Currency Fair

Engineering Test

Purpose

Lay out a high-level design for a 3 component exchange system that illustrates key design principles and choices.

Implement key components to showcase the design.

Goal

Provide insight into my thought process, design acumen, and skill set in a bid to join the Currency Fair engineering team.

Design Considerations

Message Consumption

- Secure
- Fast
- Scale horizontally
- Ensure xtn integrity
- Enforce data integrity
- Safe

Message Processing

- Efficient
- Scale independent of other tiers
- Scale vertically and horizontally
- Maintain xtn integrity

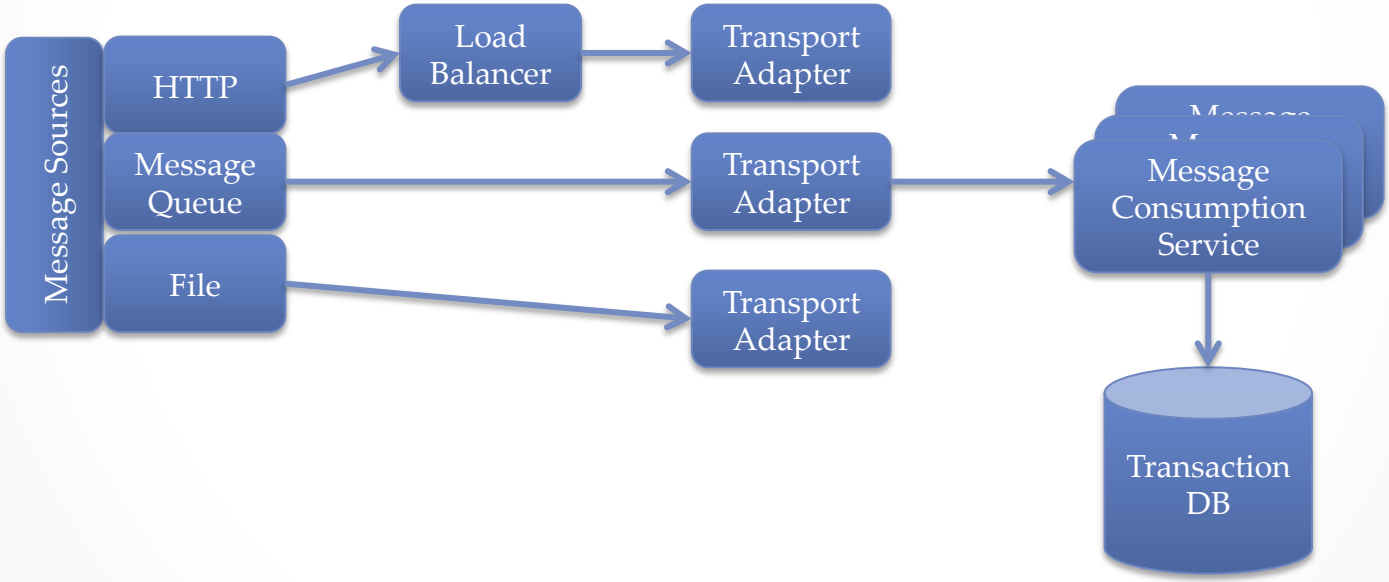
Message Frontend

- Flexible
- Support multiple channels
- Low impact to xtn system
- Real-time

Message Consumption Implementation

- Implement over secure channel taking into account AuthZ and AuthN considerations
- Design to support multiple Qualities of Service (QoS) through multiple channel technologies
- Ensure data validity
- Capture meta-data with message
- Transactional data store
- Strict transaction management to ensure data integrity
- Stateless to minimize footprint for scalability and maximize performance
- Software technology - PHP

