SENG 468 - Software System Scalability

Course Project

The End-To-End Development and Analysis of a Production Day Trading System

Important Dates

See the SENG 468 Bright space for all due dates and the full list of project milestones and deliverables throughout the term.

Note:

- Final project submission will be due before finals start

Project Overview

Your consulting group has been contracted by DayTrading Inc. to develop a prototype end-to-end solution to support their daily trading clients and services. DayTrading Inc.'s competitive advantage comes from providing its clients with the fastest end-to-end processing times in the industry and it is seeking to improve its competitive advantage in this area through re-implementing its complete internet-based distributed day trading system. DayTrading is soliciting proof-of-concept solutions from multiple consulting groups and will award the final contract to the system that has

minimal transaction processing times, supports all required features and does so at a minimal cost, in terms of up-front hardware costs, development costs, and on-going maintenance and upgrade costs. DayTrading's client base is continually growing so they are also very interesting in capacity planning issues and in a solution that has the capability to grow incrementally without adversely impacting transaction performance, and where such growth can be supported at minimal cost increments. DayTrading prides itself in providing its clients with the industry-leading security; therefore, all bidders must fully vet and document their system with regards to security issues. System downtime is also not acceptable to DayTrading Inc. and system availability is a key design concern. The winning bid will also have its prototype development costs rolled into the overall system development contract and budgets, so full tracking of the work effort required to design, implement, test, analyze and document the prototype is required.

Your group's task is to build a prototype system for DayTrading Inc.'s evaluation and develop the complete documentation for your solution. This documentation must formally cover all issues of concern to DayTrading Inc., who are well known in the industry for having a strong technical review process and for insisting that all claims be formally verified and supported through formal analysis and testing. For example, capacity planning estimates must be supported through solid experimental testing and analysis of the prototype system. It is insufficient merely to claim a certain level of capacity is achievable through extrapolating incomplete testing scenarios. Included in your system must be a web application which provides both a user interface to your backend day trading system as well as a local store of the current state of the given user's account information.

Outside of the suggested list of software components that DayTrading currently supports in-house, there are no restrictions on the system design (including language choices, database server, web server, system architecture, etc.). Each design team is free to implement the best solution in the manner that they deem best. DayTrading does insist that all design decisions be fully and formally documented including their rationale as to why the given choice was the best available.

It is important to realize that the scope of this project has been intentionally left quite open to enable each group to explore the design space of the problem and to develop their best solution to the problem posed. Hence, issues such as error conditions that need to be specified, account management processes, etc. have been left purposely undefined within the scope of this document. As such, the project closely mimics the level of project specification and detail available in the real-world in the development of commercial projects for actual clients. It is expected that as the project progresses questions will arise about Day Trading's implementation requirements. Such questions can be, and are expected to be, brought up during class or via email as they arise.

Distributed systems and the internet are a complex topic covering a wide variety of issues and concepts. It is hoped that the course project will form a common focal point within which class discussions about the implementation, development and engineering of such systems can take place. Gaining an understanding of the issues inherent in the development of these types of systems requires hands-on experience which the course project is designed to facilitate.

Group Composition

Each group may consist of a maximum of 4 or 5 members. You should choose the group with experience in each of the core project areas, namely: database design, web site development, project infrastructure setup, system security. It is likely no one will be experienced within the areas of performance testing and capacity planning of distributed systems. There are core issues within the project as they drive the iterative process of identifying and rectifying performance bottlenecks. Hence, each team member will want to ensure that they understand and gain skills in these areas. A sizable portion of the project will be focused on the testing activities and structuring of the tests to elicit the required performance and capacity planning information.

Groups will also be responsible for their own project planning and setting their own internal milestones. The group should assign/elect someone to take the lead on this effort, which includes ensuring that the workload is shared equitably among group members.

The project accounts for 40% of the course mark, so the required work effort is commensurate with this percentage. If your group leaves the project to the last minute, then expect that your marks will reflect this choice. Project grades will reflect the work effort, thoroughness, and commitment of the group to developing a well-engineered solution to the problem posed.

The workload required to excel at this project is substantial. Work on the project will be required throughout the term. It is strongly recommended that you bring the project early through the rapid prototyping of a basic end-to-end solution. The system can then be expanded through testing and refining as the term progresses. You should expect that significant portions of your solution will likely need to be reworked in order to achieve high transaction throughput.

