

Practical Guide to Payment Engineering for SaaS/Fintech

This document is a hands-on, implementation-focused guide for building robust payment systems in a SaaS or fintech environment. It covers card authorisation, capture, multi-provider strategies, SCA, ledgering, reconciliation, and compliance.

1 . Authorisation vs Capture

Authorisation: - Temporarily reserves funds on a customer card - Confirms card is valid and functional
Does not transfer money yet

Capture: - Confirms transaction - Triggers fund movement from issuer → acquirer → merchant
immediate or delayed (common in ecommerce or hotels)

Pseudocode:

```
payment = gateway.authorise(amount, card)
if payment.authorised:
    gateway.capture(payment.id)
```

2 . Implement Stripe / Adyen with Webhooks (Sandbox)

Stripe Example:

```
// Backend endpoint
app.post('/webhook', verifySignature, async (req, res) => {
  const event = req.body;
  switch(event.type) {
    case 'payment_intent.succeeded':
      markPaymentCaptured(event.data.object.id);
      break;
    case 'payment_intent.payment_failed':
      markPaymentFailed(event.data.object.id);
      break;
```

```
        }
        res.send(200);
    });
}
```

Adyen Example: - Subscribe to notifications via Adyen Dashboard - Use HMAC verification for webhooks
- Update internal payment state based on notification event

Tips: - Always verify signatures - Handle idempotency - Log events for audit

3 . Simulate SCA Flow with 3 DS 2

Flow:

1. Customer submits card
2. Payment gateway determines SCA required
3. Redirect to issuer's 3DS2 flow
4. Customer authenticates (bank app, OTP, biometrics)
5. Gateway receives authentication result
6. Webhook confirms payment authorised

Pseudocode:

```
if payment.requiresSCA:
    redirectToBankApp(payment.authUrl)
await webhook.payment_authorised
updateOrder(payment.id)
```

4 . Multi-Provider Abstraction

Purpose: - Support multiple payment providers - Fallback on failures - Optimize for fees and coverage

Interface:

```
interface IPaymentGateway {
    Authorise(request): PaymentResult
    Capture(paymentId)
```

```
    Refund(paymentId)  
}
```

Implementation Example:

```
providers = [StripeGateway(), AdyenGateway()]  
function routePayment(payment):  
    for provider in providers:  
        try:  
            return provider.authorise(payment)  
        except TemporaryFailure:  
            continue  
    raise PaymentFailedException()
```

5 . Ledger Implementation with Double-Entry

Principles: - Every transaction has debit & credit - Chargebacks, refunds, and fees reflected separately
Ensure internal books match settlements

Example:

```
PaymentCaptured:  
    debit: MerchantReceivable  
    credit: Revenue  
RefundIssued:  
    debit: Revenue  
    credit: MerchantReceivable  
ChargebackReceived:  
    debit: ChargebackLoss  
    credit: MerchantReceivable
```

Audit Tip: - Store original transaction ID in each ledger entry - Maintain timestamp and event reference

6 . Chargebacks and Refunds

Chargeback Flow: 1 . Customer disputes with issuer 2 . Issuer pulls funds via network 3 . Merchant receives notification 4 . Submit evidence or lose funds

Refund Flow: - Merchant initiates refund - Gateway sends reversal to issuer - Ledger updates independently

Pseudocode:

```
function refund(paymentId, amount):
    gateway.refund(paymentId, amount)
    ledger.recordRefund(paymentId, amount)
```

7 . Optimizing Reconciliation, Idempotency, Reliability

Idempotency:

```
function authorise(request):
    if request.idempotencyKey in db:
        return db[request.idempotencyKey]
    response = gateway.authorise(request)
    db[request.idempotencyKey] = response
    return response
```

Reconciliation: - Compare internal ledger, gateway report, and acquirer report - Raise alerts for mismatches

Reliability: - Retry soft declines with exponential backoff - Store webhook events and replay if fails - Use Outbox pattern for event-driven systems

8 . Multi-Provider Routing Strategies

Use-Cases: - Geographic optimization (EU cards → Adyen, US cards → Stripe) - Fallback for downed providers (choose provider with lowest interchange)

Pseudocode:

```
def routePayment(payment):
    if payment.currency in ['EUR', 'GBP']:
        primary = Adyen
```

```

        fallback = Stripe
    else:
        primary = Stripe
        fallback = Adyen

    try:
        return primary.authorise(payment)
    except TemporaryFailure:
        return fallback.authorise(payment)

```

9 . Regulatory Compliance

PCI DSS: - Never store full card numbers - Use tokenization - Encrypt sensitive data

PSD 2 /SCA: - EU/UK online transactions require 2-factor authentication - 3DS2 handles bank-side authentication

Open Banking: - Customer bank login and approval for account-to-account transfers - Provides instant confirmation, reduces fraud, no chargebacks

10 . Recommended Practice Path

- 1 . Implement Stripe sandbox with webhooks
 - 2 . Add SCA/3DS2 flow simulation
 - 3 . Integrate second provider for multi-provider testing
 - 4 . Implement double-entry ledger
 - 5 . Handle refunds and chargebacks end-to-end
 - 6 . Add idempotency and reconciliation logic
 - 7 . Test multi-provider routing and failover
 - 8 . Ensure regulatory compliance (PCI, PSD 2)
-

Summary

This guide provides practical pseudocode, architecture patterns, and compliance rules to engineer robust production-grade payment systems suitable for SaaS and fintech environments. It bridges theory and implementation for interview and real-world readiness.