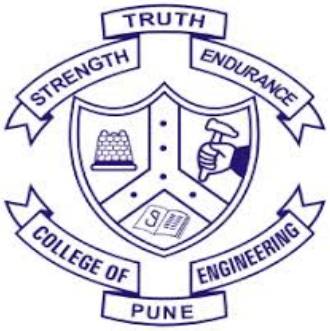

Finance Project

College of Engineering, Pune

Department of Computer Engineering and Information Technology

Trade and Position Management

Author:

Group#5

Supervisor:

Credit Suisse Team

Contents

Chapter 1

Introduction

1.1 Purpose

The document will outline in detail the design specifications for **Trade and Position Management (TPM)**. The purpose is to facilitate understanding of the system and communication between different modules/ services.

1.2 Intended Audience

The document is intended to be used by the **members of the Project team** that will implement the module and **Credit Suisse staff** which will verify the correct functionality of the system. In addition, other **COEP teams** can understand details of TPM through this document.

Chapter 2

Overview

2.1 To-Do

To create and maintain two collections **'TRADE'** and **'POSITION'**

2.2 Requirements

'ORDER' and **'FILLS'** collections from **Group#3**.

2.3 Outline

- To receive a notification from **Group#3** when there is a new entry in the **FILLS**.
- To generate **TRADE** document from corresponding **ORDER** and **FILLS** documents.
- To perform mathematical computations on data in **TRADE** document to generate **POSITION** document; also, take real-time data *(market_price)* from **Group#2**.
- Maintain **TRADE** and **POSITION** collections.
- On receiving request from **Group#1**, send **TRADE** and **POSITION** details regarding a certain client to **Group#1**.

Chapter 3

Design Specifications

3.1 Assumptions and Dependencies

- Static reference data is provided by Credit Suisse.

3.2 Technologies

3.2.1 MongoDB

- Data is replicated thrice for Recoverability.
- Provides Scalability.
- Gives Availability.

3.2.2 Flask

- Receiving notifications through flask-blinker-library.
- Signals in flask help decouple applications by sending notifications when actions occur elsewhere in the core framework or another flask extensions.
- In short, signals allow certain senders to notify subscribers that something happened.

RDBMS	MongoDB
Table	Collection
Column	Key
Value	Value
Records / Rows	Document / Object

Figure 3.1: Correspondence between RDBMS and MongoDB *(for understanding):*

3.2.3 Python

- Development language.
- SQLAlchemy is an object-relational mapper widely used in the Python world, making it easier (usually!) for developers to interface with their database of choice.

3.3 Design Details

- **Group#3** gives notification when there is a new entry in FILLS collection. Notification is received through Django-Watcher.
- On receiving notification, query the database of **Group#3** to get the new entries from ORDER and FILLS collections. Database being used is MongoDB and query result will be in JSON format.
- **create_trade()** uses data from these two documents and creates an entry for TRADE collection. Once TRADE document is created, call following functions:
  - **validate()** - price should be greater than zero; user, book etc should be valid *(maintain static sample data for validation).*

  - **figuration()** - calculate commision (either basis points, flat or cents per share) according to the client (relation between client and commission type to be stored in static file).
    * $basis\_point = qty * price * rate/10000$
    * $flat = amt$
    * $cents\_per\_share = \#shares * 10$

- For POSITION document, use data in corresponding TRADE document and call following functions:
  - realised_PL() -

Realised PL			
Trade Direction	Pre-trade Position Direction	Post-trade Position Direction	Calculation
Buy	Long or Flat	Long	realised PL=0
Sell	Short or Flat	Short	realised PL=0
Buy	Short	Short or Flat	realised PL=(trade price - avg price)*trade volume
Sell	Long	Long or Flat	realised PL=(trade price - avg price)*trade volume
Buy	Short	Long	realised PL=(trade price - avg price)*position pretrade
Sell	Long	Short	realised PL=(trade price - avg price)*position pretrade

Table 3.1: Realised PL

  - unrealised_PL() - net position * market price
  - net_position() - amt of stocks left
  - market_price() - query **Group#2** to get market_price

