Checkout.com .NET Challenge
Building a Payment Gateway

Solution by Gurdeep Singh

Introduction

The solution to the code challenge is developed in Visual Studio using .Net Core v 3.1

Documentation

- All documentation is available in folder - \Documentation

Assumptions

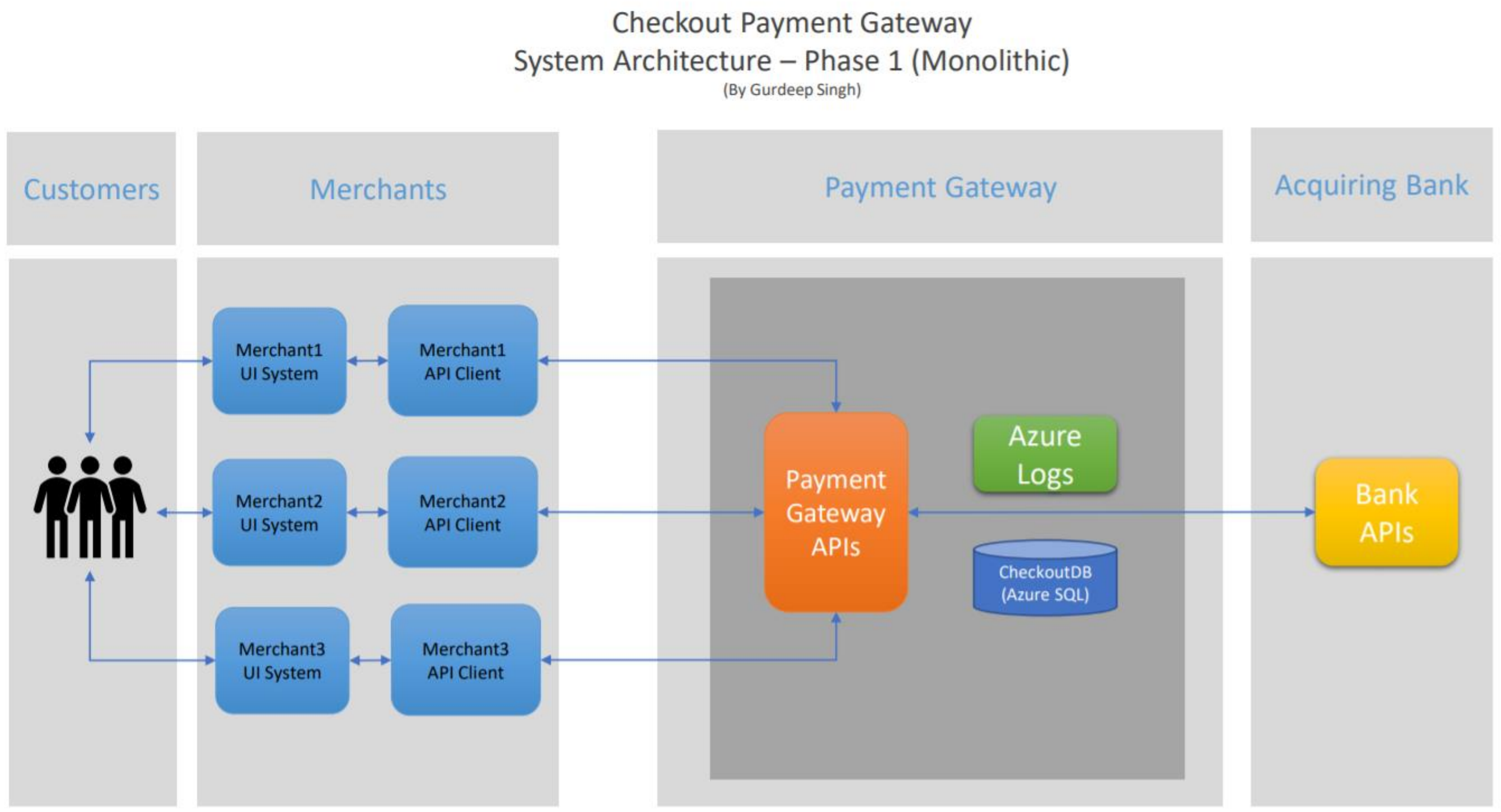
- Acquiring Mock Bank API security is omitted
- Address details validations in Credit card are omitted