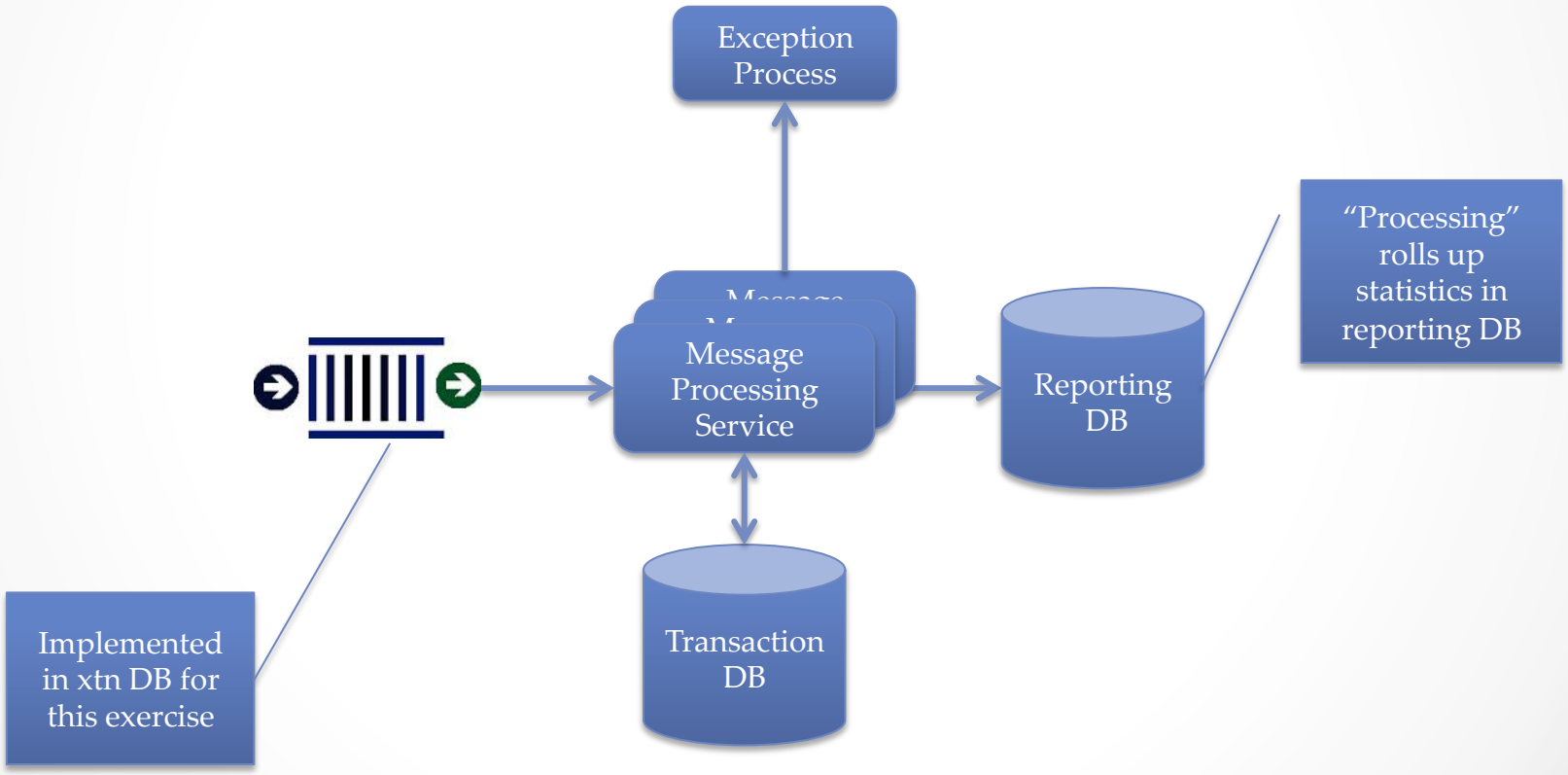
Message Consumption Logical Components



Message Processing Implementation

- Asynchronous
  - Decoupled from message consumption
  - Real-time vs. near real-time requirements are a consideration here
  - Assumes processing is an expensive operation
- Transaction integrity maintained
- Queuing often a good option to achieve high QoS and asynchronous processing
  - Queuing via MQ product or database
- Mechanism to handle exception processing
- For this exercise "processing" performs roll-up in simple "star" reporting schema.
- Software technology - Java