– avg_price() -

Average price calculations			
Trade Direction	Pre-trade Position Direction	Post-trade Position Direction	Calculation
Buy	Long	Long	(average price * position pretrade + trade price * trade volume)/position post trade
Sell	Short	Short	(average price * position pretrade + trade price * trade volume)/position post trade
Buy	Short	Short	average price = average price pretrade
Sell	Long	Long	average price = average price pretrade
Buy	Short or Flat	Long	average price = trade price
Sell	Long or Flat	Short	average price = trade price
Either	Either	Flat	average price = 0

Table 3.2: Average price

- **amend()** - Receive notification if any changes occur in existing entries of ORDER or FILLS documents and amend respective entries in TRADE and POSITION documents accordingly.

- **cancel()** - If **Group#3** sends a cancellation notification, remove document from TRADE and POSITION collections.

- **send_details()** - On receiving a ping/ request from **Group#1**, call this function and send TRADE and POSITION details of certain client back to **Group#1**.

- **Example** -
  - Account A1

    To Buy 10 shares (Rs.1/share) when initially there are none i.e. Flat→Long

    Hence, Net position = 10

    Average price = *Trade price* = Rs. 1/share

    Realised PL = 0
  - Account A1

    To Buy 20 shares (Rs. 2/share) when initially the net position is 10 i.e. Long→Long

    Hence, Net position = 30

    Average price = *(average price * position pretrade + trade price * trade volume)/position post trade* = (1*10 + 2*20)/30 = 1.66

    Realised PL = 0
  - Account A1

    To Sell 15 shares (Rs.3/share) when initially the net position is 30 i.e. Long→Long

    Hence, Net position = 15

    Average price = *average price pretrade* = 1.66

    Realised PL = *(trade price - avg price)*trade volume* = (3-1.66)*15 = 20.1

3.4 Database Schema

Order Document	
Fields	Type/ Possible Values
Order_id	string
Client_id	string
Side	Buy/ Sell
Product_id	string
Size	int
Asked_price	int
Order_stamp	date
Reason_for_cancellation	string
State	Live/ Filled/ Cancelled/ Closed/ Rejected

Table 3.3: Order

Fills Document	
Fields	Type/ Possible Values
Order_id	string
Fill_id	int
Quantity_size	int
Price	float
Exchange_id	int
Exchange_stamp	date
Counter-party	string

Table 3.4: Fills

Trade Document	
Fields	Type/ Possible Values
Order_id *(fk)*	string
Client_id	string
Trade_id	int
Fill_id	int
Quantity_size	int
Price	float
Exchange_id	int
Order_stamp	date
Exchange_stamp	date
Trade_stamp	date
Counter-party	string
Commision	int

Table 3.5: Trade

Position Document	
Fields	Type/ Possible Values
Order_id *(fk)*	string
Client_id	string
Realised_PL	float
Unrealised_PL	float
Net_position	int
Avg_price	float
Market_price	float

