Payment Gateway Demo API

Contents

[1) Code Repository 2](#_Toc101900153)

[2) Application Structure: 2](#_Toc101900154)

[2.1) PaymentGatewayDemo.API 2](#_Toc101900155)

[2.2) PaymentGatewayDemo.Application 3](#_Toc101900156)

[2.3) PaymentGatewayDemo.Core: 3](#_Toc101900157)

[2.4) PaymentGatewayDemo.Infrastructure: 3](#_Toc101900158)

[2.5) PaymentGatewayDemo.ThirdParty: 3](#_Toc101900159)

[2.6) CKOBankSimulator.API: 4](#_Toc101900160)

[3) Sequence Diagram 4](#_Toc101900161)

[4) Steps to Run Application Using Swagger: 4](#_Toc101900162)

[5) Dummy Data to Test API: 7](#_Toc101900163)

[6) Assumptions: 8](#_Toc101900164)

[7) Extra Mile: 8](#_Toc101900165)

[8) Improvements: 8](#_Toc101900166)

[9) Deployment 9](#_Toc101900167)

1. Code Repository**:** Placed the code at the below Git Hub repository

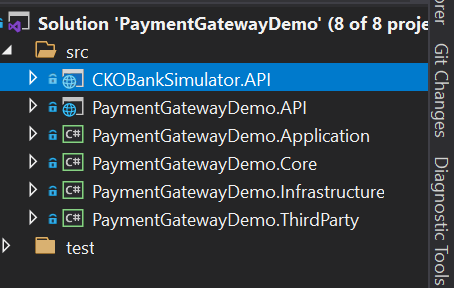
[**Code Repository**](https://github.com/ATyagi6/PaymentGateway.git)

# Application Structure:

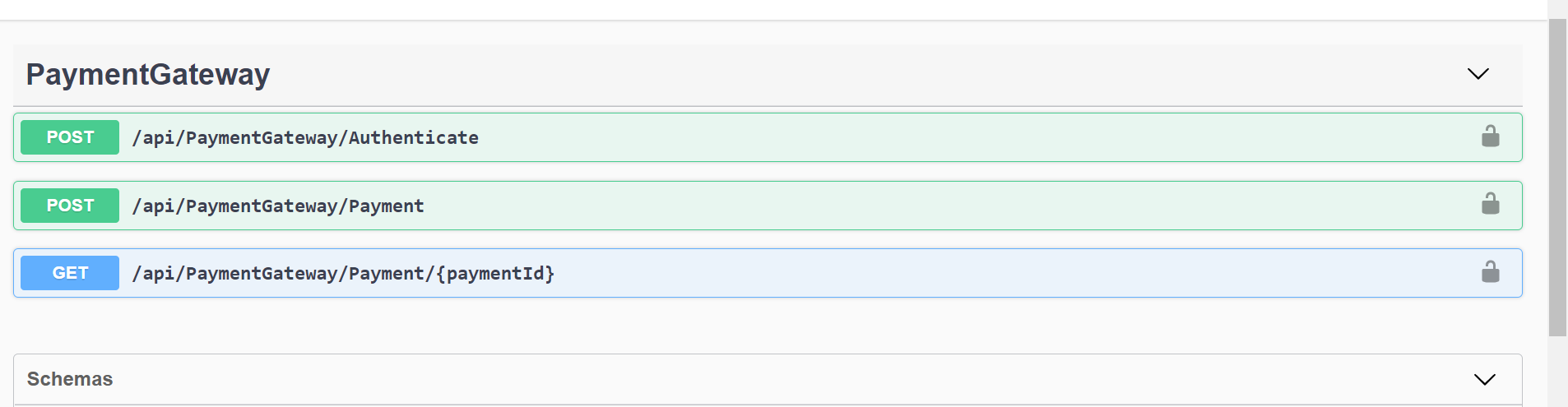
Used the clean code approach in development of this project/demo. Below is the high-level code structure of the application.

**IDE:** Microsoft Visual Studio Community 2019

**.Net Core Version**: .Net Core 5.0



* 1. PaymentGatewayDemo.API – This is the main API and contains three Actions, as mentioned in the below screenshot.



