PAYMENT RULE ENGINE CASE STUDY

1. Introduction

The Payment Rule Engine is designed to evaluate and process payment transactions based on a set of flexible and dynamic rules. This engine allows for real-time decision-making regarding payment methods, routing, fees, and additional verification requirements. Leveraging AWS services such as Lambda, DynamoDB, and API Gateway, this solution provides a scalable and maintainable way to manage payment processing rules. The engine is built to accommodate a variety of rules and criteria, including customer type, transaction amount, payment method, country, currency, 3DS information etc. By storing rules in DynamoDB and processing them with AWS Lambda, the system is both adaptable and resilient, ensuring that new rules can be added without modifying the core codebase.

2. Architecture

Overview

The Payment Rule Engine uses AWS services to provide a scalable and efficient solution for evaluating payment rules. The architecture is composed of the following components:

- **AWS API Gateway**: Exposes the rule engine as a RESTful API. It handles incoming payment requests and routes them to the Lambda function for processing.
- **AWS Lambda Authorizer**: Lambda authorizers provide a flexible mechanism to secure APIs by validating and allowing requests based on custom criteria defined in Lambda functions.
- **AWS Lambda**: Hosts the rule evaluation logic. It retrieves rules from DynamoDB, evaluates them against the incoming transaction data, and applies the necessary actions.
- **AWS DynamoDB**: Stores the rules in a flexible, JSON-based format. DynamoDB allows for easy updates and additions of new rules without changing the code.

