and receive either a successful or an unsuccessful response
- **Retrieve a payment's details** - A merchant should be able to retrieve the details of a previously made payment

Solution Architecture

Architectural diagram of the system:



The system consists of:

FakeTokenService - In the current context, this service is a mock responsible for generate a token and validate the payment requests.

FakeBankService – In the current context, this service is a mock responsible for accepting or declining a payment process request.

PaymentService - Main service in the system. It is responsible for processing payments and retrieving information about payments.

PaymentService details:

PaymentService is a .Net Core Web API project.

PaymentService uses an in-memory Entity Framework Core database.

Authentication is implemented using a token validation middleware.

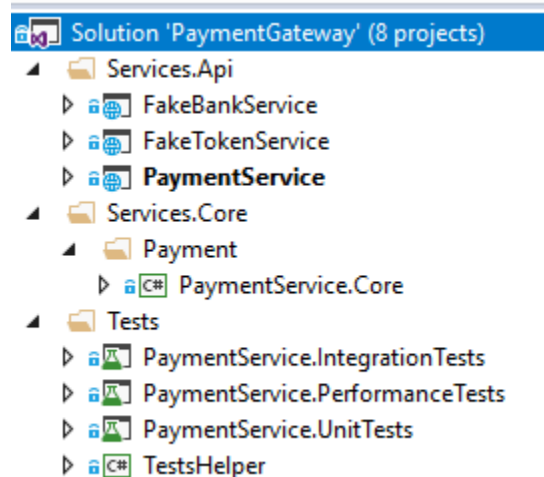
Logging mechanism is implemented using Serilog (Sinks.RollingFile). In the current context, Serilog is configured (check appsettings.json file) to create logs file in Logs folder (inside project directory).

Application metrics mechanism is implemented using Azure App Insights (check appsettings.json for InstrumenationKey)

The solution is hosting in Azure App Services.

Solution Structure

Below is a snapshot of the solution open in [Visual Studio](#) (2017 Professional Edition) IDE.



Folder: Services.Api

Contains three ASP.NET Core Web API projects defined based on the following responsibilities:

Project	Responsibilities
FakeTokenService	Simulate: <ul style="list-style-type: none">○ Generate token request○ Validate token request
PaymentService	<ul style="list-style-type: none">○ Process payment○ Retrieve payment's details
FakeBankService	Simulate: <ul style="list-style-type: none">○ Payment process response

Folder: Services.Core

Contains most of the business logic for implementing payment service API. At this initial phase the other two simulated entities (Acquiring Bank and Token Provider Service – which will be discussed later) do not need additional business logic projects.

Folder: Tests

Contains unit tests, integration tests and performance tests for payment service. There is also a helper project (TestsHelper) used for define settings needed in integration and performance tests projects.





Assumptions

- Project use an Entity Framework Core in-memory database

Note: It can be switch to an Azure Database or an on-prime SQL Server Database

- FakeTokenService will return a hard-coded token value defined in appsettings.json file
- Generated token by the FakeTokenService will be used in all requests to PaymentService in order to be consider initiated by an authenticated user
- PaymentService will apply a simple validation for request:
 - Card number should contains between 8 and 19 digits
 - CCV should contains 3 digits
 - ExpiryYear should be between 1 and 12*Note: proper validation according to a standardization (ex ISO/IEC 7812)*
- FakeBankService will not be called with a token
- FakeBankService will return the following statuses:
 - “Accepted” – if a payment process request is initialized with an “amount” greater than 0
 - “Declined” – if a payment process request is initialized with an “amount” equals to 0

Prerequisites

-  Azure Portal (in case of Azure hosting approach)
-  Visual Studio 2017
-  Net Core 2.1
-  Postman

Testing

The project is currently hosted in Azure App Services:

PaymentService: <https://mp-payment-gateway-payment-service.azurewebsites.net>

FakeTokenService: <https://mp-payment-gateway-fake-token-service.azurewebsites.net>

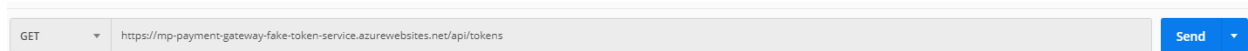
FakeBankService: <https://mp-payment-gateway-fake-bank-service.azurewebsites.net>

Testing flow-using POSTMAN:

1. Send a token generation request

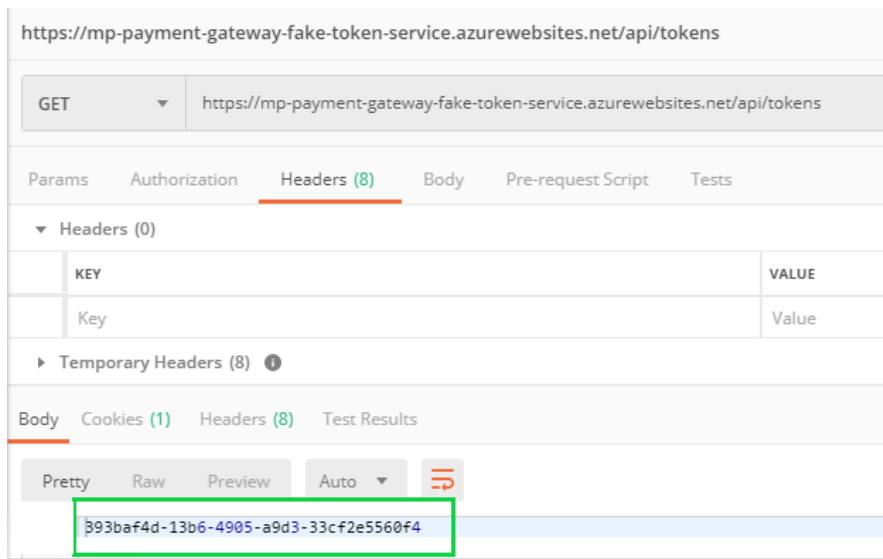
Request:

Get: <https://mp-payment-gateway-fake-token-service.azurewebsites.net/api/tokens>



Response: 393baf4d-13b6-4905-a9d3-33cf2e5560f4

Generated token:



2. Keep generated token – it will be used in payment process flow

Generated token: 393baf4d-13b6-4905-a9d3-33cf2e5560f4 will be used in requests headers.

3. Send a process payment request

Request:

POST: https://mp-payment-gateway-payment-service.azurewebsites.net/api/payments/

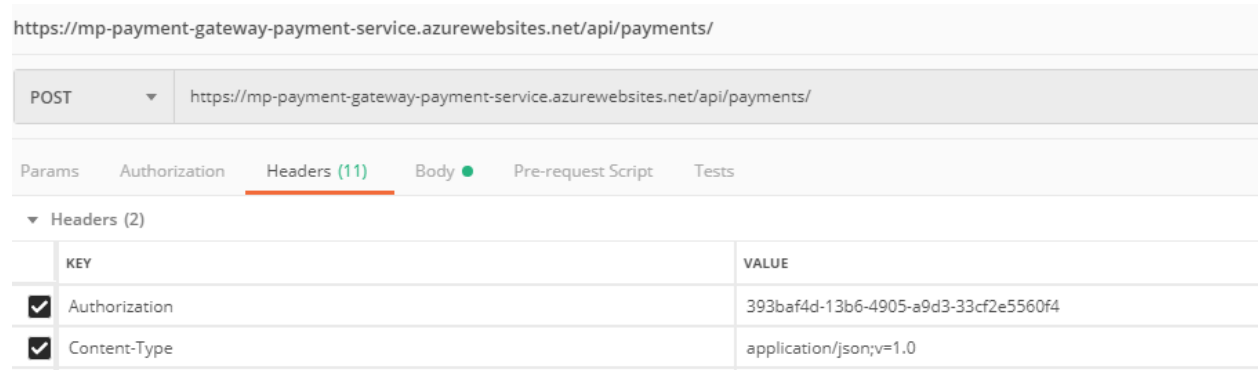
Headers:

Authorization: {generated_token}

Content-type: {application/json;v=1.0}

Note: v=1.0 could be missing (By default the PaymentService API will consider version 1.0 in this case)

For future development, a version can be specified based on requirements and API implementation.



Response:

```
{
  "paymentId": "3b1f679c-ad18-4224-9ff2-ea926bf9e70a"
  "paymentStatus": "Accepted"
}
```

4. Keep generated paymentId – it will be used for retrieving payment’s details

Generated paymentId: 3b1f679c-ad18-4224-9ff2-ea926bf9e70a will be used for retrieving payment’s details.

5. Send a retrieve payment’s details request

Request:

Get https://mp-payment-gateway-payment-service.azurewebsites.net/api/payments/{generated_payment_id}

Note: current test scenario: <https://mp-payment-gateway-payment-service.azurewebsites.net/api/payments/3b1f679c-ad18-4224-9ff2-ea926bf9e70a>

Headers:

Authorization: {generated_token}

Content-type: {application/json}

<https://mp-payment-gateway-payment-service.azurewebsites.net/api/payments/3b1f679c-ad18-4224-9ff2-ea926bf9e70a>

GET <https://mp-payment-gateway-payment-service.azurewebsites.net/api/payments/3b1f679c-ad18-4224-9ff2-ea926bf9e70a> Send

Params Authorization **Headers (10)** Body Pre-request Script Tests Cookies Code

▼ Headers (2)

KEY	VALUE	DESCRIPTION	*** Bulk Edit
<input checked="" type="checkbox"/> Authorization	393baf4d-13b6-4905-a9d3-33cf2e5560f4		
<input checked="" type="checkbox"/> Content-Type	application/json		

Response:

```
{
  "maskedCardNumber": "*****5678",
  "ccv": "077",
  "expiryYear": 2021,
  "expiryMonth": 11,
  "paymentStatus": "Accepted"
}
```

The payment’s details contains masked card number, card details and paymentStatus (same value with paymentStatus received at step 3).

Testing flow-using another Azure account or locally

For hosting and testing the app using your Azure account additional steps need to be executed (create Azure resource group / use existing resource group, publish the services in Azure (PaymentService, FakeTokenService, FakeBankService) and change the following settings from appsettings.json file:

```
"TokensProvider": "https://mp-payment-gateway-fake-token-service.azurewebsites.net/",  
"PaymentsProcessor": "https://mp-payment-gateway-fake-bank-service.azurewebsites.net/",
```

For testing the app locally, the same settings (TokensProvider, PaymentsProcessor) from appsettings.json file will be changed with values (application urls) from launchSettings.json files (for each service).

Make sure the ports used are not blocked (Windows Firewall) so the connection to services will not be refused.

The flow can be tested using PaymentService.IntegrationTests project with the above changes executed and the PaymentService address should be configured in Configuration class from TestHelper project.

Improvements / Alternatives

Architecture	Add an API Gateway layer – based on microservices architecture and expose the services endpoints
Security	Implement JWT security
Database	Use Azure Databases instead of in-memory database
Logging	Use Serilog Sinks AzureTableStorage inside of local RollingFile
Code section	Add an interface for IApiClient that can be implemented by any type of rest client without breaking the code Add strong payment request validation Increase code coverage (for proper performance tests could be used Azure Test Plans)
CD/CI	Use Docker for containerization