**Rest API Documentation for Payment Gateway and Key Design Considerations and Assumptions**

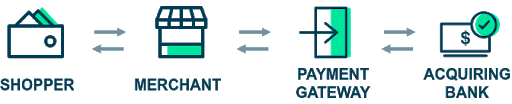
# Overview

This project is building a payment gateway

E-Commerce is experiencing exponential growth and merchants who sell their goods or services online need a way to easily collect money from their customers.

In this project I attempt to build a payment gateway, an API based application that will allow a merchant to offer a way for their shoppers to pay for their product.

Processing a card payment online involves multiple steps and entities:

**[](https://github.com/cko-recruitment/.github/blob/main/images/card_payment_overview.png)**

**Shopper:**Individual who is buying the product online.

**Merchant:**The seller of the product. For example, Apple or Amazon.

**Payment Gateway:**Responsible for validating requests, storing card information and forwarding payment requests and accepting payment responses to and from the acquiring bank.

**Acquiring Bank:**Allows us to do the actual retrieval of money from the shopper’s card and pay out to the merchant. It also performs

This document outlines the key design considerations and assumptions made during the implementation of the Payment Gateway system. The system allows merchants to process payments from shoppers and retrieve details of previously made payments. It also simulates interactions with an acquiring bank to ensure full testing of the payment flow.

# Payments API Documentation

## Base URL

https://<api-domain>/api/Payments

## Endpoints

### 1. Get Payment Details

**Description:** Retrieve details about a specific payment transaction by its ID.

**Endpoint:**

GET /api/Payments/{id}

**Path Parameters:**

* id (Guid) - The unique ID of the payment transaction.

**Responses:**

|  |  |  |
| --- | --- | --- |
| **Status Code** | **Description** | **Response Body** |
| 200 | Payment details retrieved successfully | PostPaymentResponse JSON object |
| 404 | Payment with the given ID was not found | null |

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Request Example:**

GET /api/Payments/a3f5b231-9c8d-4bdf-b9a2-7c57837c2c6e

**Response Example:**

{

"id": "0bb07405-6d44-4b50-a14f-7ae0beff13ad",

"status": "Authorized",

"cardNumberLastFour": "8877",

"expiryMonth": 4,

"expiryYear": 2025,

"currency": "GBP",

"amount": 100

}

### 2. Create a Payment Request

**Description:** Create a new payment request using card details.

**Endpoint:**

POST /api/Payments

**Request Body:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Type** | **Required** | **Description** |
| cardNumber | String | Yes | The card number (14-19 numeric characters). |
| expiryMonth | int | Yes | Expiry month of the card (1-12). |
| expiryYear | Int | Yes | Expiry year of the card (e.g., 2025). |
| currency | string | Yes | The currency code (EUR, USD, GBP). |
| amount | Int | Yes | The transaction amount (integer).In the minor currency unit. For example, if the currency was USD then   * $0.01 would be supplied as 1 * $10.50 would be supplied as 1050 |
| cvv | string | Yes | The CVV of the card (3-4 numeric characters). |

|  |  |  |  |
| --- | --- | --- | --- |

**Responses:**

|  |  |  |
| --- | --- | --- |
| **Status Code** | **Description** | **Response Body** |
| 200 | Payment request created successfully | PostPaymentResponse JSON object |
| 404 | Failed to create a payment | Error message |
| 400 | Bad Request | ProblemDetailsResponse JSON object |

**Request Example:**

POST /api/Payments

Content-Type: application/json

{

"cardNumber": "1234567812345678",

"expiryMonth": 12,

"expiryYear": 2025,

"currency": "USD",

"amount": 100,

"cvv": "123"

}

**Response Example (Success):**

json

CopyEdit

{

"id": "a3f5b231-9c8d-4bdf-b9a2-7c57837c2c6e",

"status": "Authorized",

"cardNumberLastFour": "5678",

"expiryMonth": 12,

"expiryYear": 2025,

"currency": "USD",

"amount": 100

}

## Models

### 1. PostPaymentRequest

Request model for creating a payment.

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| cardNumber | string | Card number (14-19 numeric characters). |
| expiryMonth | int | Expiry month of the card (1-12). |
| expiryYear | int | Expiry year of the card (e.g., 2025). |
| Currency | string | The currency code (EUR, USD, GBP). |
| amount | int | The transaction amount (integer). |
| cvv | string | CVV of the card (3-4 numeric characters). |

### 2. PostPaymentResponse

Response model for payment creation and retrieval.

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| Id | Guid | Unique identifier of the payment. |
| status | string | Status of the payment (e.g., Authorized, Declined). |
| cardNumberLastFour | string | Last four digits of the card number. |
| expiryMonth | Int | Expiry month of the card. |
| expiryYear | Int | Expiry year of the card. |
| currency | string | Currency code of the payment. |
| amount | Int | The amount for the transaction. |

### 3. ProblemDetailsResponse

The ProblemDetailsResponse model is used to return detailed error information when a request fails due to validation errors or other issues. It follows the general structure of

RFC 7807 Problem Details for HTTP APIs

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| type | string | A URI reference that identifies the problem type. It provides more details. |
| title | string | A short, human-readable summary of the problem. |
| status | int | The HTTP status code generated by the server for this error. |
| traceId | string | A unique identifier for the specific request, useful for debugging. |
| errors | Dictionary<string, string[]> | A dictionary of validation errors. Each key represents a field name, and its value is an array of error messages. |

An example where credit card number is missing in PostPaymentRequest:

{

"type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",

"title": "One or more validation errors occurred.",

"status": 400,

"traceId": "00-d7ddcb68b1f5fe0a90173574a0732249-d65aa060a310dd78-00",

"errors": {

"CardNumber": [

"Card number is required."

]

}

}

## Error Handling

1. **400 Bad Request:** Validation errors in the input (e.g., invalid card number or CVV).
2. **404 Not Found:** Payment with the specified ID does not exist.
3. **500 Internal Server Error:** Unexpected server-side issues.

# Design Considerations

**1. Validation of Input Data**

* **Decision:** All payment request fields (e.g., card number, expiry date, currency, amount, CVV) will be strictly validated to ensure compliance with business rules.
* **Rationale:** Validation prevents invalid requests from being processed by the acquiring bank, reducing unnecessary errors and improving system reliability.
* **Trade-offs:** Additional development effort required to implement validation logic.

**2. Simulated Acquiring Bank Integration**

* **Decision:** The acquiring bank component will be simulated to allow end-to-end testing without requiring integration with a real bank.
* **Rationale:** This approach reduces complexity and cost during development while enabling full control over simulated responses.
* **Trade-offs:** Does not test real-world acquiring bank behavior.

**3. Masked Card Number in Responses**

* **Decision:** Responses will only include the last four digits of the card number to ensure compliance with security and privacy regulations.
* **Rationale:** Prevents exposing sensitive cardholder information, adhering to PCI DSS standards.
* **Trade-offs:** Merchants have limited visibility into full card details, which is a necessary compromise for security.

**4. In-Memory Storage for Payments**

* **Decision:** Use an in-memory repository to store payment details temporarily.
* **Rationale:** Simplifies implementation and avoids the need for database setup during the initial phase.
* **Trade-offs:** Data is not persistent and will be lost when the application restarts.

**5. Status Codes and Responses**

* **Decision:** Standard HTTP status codes (e.g., 200, 400, 404) will be used to communicate results, along with meaningful error messages.
* **Rationale:** Ensures consistency and aligns with RESTful API best practices.
* **Trade-offs:** Developers must handle various status codes explicitly in client applications.

**6. Limited Currency Support**

* **Decision:** The system will support a maximum of three ISO currency codes (e.g., USD, EUR, GBP) during the initial phase.
* **Rationale:** Limits complexity and ensures focus on core functionality.
* **Trade-offs:** Merchants dealing in unsupported currencies will need to wait for future updates.

**7. Secure Card Number Handling**

* **Decision:** Card numbers will be stored in secure memory (e.g., using SecureString) and cleared from plain-text variables as soon as possible.
* **Rationale:** Mitigates the risk of exposing sensitive card information.
* **Trade-offs:** Additional complexity in implementing secure string handling.

**8. Error Handling and Logging**

* **Decision:** Comprehensive error handling will be implemented, with detailed logs for debugging and monitoring.
* **Rationale:** Improves system reliability and aids in diagnosing issues during testing and production.
* **Trade-offs:** Increased logging may require attention to ensure sensitive information is not inadvertently logged.

**Assumptions**

**1. Merchant Assumptions**

* Merchants will store the Id returned from the system to retrieve payment details later.

**2. Currency and Amount Assumptions**

* Supported currencies (e.g., USD, EUR, GBP) will remain stable and not require frequent updates.
* Merchants will submit amounts in minor currency units (e.g., cents for USD).

**3. Error Scenarios**

* If the acquiring bank simulation fails, the system will return a 404 Internal Server Error with a meaningful message.
* Invalid requests will return a 400 Bad Request with details about the validation failure.

**4. Security Assumptions**

* Card numbers will not be logged or stored in an unencrypted format at any stage.
* Secure transport protocols (e.g., HTTPS) will be enforced for all API communications.

**Impact Analysis**

**If Assumptions Change:**

* **New Currency Requirements:** Adding more currencies may require validation logic and additional ISO currency support testing.

**References**

* [ISO 4217 Currency Codes](https://www.iso.org/iso-4217-currency-codes.html)
* [PCI DSS Security Standards](https://www.pcisecuritystandards.org/)
* [HTTP Status Codes Documentation](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)