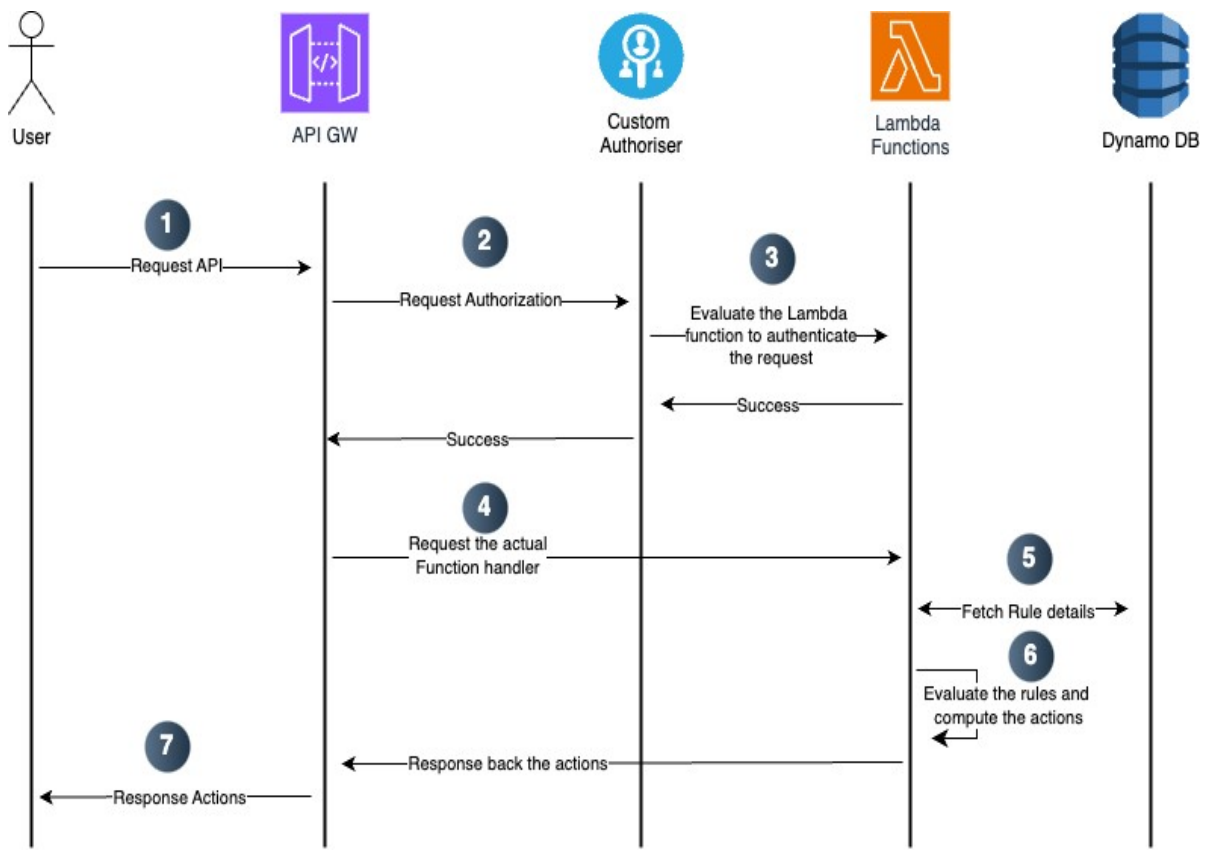
Data Flow

1. **Transaction Request**: The transaction request is sent to API Gateway.
2. **Lambda Authorization:** API GW invokes the custom authenticating lambda function to authenticate the request.
3. **Lambda Invocation**: API Gateway triggers the Lambda function with the transaction data.
4. **Rule Evaluation**: The Lambda function scans DynamoDB for applicable rules and evaluates the transaction against these rules.
5. **Action Application**: Based on the evaluation, the Lambda function applies the necessary actions and constructs the response.
6. **Response Return**: The Lambda function returns the evaluation results to API Gateway, which then sends the response back to the requester.

3. Use Case

Overview

The Payment Rule Engine processes payment transactions by evaluating them against a set of dynamic rules. The system interacts with multiple AWS components to ensure that transactions are processed according to predefined criteria. This use case describes the flow of interactions from the user to the final response, involving AWS API Gateway, Custom Authorizer, Lambda, DynamoDB, and the rule evaluation process.



Actors

- **User**: The entity initiating a transaction.
- **API Gateway**: Exposes the Lambda function as a RESTful API and handles incoming requests.
- **Authorizer**: An entity or service used to authenticate and authorize API requests.
- **Lambda Functions**: It acts as the compute service that evaluates the transaction against the rules and also as a custom Authorization logic to authenticate the requests.
- **DynamoDB**: The database service where rules are stored.

Flow of Interactions

1. **User Initiates a Payment Transaction**
   • The user sends a payment request, including transaction details such as country, payment method, amount, and other relevant data, to the system.

2. **API Gateway Receives the Request**
   • The request is routed through AWS API Gateway, which serves as the entry point for the payment rule engine.
   • API Gateway invoke the custom authorizer to validate the request. This step ensures that the request is authenticated and authorized before further processing.

3. **API Gateway Invokes the Authorizer Lambda function**
   Custom authorizer calls the Lambda function to evaluate the authentication conditions, and returns the status to the API Gateway.

4. **API Gateway Invokes the Lambda Function**
   • Upon successful validation, API Gateway triggers the AWS Lambda function with the payment transaction data.

5. **Lambda Function Fetches rule details from Dynamo DB**
   • The Lambda function queries DynamoDB to retrieve the rules applicable to the transaction.

6. **Rule Evaluation**
   •The Lambda function evaluates the transaction against the retrieved rules:
      •**Matching Rules**: Each rule is checked to see if it matches the transaction criteria (e.g., country, payment method, transaction amount).
      •**Applying Actions:** If a rule matches, the corresponding actions are determined (e.g., routing, fee application).

7. **API Gateway Sends the Response**
   •The Lambda function returns the constructed action response to API Gateway.
   •API Gateway forwards the response back to the user.

4. Security

Data Security

- **Encryption**: Data in transit between API Gateway and Lambda is encrypted using HTTPS. Data stored in DynamoDB is encrypted at rest.
- **Access Control**: Lambda functions and DynamoDB tables use IAM roles and policies to ensure that only authorized services and users have access. API Gateway is configured to use custom authentication with Lambda Authorizer.

Rules Management

- **Access Control**: Rule creation and modification in DynamoDB are restricted to authorized personnel through IAM policies.
- **Audit Logging**: Changes to rules and access to DynamoDB are logged using AWS CloudTrail, providing visibility into access patterns and changes.

Best Practices

- **Least Privilege**: IAM roles and policies follow the principle of least privilege, granting only the permissions necessary for each component.
- **Regular Security Reviews**: Regular reviews and updates of IAM roles, policies, and security configurations ensure ongoing compliance with security best practices.
- **Rate limiting and throttling:** Rate limiting and throttling is configured at the API Gateway to ensure that the fare access and high availability of the system.

5. API Specifications

The api specification for the application is made available in the GIT repository : **https://github.com/arjunr1432/payment-rule-engine/blob/main/payment-rule-engine-app/src/main/resources/openapi.yml**

Optionally if the application is started in local, it can be accessed from:
http://localhost:8080/swagger-ui/index.html

The main request and response data models can be consolidated as follows.

Request Models

SL No	Field Name	Data Type	Description
1	Country	String	ISO Country names
2	Currency	String(3)	3 Character ISO currency codes
3	PaymentMethod	String	One among the following payment methods **"MasterCard","Visa","CreditCard","Vipps"**
4	CustomerType	String	One among the following customer types **"Employee","Debtor","Creditor","Manag**

			er"
5	TransactionAmount	Double	The transaction amount to be processed.
6	Previous3DS	Boolean	Boolean flag about the status of previous 3DS.
7	DaysSinceLast3DS	Integer	No of days from last 3DS has been completed.

Response Models

SL No	**Action Name**	**Data type**	**Description**
1	PaymentMethod	String	One among the following payment methods **"MasterCard","Visa","CreditCard", "Vipps"**
2	Route	String	Selected routing based on the configuration.
3	Waive3DS	Boolean	Boolean flag to skip 3DS
4	EnableFeature	Boolean	Boolean flag to enable feature
5	RequireAdditionalVerification	Boolean	Boolean flag to incur additional verification
6	Fee	Double	Computed fee for the transaction amount.

Accessing The API

The service is deployed and available in cloud infrastructure described above. You can consume it according to the API specification available in the GitHub repository. The public URL to the service and some other configurations required are as below.

- **URL:** https://2ofrmwsvdf.execute-api.eu-north-1.amazonaws.com/payment-engine-stage/evaluateRules
- **Security Header:** Add the header "**jwt_token"** with value **"allow"** to successfully get authenticated to the system.

Request:

```
curl --location 'https://2ofrmwsvdf.execute-api.eu-north-1.amazonaws.com/payment-engine-stage/evaluateRules' \
--header 'jwt_token: allow' \
--header 'Content-Type: application/json' \
--data '{
    "Country" : "Norway"
}'
```

Response:

```
{
"PaymentMethod": "Vipps"
}
```

6. Sample Request/Response

SL No	Request	Response

1	{
	"Country" : "Norway"
	}
		{
		"PaymentMethod": "Vipps"
		}

2	{
	"PaymentMethod" : "MasterCard",
	"Currency": "SEK"
	}
		{
		"Routing": "AcquirerC"
		}

3	{
	"PaymentMethod" : "CreditCard",
	"TransactionAmount": 9999.0,
	"Currency": "NOK"
	}
		{
		"Fee": 199.98
		}

4	{
	"Previous3DS": "Success",
	"DaysSinceLast3DS": 123
	}
		{
		"Error": "[Validation Errors: 1:- Previous3DS should be boolean.]"
		}

7. Further Improvements and Changes

- Improve the current API authentication with mTLS/OAuth2.0 based security for Authentication and Authorization.
- Automate the CI/CD pipelines for the smoother operations.
- Optionally add a back-office frontend application to easily view and update the rule details.
- Deploy the API specification details with the help of AWS services to make it public and accessible.

8. Summary

The Payment Rule Engine is a robust, flexible system for processing payment transactions based on dynamic rules. By leveraging AWS services such as Lambda, DynamoDB, and API Gateway, the engine provides a scalable solution that can easily adapt to new requirements. The architecture supports real-time decision-making, allowing for the efficient handling of payment transactions with varied criteria. Security measures are implemented to protect data and control access, ensuring that the system operates securely and efficiently.

9. References

- **https://docs.aws.amazon.com/**
- **https://medium.com/**
- **https://github.com/**
- **https://chatgpt.com/**