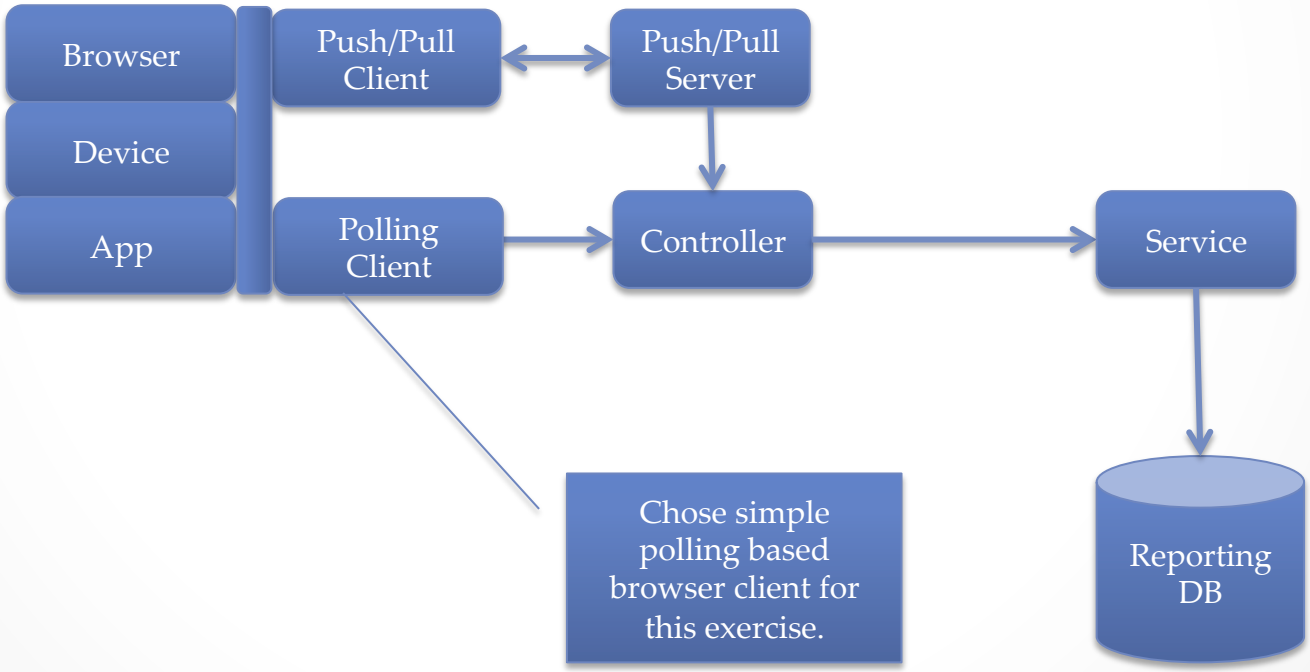
Logical Components



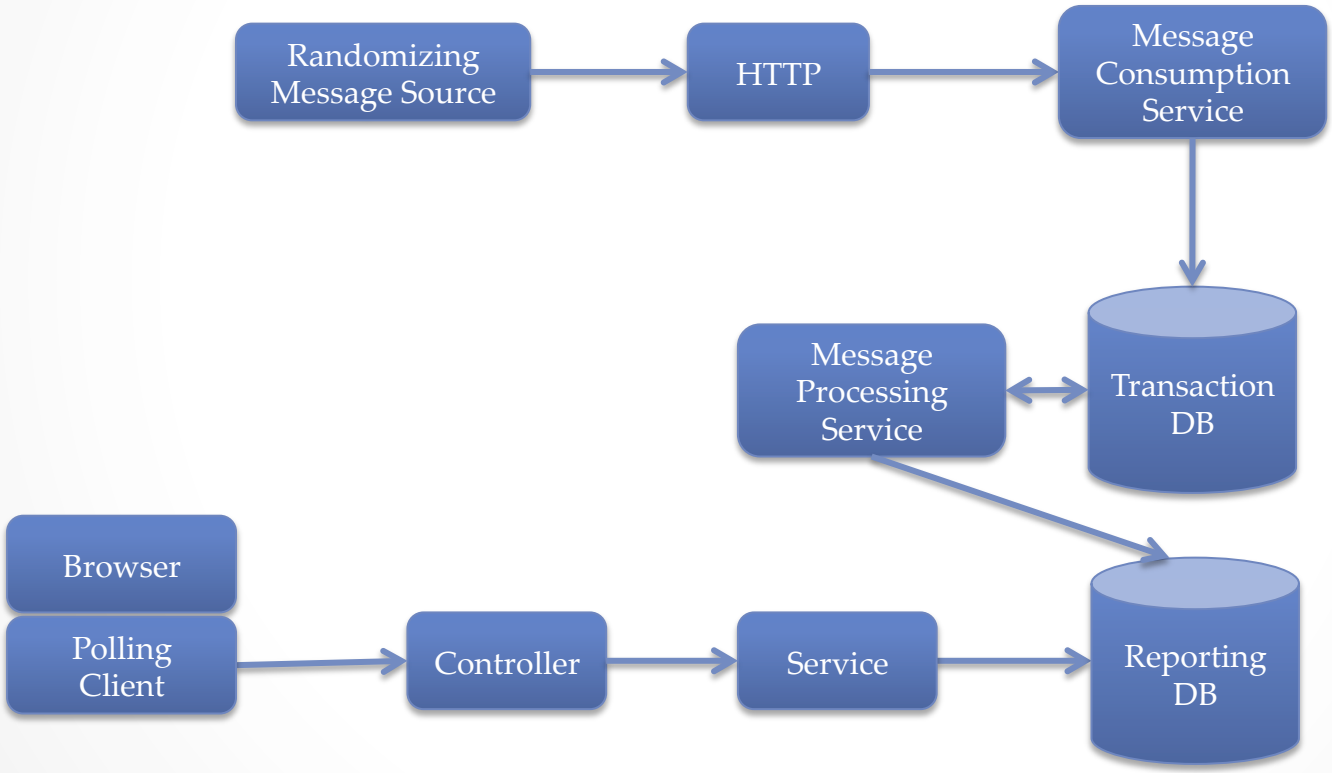
Message Frontent Implementation

- Many options available
- RESTful(-esque) services to maintain flexibility
- MVC paradigm
- Chose simple polling client for brevity for this exercise
- Software technology - PHP

