Payments Engineering

Lessons Learned in Writing Software for Payments
● **What is payments engineering?**
  ○ Overview
  ○ What’s a payment (system) ?
  ○ Credit Cards

● **Problem Solving in Payments Software**
  ○ Reconciliation and auditing
  ○ Double charges
  ○ Testing and observability

● **Q&A**
What is Payments Engineering?

Building software in the problem domain of payments. Working with or building payment systems.

- Understanding real payment systems
- Complex state management
- Security and privacy
- Precision, correctness
- Auditability, reconciliation
- Fault tolerant
- Expensive mistakes
What’s a Payment Anyway?

Payment: “A transfer of value between a sender and a receiver, denominated in some currency.”[1]

What’s a Payment (System) Anyway?

Payment System: “Connects senders and receivers, and provides framework for transferring value.” [1]

Some examples:

- Credit Cards: Visa, Mastercard
- Bank transfers - ACH, wires
- Cash
- Digital currencies

Credit Cards

- Pretty old idea
- Modern cards still modeled on processes that pre-date computers

https://www.thebalancemoney.com/history-of-credit-cards-4766953
Credit Cards

Credit Cards

https://www.nytimes.com/wirecutter/blog/your-mesopotamian-credit-card/
Credit Cards: a Beautiful State Machine
Credit Cards: Parties Involved

- Credit Card Networks (Visa, MasterCard, AMEX, etc)
  - Issuing Bank
    - Card Holder
  - Acquiring Bank
    - Merchant
Credit Cards: Authorize
Credit Cards: Authorize
Credit Cards: Authorize
Credit Cards: Authorize
Credit Cards: Authorize
Credit Cards: Capture
Credit Cards: Settlement
Credit Cards: Unhappy path(s)
Credit Cards: Unhappy path(s)
Problem Solving in Payment Software
Modeling Payments

- Long, complex life cycles
- Things happen at a point in time
Problem: Reconciliation and auditing

- How much money did we make in February?

<table>
<thead>
<tr>
<th>id</th>
<th>date</th>
<th>amount</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>2024-02-01</td>
<td>$100.00</td>
<td>settled</td>
</tr>
<tr>
<td>1002</td>
<td>2024-02-03</td>
<td>$20.00</td>
<td>settled</td>
</tr>
<tr>
<td>1003</td>
<td>2024-02-04</td>
<td>$300.00</td>
<td>settled</td>
</tr>
<tr>
<td>1004</td>
<td>2024-02-05</td>
<td>$150.00</td>
<td>settled</td>
</tr>
<tr>
<td>1005</td>
<td>2024-02-07</td>
<td>$10.00</td>
<td>settled</td>
</tr>
</tbody>
</table>
Problem: Reconciliation and auditing

- How much money did we make in February?

<table>
<thead>
<tr>
<th>id</th>
<th>date</th>
<th>amount</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>2024-02-01</td>
<td>$100.00</td>
<td>settled</td>
</tr>
<tr>
<td>1002</td>
<td>2024-02-03</td>
<td>$20.00</td>
<td>settled</td>
</tr>
<tr>
<td>1003</td>
<td>2024-02-04</td>
<td>$300.00</td>
<td>settled</td>
</tr>
<tr>
<td>1004</td>
<td>2024-02-05</td>
<td>-$150.00</td>
<td>refunded</td>
</tr>
<tr>
<td>1005</td>
<td>2024-02-07</td>
<td>-$10.00</td>
<td>charged back</td>
</tr>
</tbody>
</table>
**Problem: Reconciliation and auditing**

- Things happen at a point in time. Record facts.

<table>
<thead>
<tr>
<th>id</th>
<th>date</th>
<th>amount</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>2024-02-01</td>
<td>$100.00</td>
<td>settled</td>
</tr>
<tr>
<td>1002</td>
<td>2024-02-03</td>
<td>$20.00</td>
<td>settled</td>
</tr>
<tr>
<td>1003</td>
<td>2024-02-04</td>
<td>-$300.00</td>
<td>refunded</td>
</tr>
<tr>
<td>1004</td>
<td>2024-02-05</td>
<td>-$150.00</td>
<td>charged back</td>
</tr>
<tr>
<td>1005</td>
<td>2024-02-07</td>
<td>$10.00</td>
<td>settled</td>
</tr>
</tbody>
</table>
Solution: Immutable architecture

- Git
- Ledgers
- Kafka
Solution: Immutable architecture

https://www.reddit.com/r/linuxmemes/comments/17n0nzb/the_holy_trinity

ty https://xeiaso.net
Problem: Avoiding double charges

- “Don’t click twice”
- “Something went wrong”
The Network is Unreliable

- Problems w/ distributed systems
  - “Fallacies of Distributed Computing (L Peter Deutsch, James Gosling ~ 1996)
- 99% - 99.9% success rate?
Solution: Idempotency

- **POST Once Exactly (POE) IETF 2005**
- Idempotency Keys (Stripe, AWS, etc)

Idempotency Keys

hash(user_id, order_id, amount) ->
fade33ef5120fdd0da8c44cc7cfd0bb21a5974272112d2aff410ceaf0445187a

OR

uuid4() -> ab1eaf28-e0f1-1lee-93cd-2f34b72080a7
Idempotency Keys

```bash
curl https://api.stripe.com/v1/charges \
  -u sk_test_BQokikJ0vBiI2HlWgH4olfQ2: \
  -H "Idempotency-Key: AGJ6FJMkGQIpHUTX" \
  -d amount=2000 \
  -d currency=usd \
  -d description="Charge for Brandur" \
  -d customer=cus_A8Z5MHwQS7jUmZ
```

https://stripe.com/blog/idempotency
Click to your Heart’s Content
Fundamentals - Testing

Your code will get tested, one way or another.

- Unit Tests
- Integration
- UI / End-to-End
Unit Tests

- Fast (seconds)
- Isolated
-Mocks and Stubs
- Domain/Business Logic
Integration Tests

- Slower than unit tests but should still be fast
- Test multiple components working together
- Possibly from real infra
UI / End-to-End

- Slower, sometimes fragile
- Real calls against staging environments
- Go through entire “flows”
Fundamentals - Logging and monitoring

- Log enough to be useful
- Avoid logging entire objects
- Understand your key metrics
Q&A