Design & Architecture

Following components and principals are practiced during the development of this solution

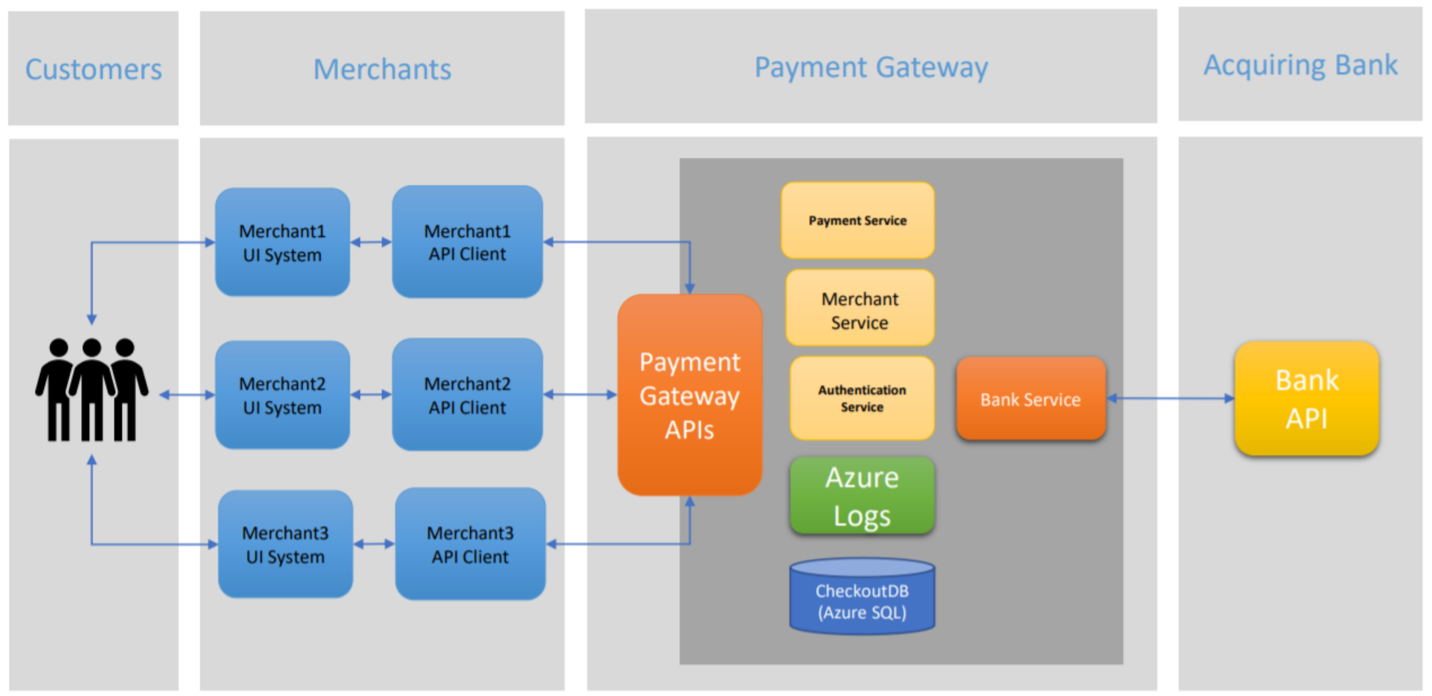
- **Diagrams** – Tech Diagram (Checkout_Architecture.pdf) is Documentation folder.

- **Phase 1** – This is how the current code is implemented. The tech diagram is as below. The architecture is monolithic as all of the system components are available in single Repository library.



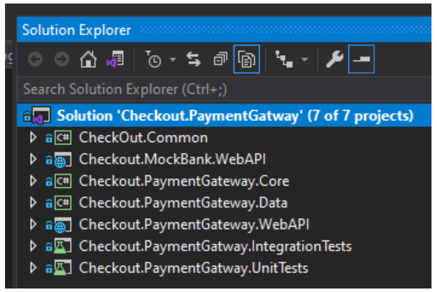
- **Phase 2** – As system grows the better architect may be achieved by migrating into Microservices architecture. The logical breakdown of functions such as Payment service, Bank service, Authentication Service & Merchant Service can be developed. Each logical function can be independent microservices. This can be benefit as maintenance will become easier and can be re-sued by other systems in the organisation.