This is the api that will be called by Merchant to submit payment or get previous payment details.

* 1. PaymentGatewayDemo.Application**:** This is the Class library and it is acting as application layer.

**Below are the functionalities implemented in this project**

* Implementation of CQRS for commands and queries.
* Mapping of Entities to DTO and vice versa.
  1. PaymentGatewayDemo.Core:This is the class library project and it is acting as domain layer, which is the core of the application all the domain specific logic will be sitting in this project.

**Below are the functionalities implemented in this project**

* Domain Classes/Entities are sitting here
* Interfaces for the functionality needed by the domain and their implementation will be done in the infrastructure or ThirdParty project, so that the core is not dependent on any external dependencies and other project depends on the Core project.
* Defined the Types of exception can be raised by the application as well in this project.
* Logic for Masking the Credit Card.
  1. PaymentGatewayDemo.Infrastructure:This is a class library and in this project all the database logic is sitting. Used the In-Memory Entity Framework.

**Below are the Functionalities implemented:**

* Used EF in Memory for simplicity. Here loading the default payment details and the merchant details in DB Context class.
* Implemented the Interfaces defined in Core project for database interaction.
* Logic to Generate JWT token for api authentication.
  1. PaymentGatewayDemo.ThirdParty:

All the logic to interact with the third party apis, sdk will be sitting here. Currently for payment gateway demo API we have Bank Simulator as third party so all the logic for calling the Bank Simulator is sitting in this project.

**Functionalities Implemented:**

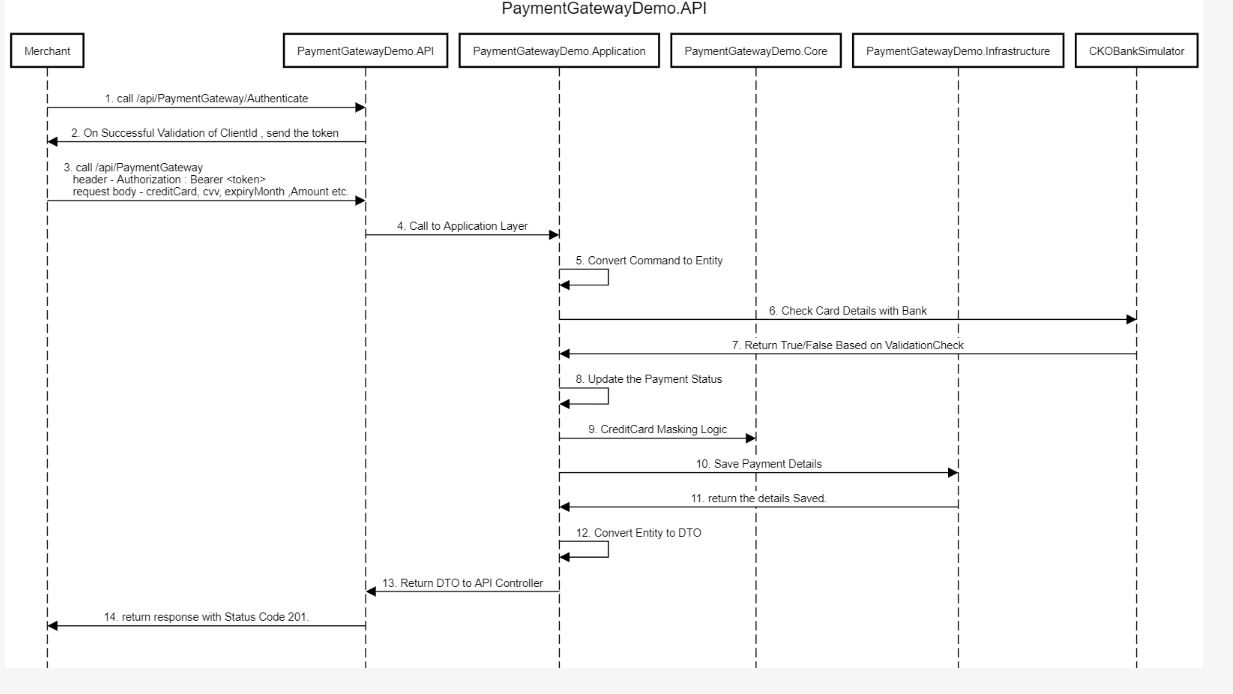
* Used Polly Package to add Resiliency in the project by using the Retry Policy with exponential back off.
  1. CKOBankSimulator.API: This is a simulator that has been built to support Payment Gateway Demo application.