Table 3.6: Position

Chapter 4

Diagrams

4.1 Class Diagram

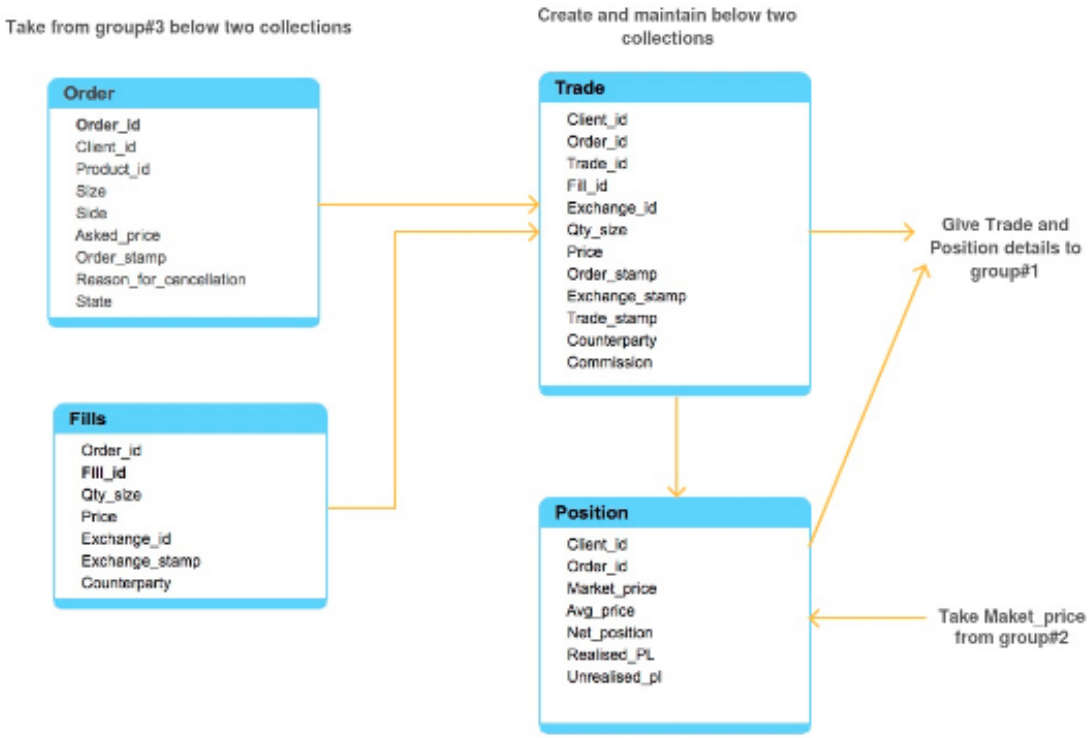

Figure 4.1: Class Diagram

4.2 Module Architecture

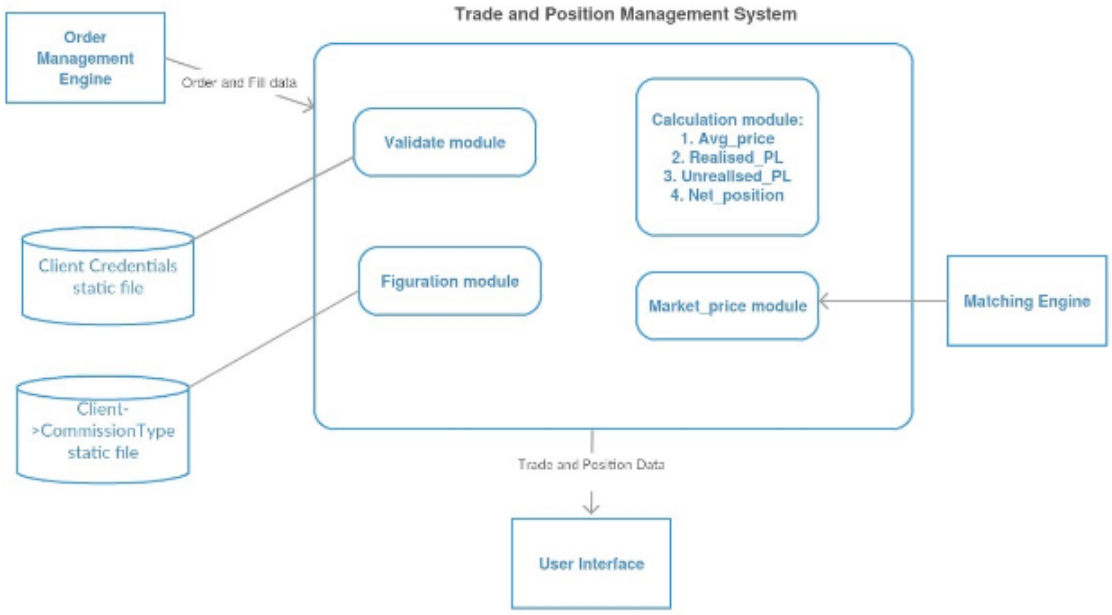

Figure 4.2: Module Architecture

Chapter 5

References

- Middle Office Functions-High Level Requirement.docx
- http://flask.pocoo.org/
- http://flask.pocoo.org/docs/0.12/signals/
- https://www.michaelcho.me/article/sqlalchemy-commit-flush-expire-refresh-merge-wh
- https://docs.mongodb.com/
- https://docs.mongodb.com/manual/reference/sql-comparison/