Checkout Payment Gateway
System Architecture – Phase 2 (Microservices)
(By Gurdeep Singh)



- **Entity Framework .Net core –** For Object relational mapping and Data layer the Entity framework for .Net Core has been used**.**

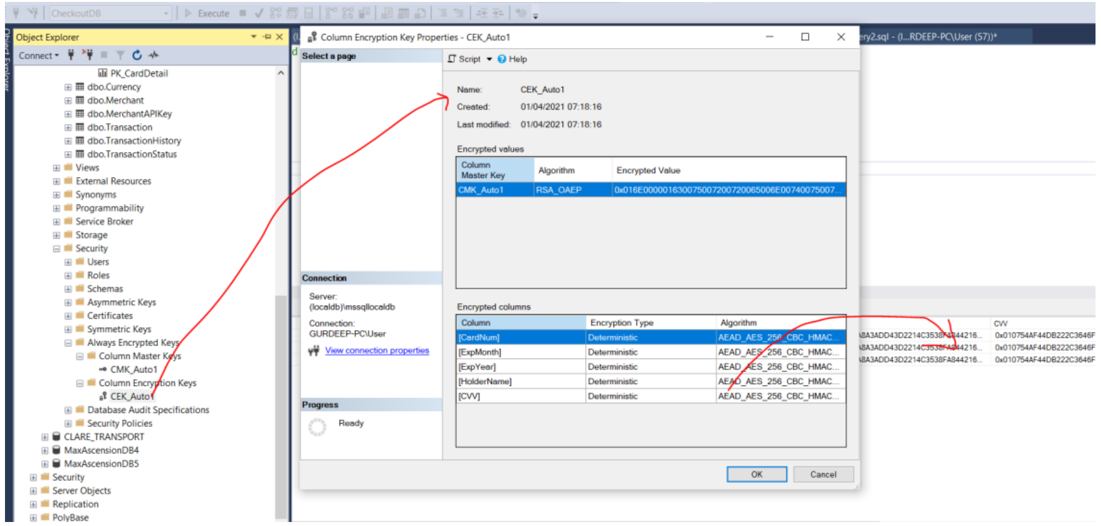
- **Repository pattern & Unit of Work –** Repository pattern and Unit of work is used. This helps to keep the architecture clean and more maintainable as project grows over the time.
- **Project Structure** – Following are the projects in the solution.



Solution 'Checkout.PaymentGatway' (7 of 7 projects)
- CheckOut.Common
- Checkout.MockBank.WebAPI
- Checkout.PaymentGateway.Core
- Checkout.PaymentGateway.Data
- Checkout.PaymentGateway.WebAPI
- Checkout.PaymentGatway.IntegrationTests
- Checkout.PaymentGatway.UnitTests

Database

- Azure SQL DB is used under local DB Instance.
- Migration scripts are available in **Checkout.PaymentGateway.Data** Project.
- **Data Encryption** – The sensitive data in Credit Card Details table is encrypted using SQL Always Encrypted System and Column keys.



Column Encryption Key Properties - CEK_Auto1

Name: CEK_Auto1
Created: 01/04/2021 07:18:16
Last modified: 01/04/2021 07:18:16

Encrypted values

Column Master Key	Algorithm	Encrypted Value
CMK_Auto1	RSA_OAEP	0x016E000001630075007200720065006E00740075007...

Encrypted columns

Column	Encryption Type	Algorithm
[CardNum]	Deterministic	AEAD_AES_256_CBC_HMAC...
[ExpMonth]	Deterministic	AEAD_AES_256_CBC_HMAC...
[ExpYear]	Deterministic	AEAD_AES_256_CBC_HMAC...
[HolderName]	Deterministic	AEAD_AES_256_CBC_HMAC...
[CVV]	Deterministic	AEAD_AES_256_CBC_HMAC...