Message Consumption Logical Components



Components Implemented for the Test



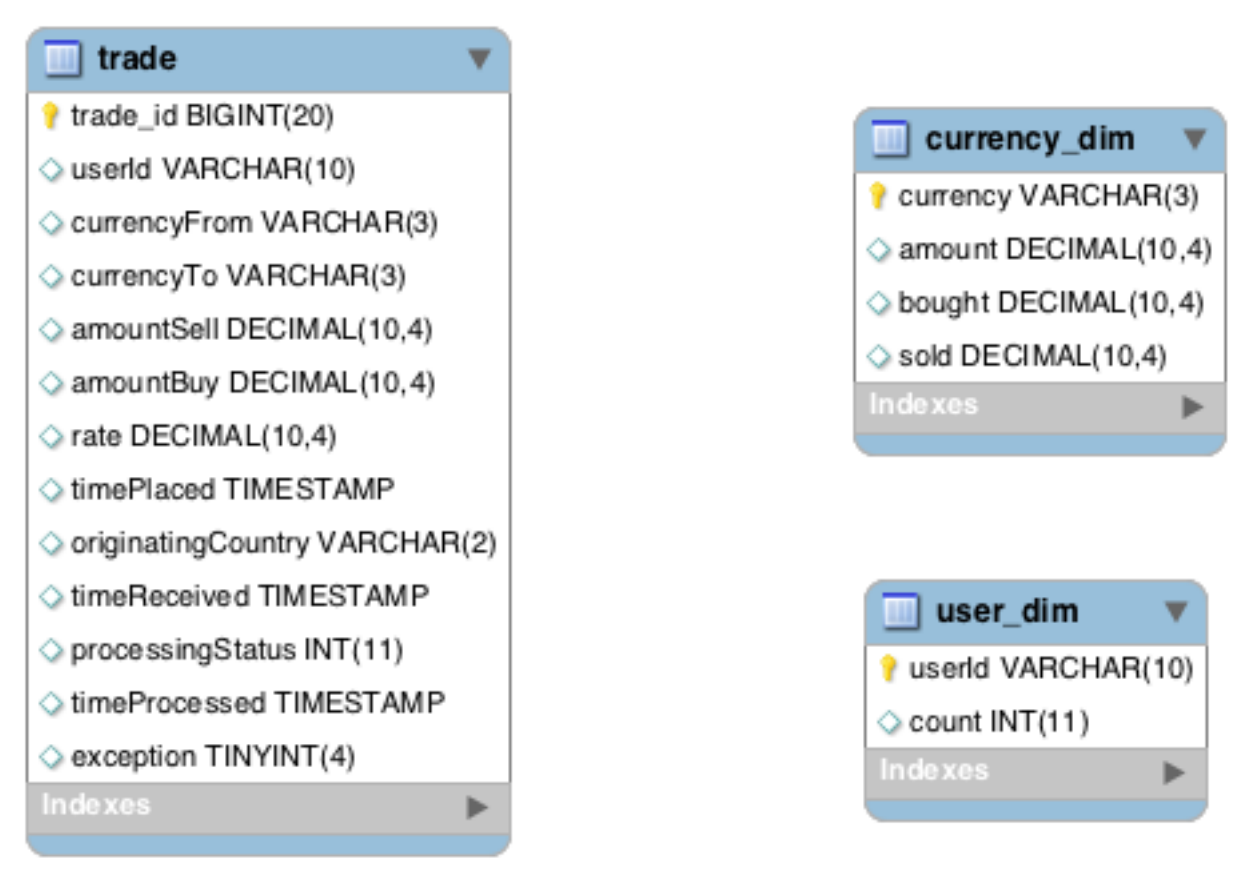
Fine Print

- Example is far from a complete system, attempted to show what's possible and where key implementation points exist
- Implemented as a proof of concept:
  - Lots of room for improvement
  - Did not use any frameworks, i.e. Spring, ORM, Logging, etc.
  - Could be more OO
  - Some liberties taken with respect to configuration, connection pooling, etc.
  - Frontend implementation less fancy than originally planned
- Did not include security or all necessary transactional specifics although showed where these could be considered
- Did not have environment to publicly host the project but all components are functional
- Did not implement any automated unit or integration tests
- Built on LAMP architecture