**Added the simple functionality:**

* This simulator has some validation for the incoming payment request and based on pass and fail of the validation check it will return true or false, which will be used by Payment API to set the payment status.

1. Sequence Diagram**:**

Below is the high-level flow of the Post Method call in the Payment Demo API



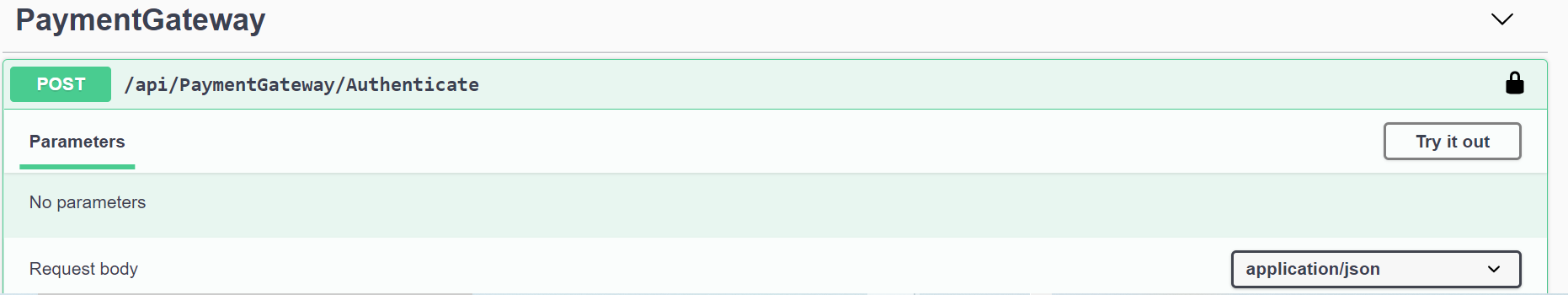
# Steps to Run Application Using Swagger:

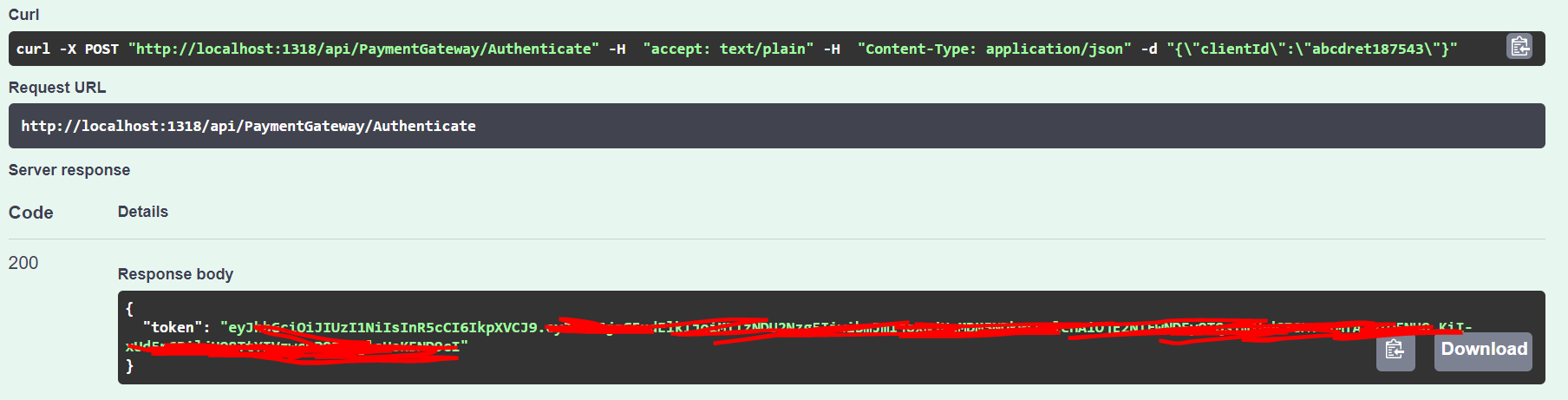
**Pre-Requisite: Solution has been configured for multiple Start up project**

* **CKOBankSimulator.API**
* **PaymentGatewayDemo.API**

**Step 1:** JWT Authentication is implemented in the API, for Post and Get Methods/Actions in API, so before calling the post method i.e. /api/PaymentGateway, we need to get the JWT token and for the API has exposed a Post Method with Allow Anonymous.

