**Components of the solution:**

There are 2 C# .Net solutions;

* **AcquiringBankSimulator** which is simulator Web API for an Acquiring bank with 2 endpoints: one POST that accepts the request and returns a “Successful” result object of ProcessPaymentResult, and one GET endpoint that returns an object of RetrievePaymentResult
* **PaymentGateway**: The actual assignment; One ASP.net core web API with 3 class libraries representing different layers of the system.
  + 3 unit test projects representing each layer of the system

**How to run the solutions:**

1. Open AquiringBankSilmulator with visual studio (2022?) and click run, this will open the swagger in a browser. Copy the base URL and the port from the browser and store to be copied later into appsettings.json in PaymentGateway API.
2. Open PaymentGateway API with visual studio, paste the base URL of AcquiringBankSimulator with correct port into appsettings.json under *"AcquiringBank": "BaseURL"* and run the API. This should bring up the swagger page for PaymentGateway API that consist of 2 endpoints. You can use Swagger or Postman to call these 2 endpoints.

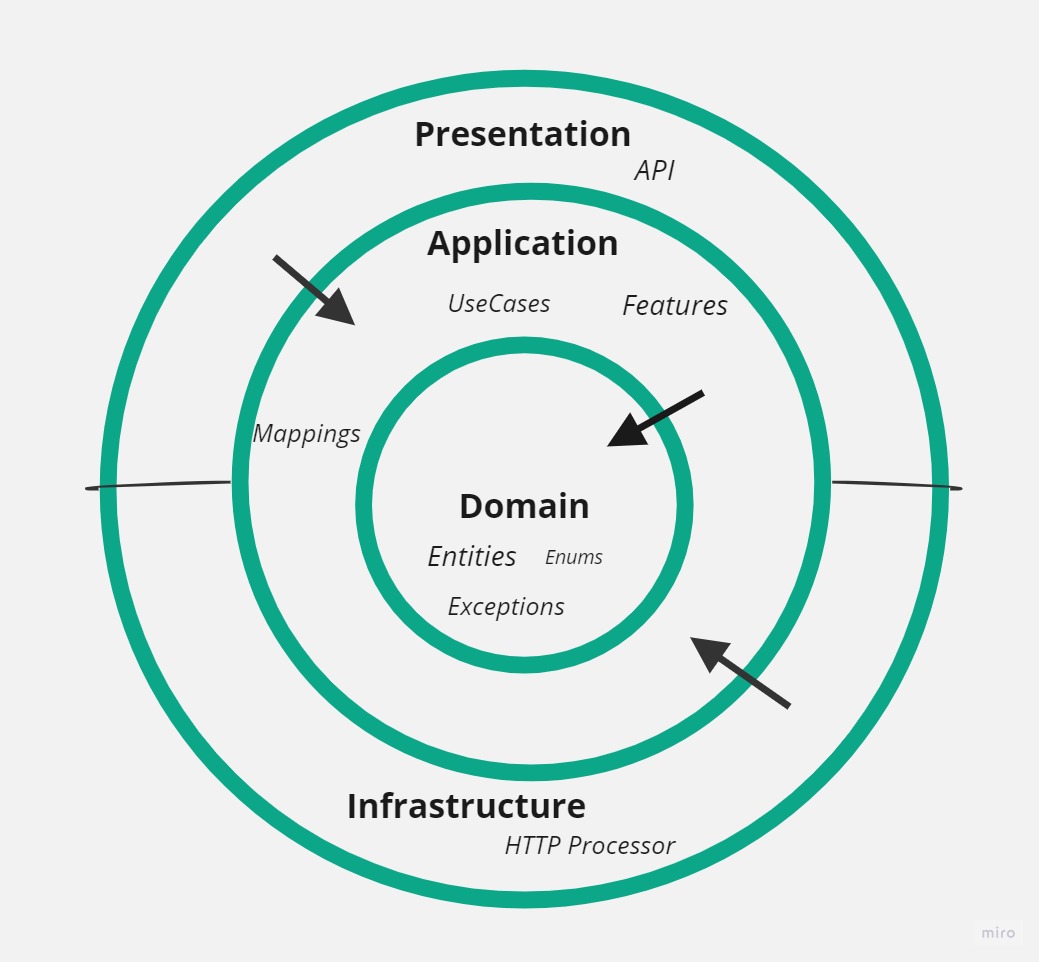
NOTE: if you get SSL error when opening Swagger page, consider temporarily allowing insecure localhost in your browser, here is the URL for Chrome:

Chrome://flags/#allow-insecure-localhost

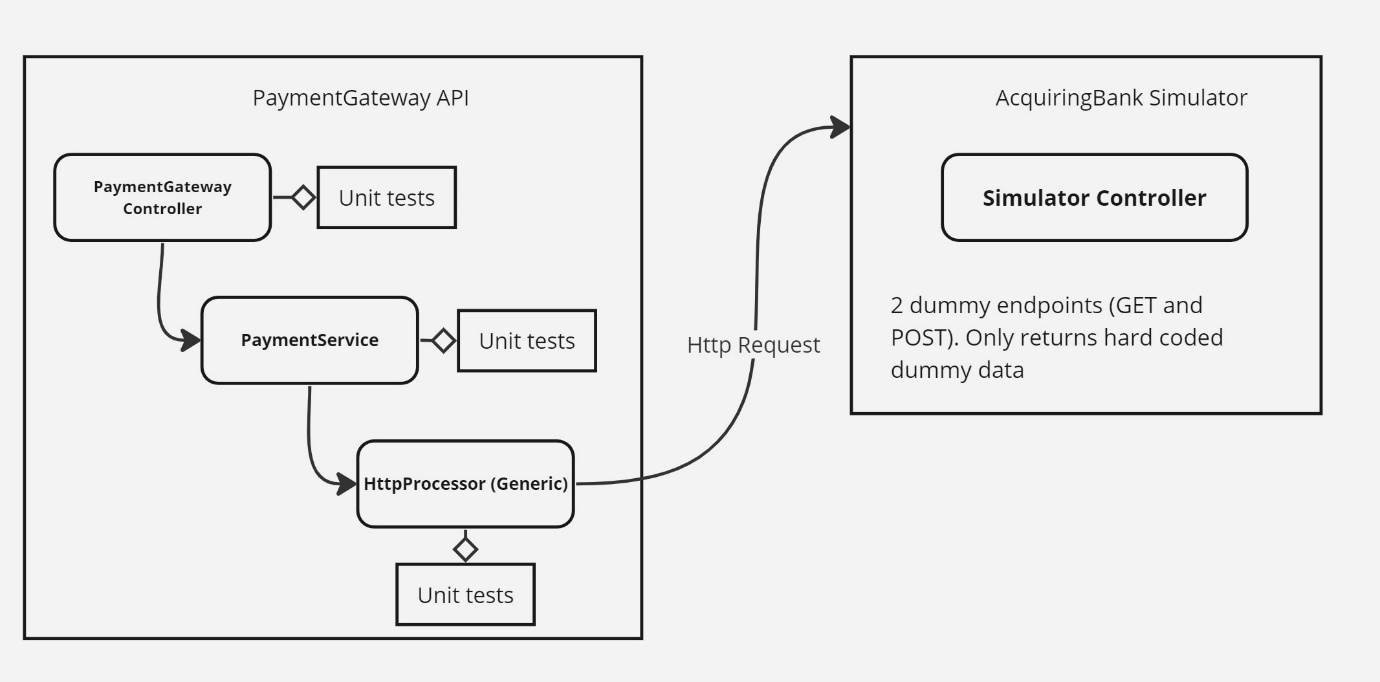
**Code Architecture and System Components:**

To demonstrate different layers of areas and domains of concern, a Clean Architecture has been implemented in 4 different projects under /src folder. Each layer only depends on the inner layer:

* Domain layer contains entities, models, exceptions and Enums.
* Application layer holds logic for Use Cases and features of the system, mappings and validations. PaymentService class has 2 functions that map the models and submits a request to HttpProcessor to send Http request
* Infrastructure deals with low level data infrastructure (Http Processor). HttpProcessor class is a generic class that can send any type of HTTP request. Retry and circuit breaking mechanism can be implemented here in the future. Loggings and other infrastructure stuff could also be implemented here in the future
* Presentation layer is the PaymentGateway API containing 2 endpoints, one GET to get payment information and one POST to process a payment.



**Request Flow Diagram:**



**Assumptions:**

* Acquiring bank will always be a separate API and gateway API shall communicate with it using HTTP
* Credit card parameters to be submitted to Payment Gateway to process a payment are: FullName, CardNumber, MerchantId, Cvv, Amount, Currency, ExpiryMonth, ExpiryYear.
* Payment information and records to be returned when retrieving a payment include above parameters without CVV and merchant Id plus the result of that payment, Created date and a PaymentId.
* When retrieving a payment information, CardNumber should be masked and only last 4 digits to be displayed
* Credit card number shall be between 15-19 characters.
* Cvv is a 3 digits number
* Card Expiry shouldn’t be in the past

**Future Improvements:**

* At the moment there is no logging, the service will need a logging component to log information about the flow at each step
* Authentication could be added to the middleware to make sure requests are authenticated before reaching endpoints
* HttpProcessor can be enhanced to include logic for retrying an Http Request, circuit breaking HTTP requests and log requests and responses.
* HTTP processor will need to accept Headers, we can pass and accept an Idempotency Key as a header that can be stored against MerchantId to make sure payments aren’t duplicated
* We can add a health check to the system through logging under catch blocks and creating alerts accordingly
* Versioning

**Cloud:**

* The service can be containerised using a dockerfile and deployed to a Kubernetes cluster. With this, we can enable auto scaling to scale out or in number of pods according to the traffic load.
* It might be helpful to use a messaging queue (e.g. Kafka) to store payment information from the acquiring bank so the payment gateway can pick up these info faster. (with security, encryption, etc in mind)
* Payments could also be cached to minimize communications to Acquiring bank
* Jenkins, Github Actions or other cloud CI build tools can be used to run the tests and calculate test coverage before each release.