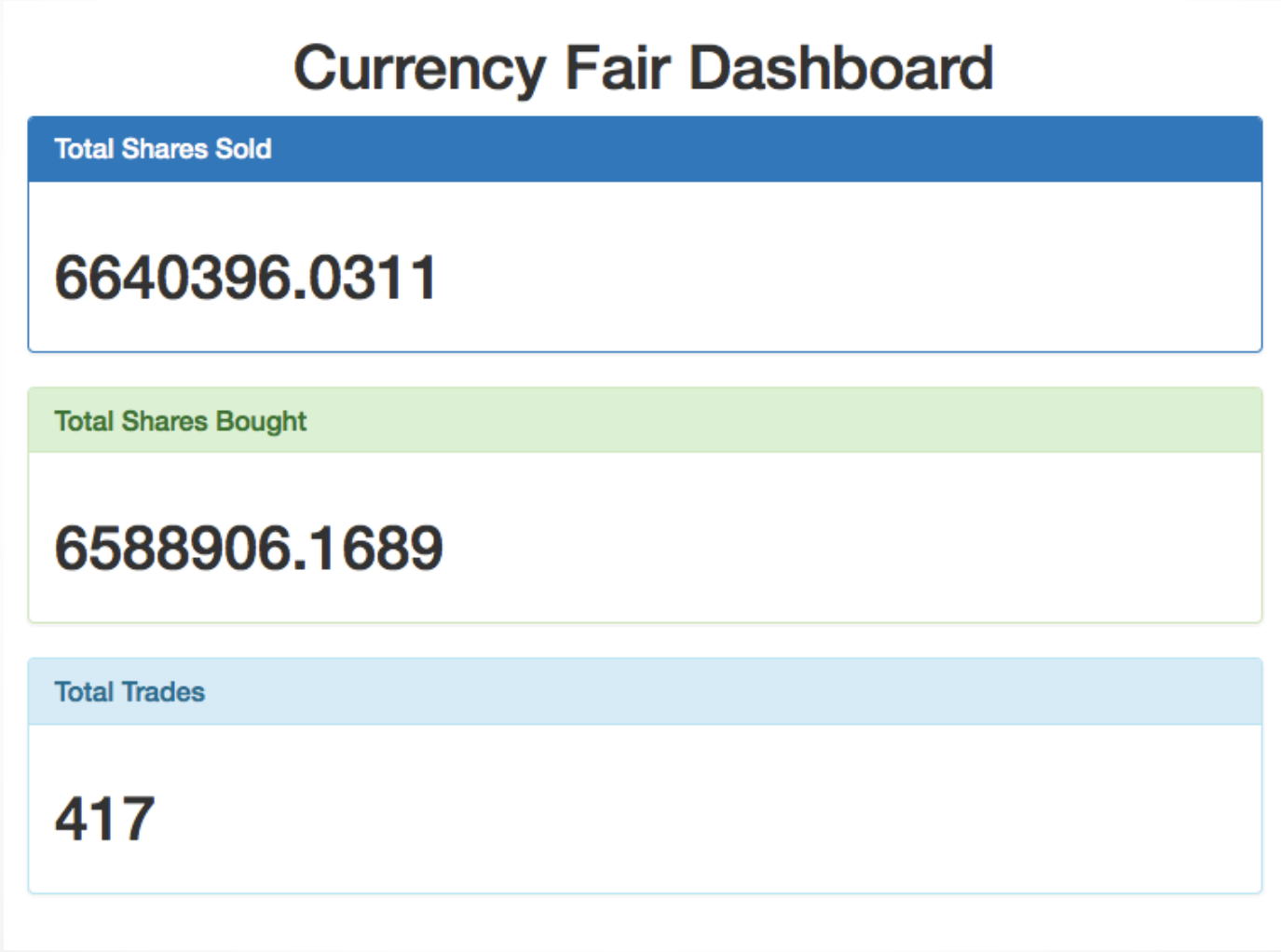
GitHub Directory Structure

- /documentation
  - This overview
  - Readme.txt installation instructions
- /data
  - SQL script to create and load schema and sample data
- /src/messageprocessor
  - Java maven project with Main class
  - Requires MySQL client JAR as shown in pom.xml
- /src/test/client
  - Bash script to simulate client messages
  - Randomizes sample data
  - Uses cURL to POST JSON data to the message consumer
- /src/web_root
  - Contents can be copied straight to PHP-enabled web server