Connection

Server:
(localdb)\mssqllocaldb

Connection:
GURDEEP-PC\User

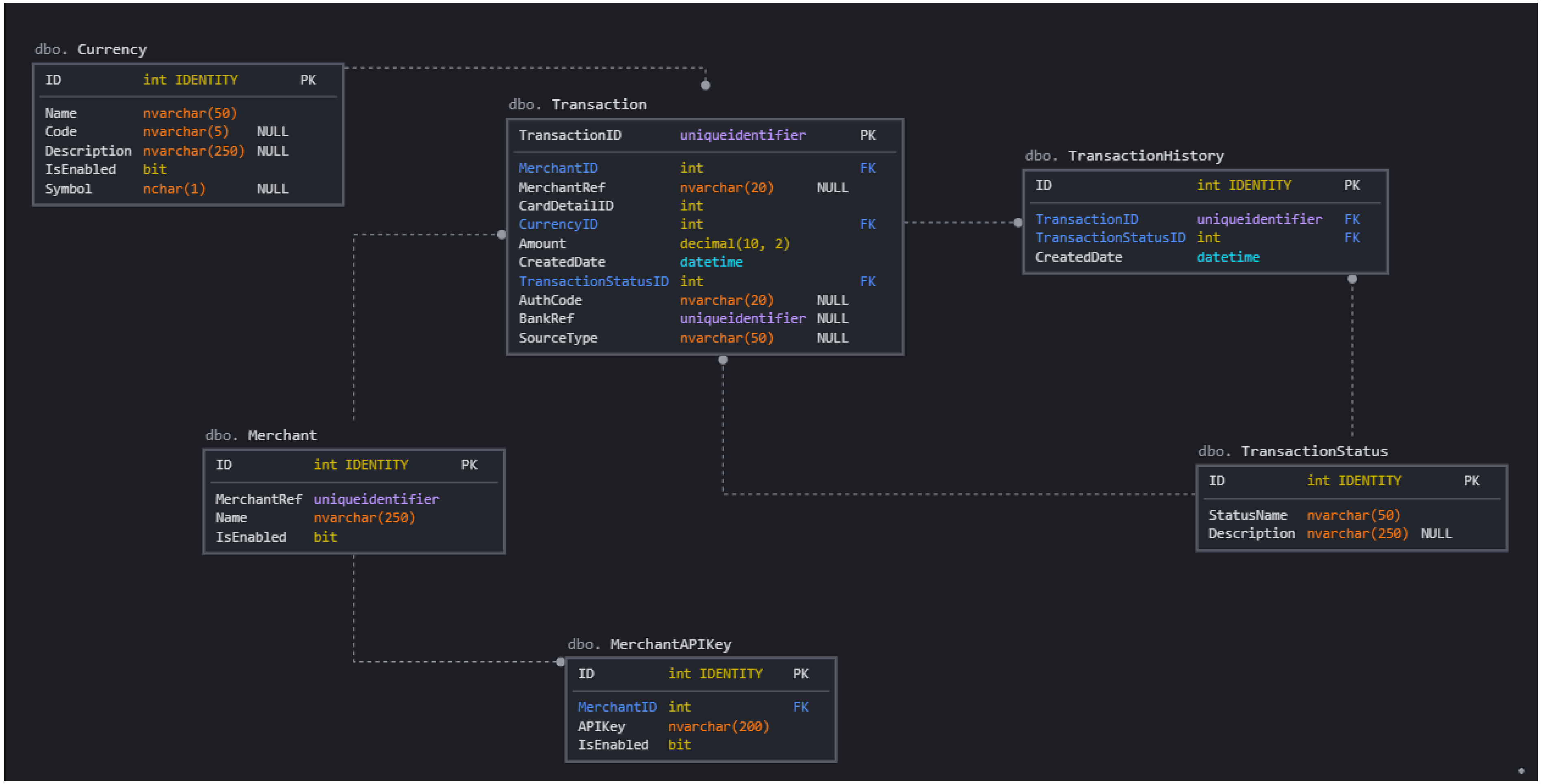
View connection properties

Progress

Ready

	CVV
A8A3ADD43D2214C3538FA844216...	0x010754AF44DB222C3646F
A8A3ADD43D2214C3538FA844216...	0x010754AF44DB222C3646F
A8A3ADD43D2214C3538FA844216...	0x010754AF44DB222C3646F

