Bhuvan Perakam

Bhuvan.perakam@gmail.com

Abstract

This document explains the API of Payment Gateway, Bank Simulator, Sequence Diagram and improvements of the Tasks

Payment Gateway documentation

Home Task of Checkout.Com

Contents

[Payment Gateway – API Documentation 1](#_Toc122010054)

[Bank Simulator – API Documentation 2](#_Toc122010055)

[Sequence Diagram 3](#_Toc122010056)

[Projects 4](#_Toc122010057)

[What are highlights of the Project 4](#_Toc122010058)

[What can be improved or implemented in the project 5](#_Toc122010059)

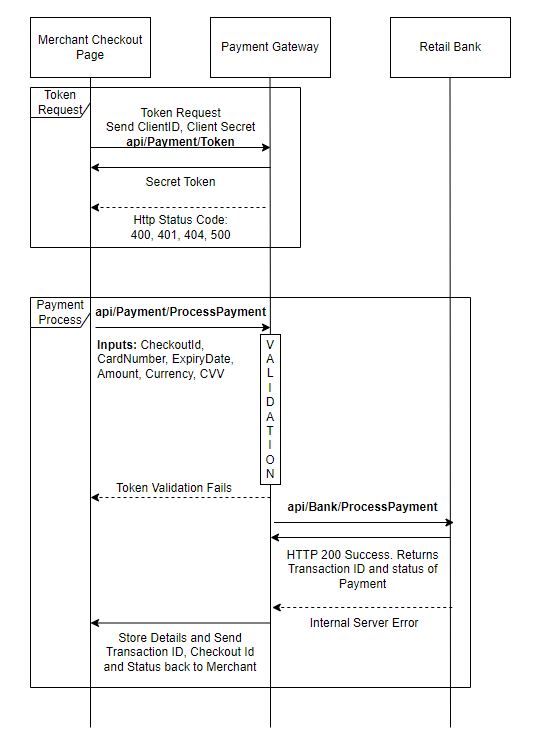
# Payment Gateway – API Documentation

|  |  |  |
| --- | --- | --- |
| **API** | **Parameters** | **Explanation** |
| api/Payment/Token  Http Verb: GET  HTTP Status Codes:  200 – OK  400 – Bad Request  401 - Unauthorized  404 – Merchant Not Found  500 – Internal Server Error | Input Parameters: Query Strings  clientId: string  clientSecret: string  Output Parameters  Token: string | This API is used to carry communication to receive the Token.  The Token needs to be sent in header in API calls as part of Authentication.  Every token has access to all the API calls in this Project.  Authorization is not implemented. |
| api/Payment/ProcessPayment  Http Verb: POST  HTTP Status Codes:  200 – OK  400 – Bad Request  500 – Internal Server Error | Input Parameters in JSON  {  "checkoutId": "string",  "cardNumber": "string",  "expiryDate": "string",  "amount": 1000,  "currency": "string",  "cvv": "string"  }  Output in JSON  {  “status”: “string”,  “BankResponseId”: “string”,  “Messages”: string  } | If the Card Validation fails, then the HTTP Status code will be Ok but the output JSON will provide the status of the card request. |
| api/Payment/RetrievePaymentDetails  Http Verb: GET  HTTP Status Codes:  200 – OK  400 – Bad Request  500 – Internal Server Error | Input Parameters:  Query String  bankResponseId: string  Output in JSON  {  "checkoutId": "string",  "merchantId": "string",  "creditCardNumber": "string",  "expiryDate": "string",  "amount": 1000,  "currency": "string",  "cvv": "string",  "paymentStatus": "string",  "bankResponseId": "string"  "transactionMessages": "string”  } |  |

# Bank Simulator – API Documentation

|  |  |  |
| --- | --- | --- |
| API | Parameters | Explanation |
| api/Bank/ProcessPayment | **Input Payload:**  {  "checkoutId": "string",  "cardNumber": "string",  "expiryDate": "string",  "amount": 0,  "currency": "string",  "cvv": "string"  }  **Output Payload:**  {  “checkoutId”: “string”,  “PaymentTransactionId”:” string”,  “Status”: “String”,  “Message”: “Message”  } | To this api, we pass all the card details to process the payment.  On return we will send the Transaction Id and status of the payment. |

# Sequence Diagram



In the Project, The Integration takes care of the merchant part, and the Banking Service is mocked.

# Projects

|  |  |
| --- | --- |
| PaymentGateway | Payment Gateway API Project.  Implemented with the thought of Hexagonal Architecture but Service to Service communication is not handled via Mediator.  This is violating the principle of Hexagonal and made it as Clean Architecture. |
| PaymentGateway.Domain | All the domains are defined in the project. |
| PaymentGateway.Services | All the Services are handled in this project.  AuthenticationService  BankingService  PaymentService  **Note:** I handled object to object communication directly using DI. This can be implemented using Mediator (MediaTR) but I avoided as the functionality is very less in the ask and also due to Time. |
| PaymentGateway.Repositories | DBContext class is defined in this project. Seeding sample data to test the APP. |
| PaymentGateway.Tests.Integration | ProcessPayment – Process Payment integration tests verifies the entire flow. It acquires Token, send request to payment and get an update from Bank.  Bank part is mocked in Process Payment |
| PaymentGateway.Tests.Services.UnitTests | UnitTests are implemented for Services class.  I have not implemented PaymentService  UnitTests due to the Time Constraint. |

# What are highlights of the Project

1. Token based Authentication
2. Handled HttpClient Socket Saturation using IHttpClientFactory.
3. Integration tests using .NET Core SUT. With these tests entire middle wares also tested properly
4. Clean Architecture
5. Scalable to future requirements
6. Serilog Logging

# What can be improved or implemented in the project

1. Fail back mechanism after some time if the payment is failed. I implemented Retry Policy but there are many scenarios which needs to be implemented, handled, and tested properly.
   1. After 2 retries, I would have designed a component to handle these failed payments.
2. Avoiding object to object communication and implementing Mediator pattern to make this pure Hexagonal Architecture.
3. Instead of Token Based Authentication, I should have used OAuth which will be more scalable in future in the scenario like Token Refresh, Redirect Url