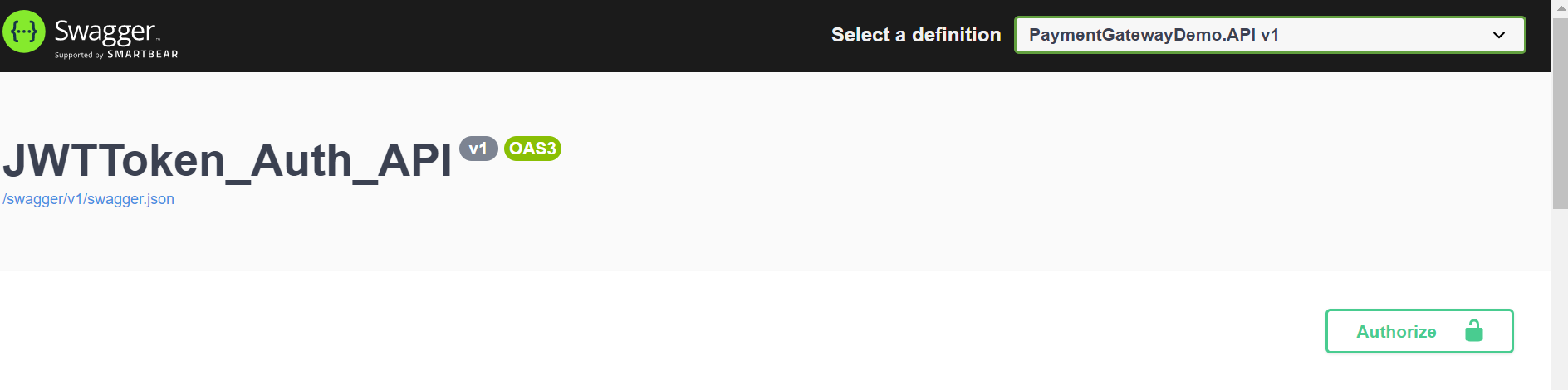
Call **/api/PaymentGateway/Authentication** method with the valid ClientID, in application I have added the dummy merchant details with client ID as **“abcdret187543”** on successful validation of the Merchant details , method will return the JWT token. Valid for 30 minutes.