- Data Model



dbo. Currency

ID	int IDENTITY	PK
Name	nvarchar(50)	
Code	nvarchar(5)	NULL
Description	nvarchar(250)	NULL
IsEnabled	bit	
Symbol	nchar(1)	NULL

dbo. Transaction

TransactionID	uniqueidentifier		PK
MerchantID	int		FK
MerchantRef	nvarchar(20)	NULL	
CardDetailID	int		
CurrencyID	int		FK
Amount	decimal(10, 2)		
CreatedDate	datetime		
TransactionStatusID	int		FK
AuthCode	nvarchar(20)	NULL	
BankRef	uniqueidentifier	NULL	
SourceType	nvarchar(50)	NULL	

dbo. TransactionHistory

ID	int IDENTITY	PK
TransactionID	uniqueidentifier	FK
TransactionStatusID	int	FK
CreatedDate	datetime	

dbo. Merchant

ID	int IDENTITY	PK
MerchantRef	uniqueidentifier	
Name	nvarchar(250)	
IsEnabled	bit	

dbo. TransactionStatus

ID	int IDENTITY	PK
StatusName	nvarchar(50)	
Description	nvarchar(250)	NULL

dbo. MerchantAPIKey

ID	int IDENTITY	PK
MerchantID	int	FK
APIKey	nvarchar(200)	
IsEnabled	bit	

- **Manual DB Creation –** SQL scripts are provided (*\Checkout.PaymentGateway.Data\Scripts*) for generating the DB Objects manually. Please run them in their naming order - 00_Create_Encryption_Keys.sql, 01_Create_DB.sql, 03_DB_Initial_Seed.sql. This should give you the complete DB required for the application.

Card Validations

- CardValidator Nuget package is used for Credit Card Validations - https://www.nuget.org/packages/CreditCardValidator/

Application logging & metrics

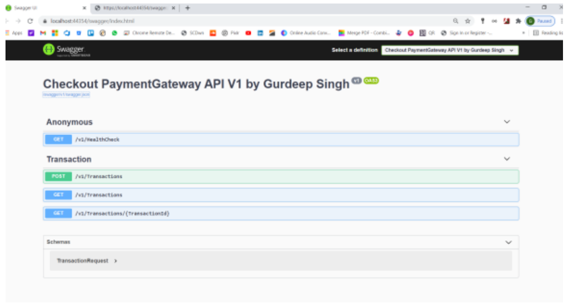
- Azure App Insights can be used for logging. The logging is added for any error and for some Payment related information which can be used for audit or error tracking.

Authentication

- Basic authentication using API Secret. **X-API-Key** is the header name and the generated string is the Value that is given by Payment gateway. Each Merchant will have API Keys assigned and is stored in **MerchantAPIKeys** table in the Database.
- **AuthenticationHandler** class is used to verify the identity and once validated the Merchant Ref is saved in Claims. This Claim value then is available for the rest of the application to use.

API client

- Postman scripts are supplied in PostManScripts folder under Documentation.
- Swagger document is also implemented - https://localhost:44354/swagger/index.html. Swagger file (swagger.json) is also located in Documentation folder.


Swagger UI
localhost:44354/swagger/index.html
Swagger
Select a definition
Checkout PaymentGateway API V1 by Gurdeep Singh

Checkout PaymentGateway API V1 by Gurdeep Singh V1 OAS3

Anonymous

GET /v1/HealthCheck

Transaction

POST /v1/Transactions

GET /v1/Transactions

GET /v1/Transactions/{TransactionId}

Schemas

TransactionRequest

Testing

- NUnit & Moq are used for unit testing and several positive and negative scenarios are tested.
- Postman repeater or JMeter are used for performance testing.

Further development

In addition to the Micro services architecture the following possible enhancements can be done

- JSON Web Token (JWT) – Jwt tokens can be used to enhance API security. These are better and secure over the API Keys security.
- Caching – For storing static information such as Currency list we can implement the caching to improve the performance. Cloud based caching such as Redis Caching can be used.
- Load balancing – For performance & Load the solution can be deployed to cloud and load balancing can be implemented.
- Segregate the Security Server – The security server to verify the JWT tokens can be physically separated from the Data Layer and other API code. This will enhance the security of the application.