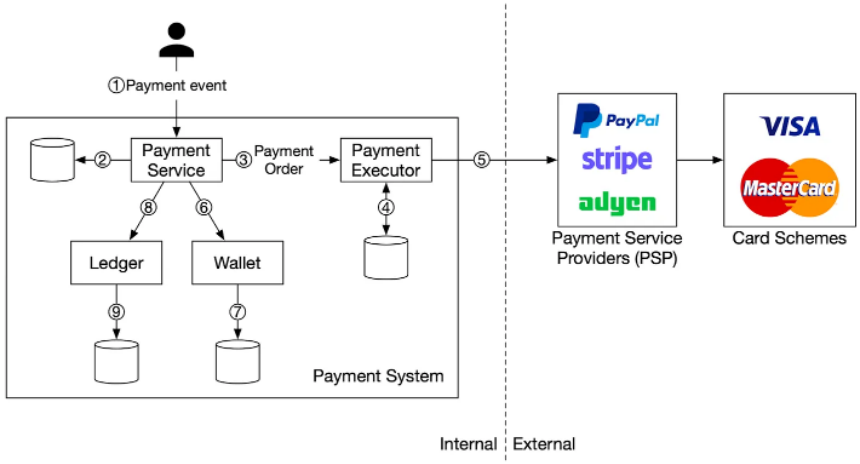
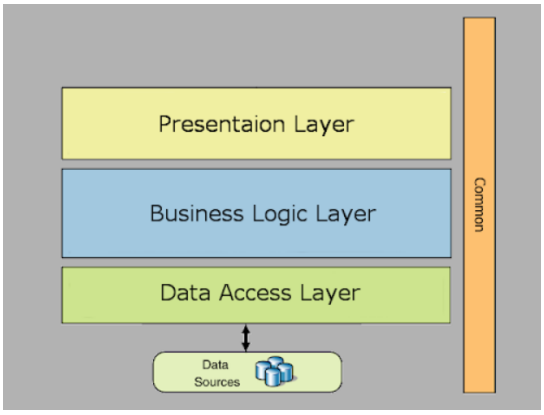
1. App Environment:
   Windows 10 OS, Visual Studio 2022, .Net 8, Intel CPU.
   To test the apps please run the two apps on the Visual Studio 2022 or you can publish them to run on a server.

2. Architecture:



   I decided to use the Payment System architecture that is widely used by the FAANG and built the component shown on the diagram above.



   I also used the N-Tire architecture with layers to separate and organise the code base so the components can be easily located and tested and updated separately.

3. Design patterns and Principles:
   I used the MVC, Factory, Repository, Strategy patterns and I used SOLID Principle. They go hand in hand to easily code with almost all software and hardware architecture and easy to test and Unit Test.

Areas for improvement:

If I had more time, I could have completed this system. But I have met the key functional requirements as prototype. But the system is designed with architecture to be extended incrementally over many sprints for it's completion.

4. Database:
   The plan was to use the EF core with Unit of Work pattern where the CRUD code could be reused. And to have transaction locking in place to stop duplication of the data, And to have

the DB designed so it could be shard for the future. But to save time I hard coded the data on a cs file to simulate the DB to complete this prototype on time. If I had more time I could of better designed the tables in place to be more compatible for data relationship and store less memory, but I did place some repeated data as constants and enum to save DB space but there are allot of other data that are on the app hard coded than have them stored in the DB because I did not have enough time to place them as constants. I would have liked to create replication of the database in place for recovery and some read only data cached to take the load of the SQL DB call.

5. Cryptography and Data integrity:
   If I had more time, I would have had the sensitive data encrypted for storage on the DB and decrypted for used. I would have also liked to have the sensitive data in transit between Apis to be encrypted and decrypted. But I did get the time to have the transit data to be converted to text and this is because cash amount when passed via api could be altered if the receiving Api is coded in another language. Taking a decimal value of C# to be converted to a decimal value of another programming language will not convert accurately. So it's better to have the cash value converted from string to whatever decimal type by the receiving programming language.

6. Clean Code and Objects:
   I wrote the code in away where its readable then fill it up with comments and named Classes, Methods and variables in a readable way but if I had more time I could have reviewed my code and improved it a lot more. I could have also made less classes to be reused and used less object mappings and used DTO's more. I took some api response codes from the Checkout website https://www.checkout.com/docs/developer-resources/codes but I did not have enough time to code to take them all I only took some. And to write test for them all would have took me even more time. And I also forgot to rename the Async methods with Async at the end.

7. System Design:
   If I had more time, I would have like to make this system an Event Driven architecture with Message Queuing so the auditing could be done in the background regularly and to avoid duplication of events. And to have UUID instead of GUID for each messages to keep messages more unique. And I would have like to have .Net Polly to manage the Api retries in case of Circuit Breaker, Hedging, Timeout, Rate Limiter and Fallback.

   I would have liked to have the Api return html payment form for the merchant and send back in progress html messages, but to save time I only coded the min requirement.
   I would have liked to add the currency exchange on the payment for international card holders.

   If I had time I would have like to add the PSP system to manage the pay settlement and reconciliation process to generate a batch file. As well as have credit card management app that runs in-between banks.

8. Error handling:
   If I had more time, I would have liked to add a try catch and an error log system to save the error on a DB and sent out emails. I also would have liked to add more logic in the code base

to detect and handle the error and return more appropriate message listed on the https://www.checkout.com/docs/developer-resources/codes site.

9. Security:
   If I had time, I would have liked to add authentication and authorisation between the Apis on top of the encryption. I would have liked to code more on the fraud detection and card limits and expiration on the bank side. I also would have liked to code 3D security to send of a pin.

10. Testing:
    I did not have time to write unit test but I have managed to black box test it. If I had more time I would have written unit tests. Please see the Appendix for the Json to test.

Appendix:

Api GetBankCardInfo:

John@example.com,

Adam@example.com,

Cevil.Accounts@ClothingRental.com.

Api GetMerchantTransactionsById:

1,

2,

3

Api Payment:


```
{
 "token": "b3415758-6d29-49f0-b1b5-5f861352434a",
 "merchantId": "2",
 "currency": "GBP",
 "amount": "90",
 "card": {
  "type": "Visa",
  "number": "4665420958526038",
  "cvv": "789",
  "firstName": "Matthew",
  "lastName": "Clark",
  "vaildFrom": "03/23",
  "expiryDate": "03/26"
 }
}
```