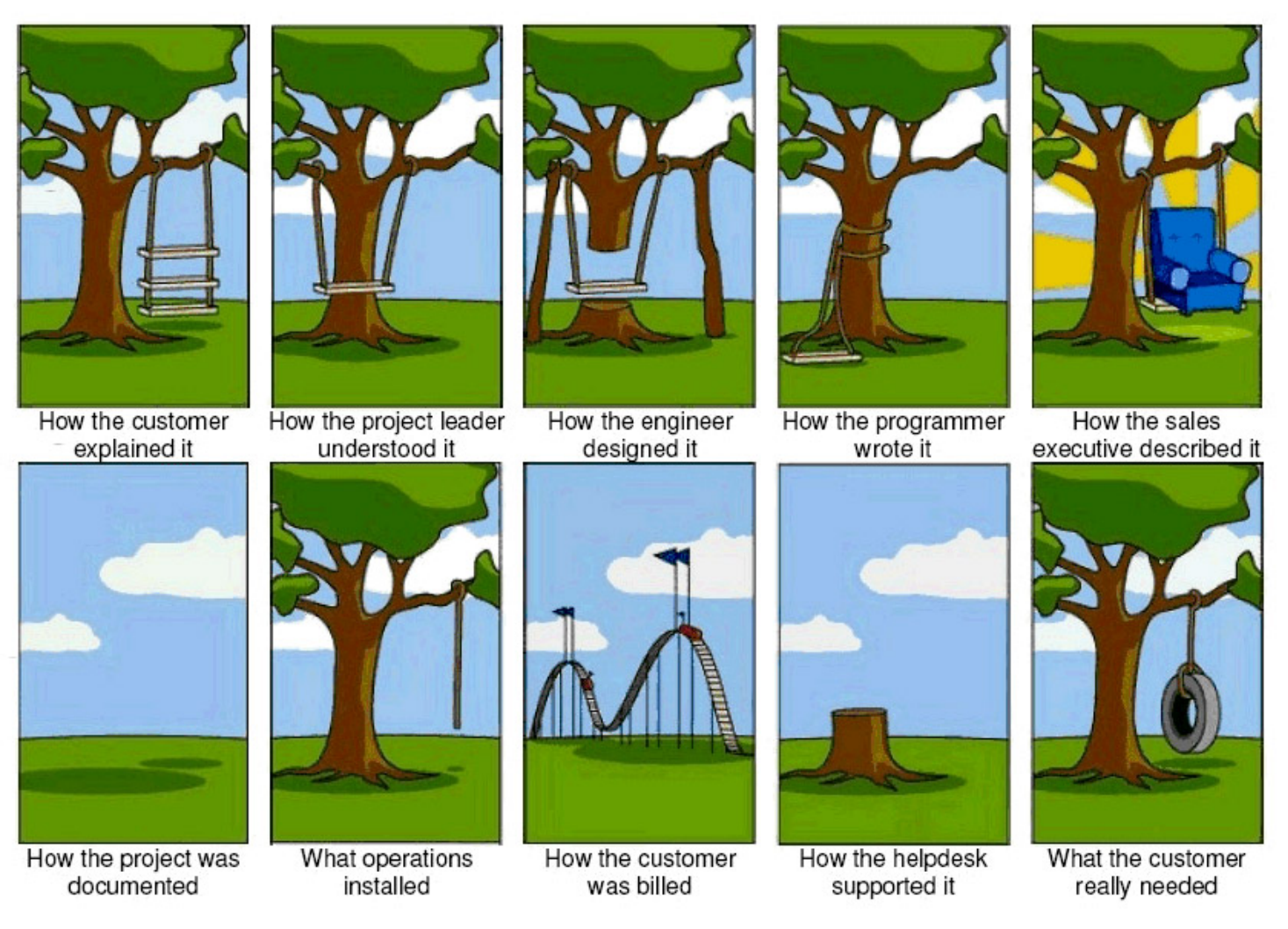
Database Schema



Frontend Screenshot



For Fun


How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed