System Design for Payment System for Ecommerce Website

Introduction

The proposed system is designed to handle the financial transactions of an Ecommerce website similar to Amazon. The system will manage the smooth transfer of money between the customer and supplier, via the Ecommerce platform. We need a system that is reliable, scalable, and flexible to cater to different payment methods globally.

Questions to interviewer:

1. **What kind of Payment system that we are building?**
   We are building the backend of an Ecommerce website's payment system. This system will handle all processes from the point a customer places an order till the point they receive the product.

2. **What all payment options supports?**
   The system should support multiple payment options for flexibility and convenience to the customer. These options include Credit Cards, Bank Transfers, and Debit Cards.

3. **Do we need to handle the Payment processing by ourselves or using 3rd party services like Stripe, Square, etc?**
   Implementing a custom payment processing system can be a complex task due to the need for handling security, compliance, and integration with various payment methods. Therefore, it is often more efficient to use third-party services like Stripe or Square, which specialize in these areas and have robust systems in place.

4. **So we don't need to store the credit card information for security and compliance requirements?**
   Correct, storing sensitive credit card information would require stringent security measures and compliance with regulations such as PCI DSS. It is generally more secure and efficient to use third-party payment processors who specialize in securely handling this data.

5. **Does this application go Global? Need to accept other currencies and international payments?**
   Answer: Yes, the system should be designed to handle international payments and support multiple currencies. This would allow the Ecommerce platform to serve customers globally and expand its customer base.

6. **How many transactions per Day? 1 million or 10 Lakh?**
   Answer: For this design, we'll assume the system needs to handle up to 1 million transactions per day. This equates to roughly 11.6 transactions per second, assuming even distribution over 24 hours.

7. **So we need to support cash-in-flow and cash-out-flow?**
   Answer: Yes, the system should handle both cash-in-flow (money coming from customers to the Ecommerce account) and cash-out-flow (money going from the Ecommerce account to the suppliers).

Functional and Non-Functional Requirements

Functional Requirements:
1. Cash-in-flow: Payment system receives money from customers on behalf of sellers
2. Pay-out-flow: Sellers receive the money once the shipping is completed to the customer after deducting the commission & logistics fees from the Ecommerce site

Non-Functional Requirements:
1. Reliability and fault tolerance: System should be capable of handling payment transactions securely and reliably.
2. Reconciliation: System should reconcile transactions at the end of the day to ensure consistency.

Estimate Calculation:

Transactions per Seconds: If we assume 1 million transactions per day and an even distribution of transactions over 24 hours, then we get about 11.6 transactions per second. Therefore, we need a system that can handle this load and store the payment transaction data accurately at this high throughput.

High-Level Design

Components involved in the system are:

1. Payment Event
2. Payment Service with DB
3. Payment Executor with DB
4. Ledger with DB
5. Wallet with DB
6. Payment Service Provider (PSP)

Detailed Design

PSP Integration
To handle actual transactions, the system should connect with a Payment Service Provider (PSP) which could be a third-party service like Stripe, Square, etc., or direct banking channels. The use of a third-party PSP is recommended because it simplifies the process by handling security, compliance, and integration with various payment methods.

The process generally involves the following steps:

1. A user adds items to their cart and clicks the checkout button.
2. The client sends a request to the payment service in the backend with the payment order information, including a unique PaymentID (UUID).

3. The payment service sends a registration request to the PSP, including the amount, currency, date, redirect URL, and orderID details.
4. The PSP creates a token key for that PaymentID and returns it to the payment service. This token is stored in the DB associated with the specific PaymentID.
5. The client is then directed to a hosted payment page provided by the PSP where they enter their payment information. This information is securely stored and processed on the PSP's system.
6. Once the payment is processed, the PSP returns the payment status to the payment service.
7. The user is then redirected to a status page where they can see the outcome of their transaction.
8. If the payment is successful, the user receives an email confirmation and an estimated delivery time for their order.

Reconciliation
Reconciliation is a critical process that ensures the integrity and consistency of transactions. Since the payment service interacts with various components asynchronously, not all responses are guaranteed.

Each night, the PSP shares a file with the payment service containing all transactions processed during the day. A reconciliation process parses this file and verifies each transaction, checking the amounts deducted from the customer's account and transferred to the seller's account. If any discrepancies are detected, an alert is sent to the admin for further investigation.

Handling Payment Processing Delays
In some cases, payment transactions might take longer to process due to various reasons. To handle this, the system should have a mechanism to update the user about the status of their transaction and any pending actions required from their side.

Communication among Internal Services
The payment system interacts with various internal services like analytics, marketing, and billing teams. To efficiently manage these interactions, an asynchronous communication system like Kafka can be used. This helps in sharing updates among different teams and services without blocking the main transaction flow.

Handling Failed Payments
The system should be robust enough to handle failed transactions gracefully. If a transaction fails, the system should have the ability to retry the transaction or initiate a refund process. All failed transactions should be tracked, and if the number of failed attempts crosses a certain threshold, they should be moved to a Dead Letter Queue (DLQ) for further analysis and manual intervention.

Consistency & Idempotency

To prevent double charging due to user actions (like clicking the checkout button multiple times) or network errors, the system should implement an idempotency mechanism. When a token is created for a transaction, a unique idempotency key (UUID) is associated with it. This key is stored both in the PSP and the payment service's database. If the user initiates multiple requests, only one is processed based on the idempotency key.

Security
Security is of utmost importance in any payment system. Various measures should be implemented to ensure the security of transactions:

1. HTTPS should be used for all request-response communication to ensure the data transferred is encrypted.
2. A rate limiter should be used to prevent any abuse by limiting the number of requests a user can send within a specific timeframe.
3. A firewall should be in place to prevent Denial of Service (DoS) attacks.
4. Data should be replicated

APIs for Payment Service

1. POST /v1/payments - This endpoint will execute a payment event.
2. GET /v1/payments/:id - This endpoint gives the status of a single payment order.

Database Design

Two main tables will be used:
1. Payment Event: Stores information about each payment event.
2. Payment Order: Stores information about each payment order.

As this is a financial use case, the system should be ACID compliant, thus a relational database is suitable.

Conclusion

The designed system will be capable of handling a large volume of transactions securely and efficiently. It will support different payment methods and provide robust mechanisms for handling failures and ensuring consistency. With proper security measures in place, the system will provide a reliable and scalable solution for handling the payment processing needs of the Ecommerce platform.