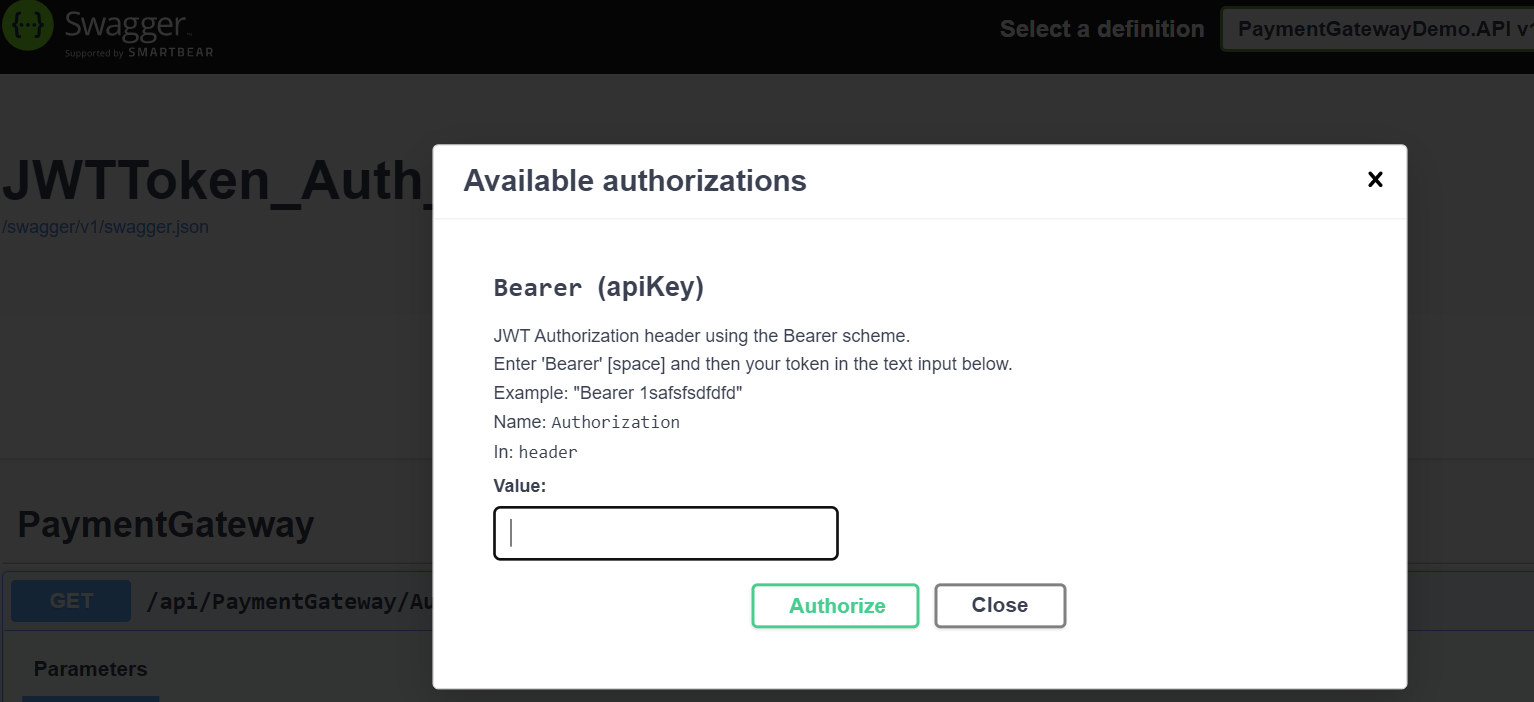


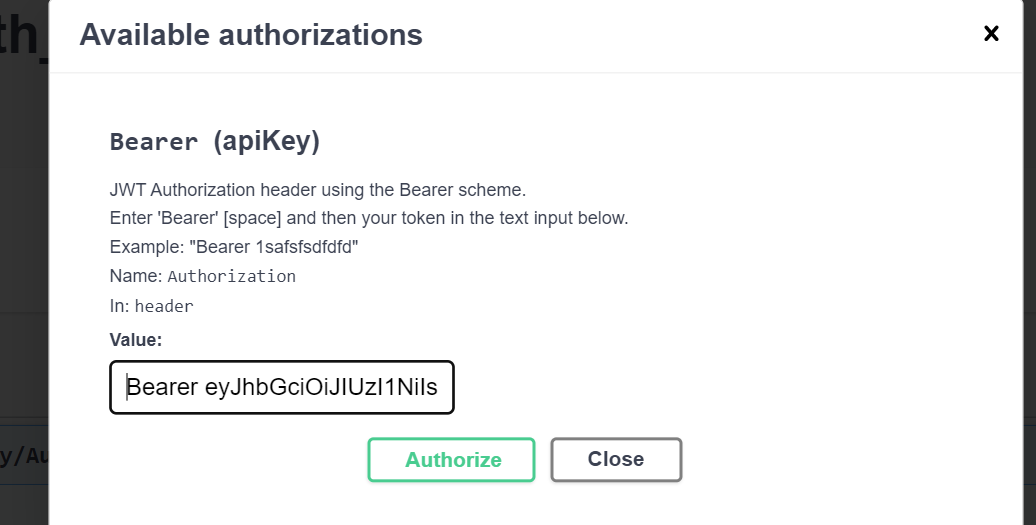
**Step 2:** Copy the JWT Token from the step 1 response.

**Step 3:** Authenticate the Swagger UI , this step is needed to pass this token in our subsequent post and get call to Payment Gateway Demo API. To do this step please click on the Authorize button (unlock symbol) on top right of the swagger page.

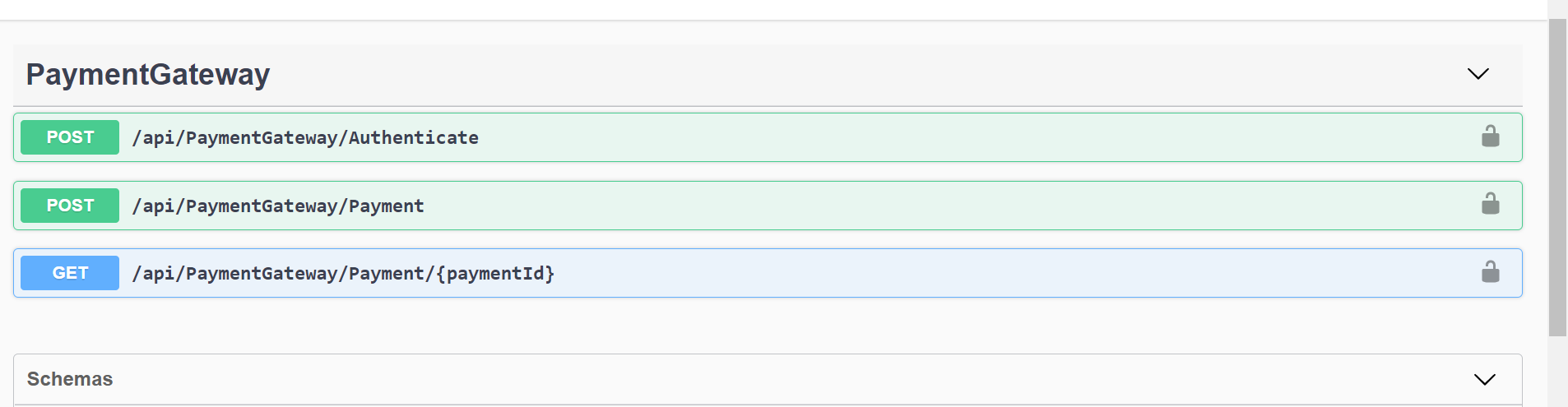


**Step 4:** Type Bearer <Space> copy the token copied in previous step and click on Authorization.

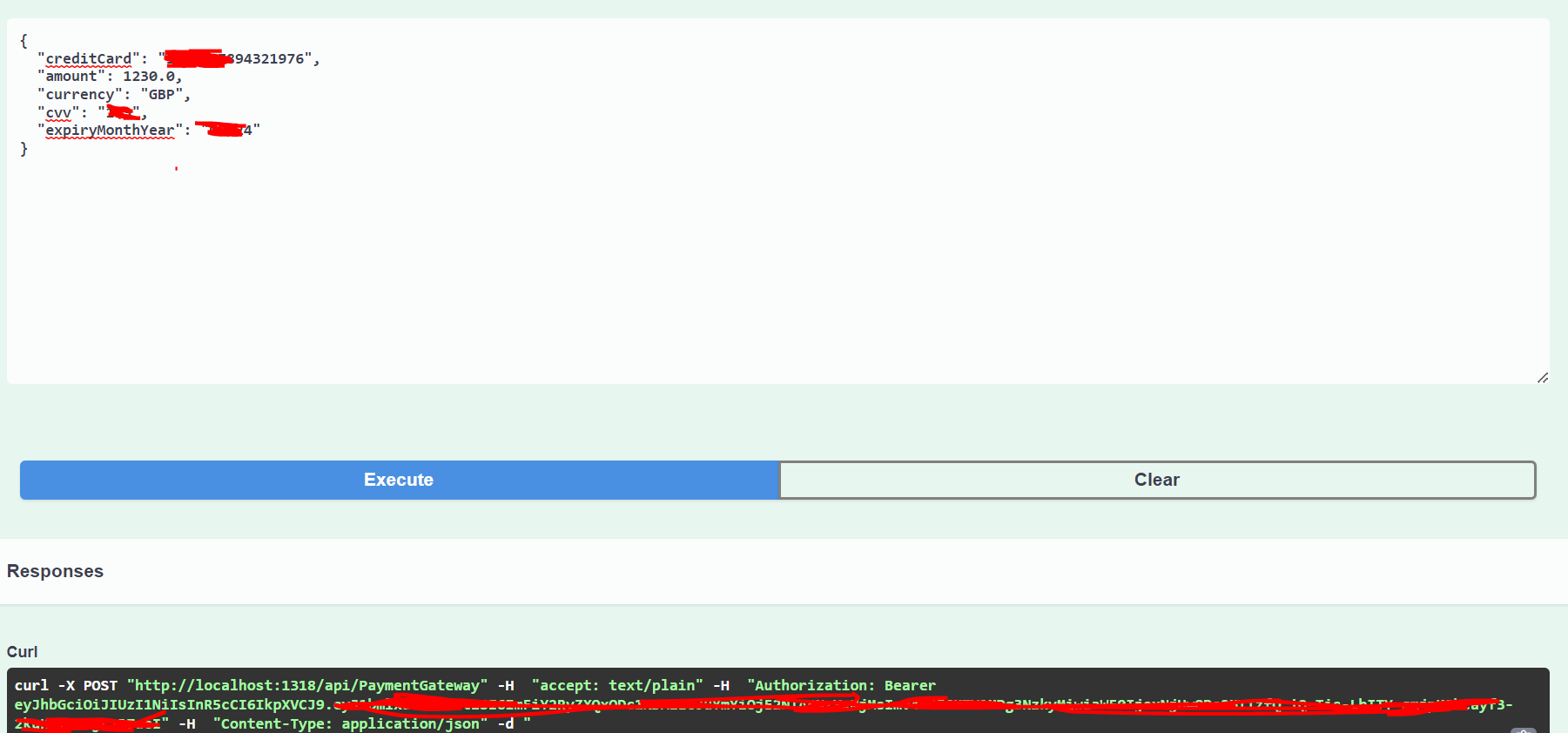




**Step 5:** On Successful Authentication, Swagger UI page will look like as shown below in screenshot (Lock will be in locked state).



**Step 6:** Call the Post/Get methods, they should not throw Unauthorised error now. The request send to the api will include the Authorization Header in the CURL window of the swagger as shown below:



# Dummy Data to Test API:

* **Dummy CreditCard used**:

CreditCard="123456789987654"

CVV ="123"

ExpiryMonthYear="12/24"

* **Dummy Merchant Details:**

MerchantId = "123456789"

MerchantName = "ABC Pvt Ltd."

ClientId = "abcdret187543"

This is the dummy data used by Bank Simulator to validate the card if we pass anything else in the payment request the bank simulator will send false in the response, in case the details are matched will return true.

# Assumptions:

* Merchant has been already registered with payment api and we have shared the ClientId, and merchant using ClientID to validate the merchant and on successful validation sending the jwt token.
* Used In Memory Entity Framework.
* Not used Automapper, as there were simple conversions, but could be used if the API grows with request and Response object.

# Extra Mile:

* Added the basic unit test cases for API.
* Implemented the Authentication.
* Implemented the Swagger for API documentation.
* Centralised exception handling in case of capturing unhandled error through middleware.
* Implementation of CQRS Pattern for cleaner controllers.

# Improvements:

* APIs could be deployed as docker, we can make use of docker compose to spin up multiple api at one go. Currently the solution properties have been configured to multiple Start up projects.
* Exception handling could be improved further.
* Right now, the API is returning the flat structure we could improve the response schema.
* Right now, the logging is done in case of exception could be extended for logging information for better traceability of the request.
* Unit test cases coverage could be improved.
* Integration Testing could also be implemented.
* Health Check could be enabled/implemented for API.
* Right now, I have not included the. gitignore file to prevent check in of appsettings.\*.json file to repository.

# Deployment

Currently both the APIs i.e., PaymentGateway.Demo.API and CKO.BankSimulator has been deployed to Azure using Azure App Service.

Below is the endpoint for the **PaymentGatewayDemo.API:**

<https://paymentgatewaydemoapi.azurewebsites.net/swagger/index.html>

Below is the endpoint for the **CKO.BankSimulator:**

<https://ckobanksimulatorapi.azurewebsites.net/swagger/index.html>