Day Trading Activity

The basic structure of the day trading system is to support a large number of remote clients through a centralized transaction processing system. Each client logs in through a web browser and then performs several stock trading and account management activities such as:

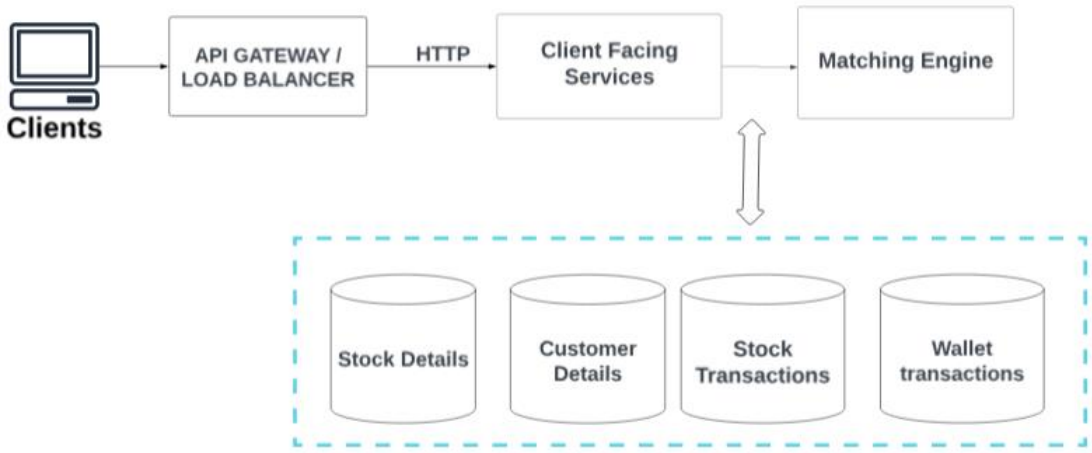
- View their account
- Add money to their account through a wallet
- Get current stock price
- Buy a number of shares in a stock (Market Order)
- Set an automated sell point for a stock (Limit Order)
- Review their complete list of transactions
- Cancel a specified transaction prior to its being completed (Only Applicable to Sell Limit Orders)

In addition to the client activities, DayTrading requires full auditing capabilities; hence, complete transaction logs must be able to be produced on demand that detail of all client activities in the

system (including timestamps of all transactions), a record of each individual transaction, each transaction's processing time information, and all account state changes within the system

Base Architecture

The base architecture for the day trading system is shown in the block diagram below.



The above architecture is meant solely as reference example of what a functional architecture would be. This basic baseline architecture **does not:**

- Address a number of the design issues that you will need to consider in developing your solution.
- Provide acceptable performance, reliability, fault tolerance, etc.

Hence, this is not the architecture that your group should seek to submit as their designed and architected solution.

Overall Architecture Goals:

The overall goals of the developed system and architecture are:

- Minimum transaction processing times.
- Full support for required features.
- Reliability and maintainability of the system.
- High availability and fault recoverability (i.e. proper use of fault tolerance)
- Minimal costs (development, hardware, maintenance, etc.)
- Clean design that is easily understandable and maintainable.
- Appropriate security
- clean web-client interface.

Documentation

DayTrading Inc. requires a complete set of documentation to be delivered at the end of the project in addition to the working prototype.

- Full documentation of architecture including complete analysis of design choices.
- Full documentation of test plans, testing results, and test analyses.
- Full documentation of work effort required to build prototype, including weekly individual logbook entries by each member.
- Complete capacity planning and transaction time documentation, including experimental results and extrapolations.
- Full project planning and execution documentation for prototype development effort.
- Full analysis of system capacity and capacity planning documentation.

All claims within the project documentation must be supported through appropriate experimental testing and analysis

All documents must be clear, concise, and correct with respect to English usage and grammar. Documentation that is not comprehensible, overly verbose, rambling, etc. will be viewed by DayTrading Inc. as indicative of the design team's general level of care and attention and therefore will reflect negatively on the team's overall evaluation.

Front-End Web Site

Each group must maintain a simple functioning front-end application which will be used by the course tutorial instructor to interact with each team's operational system.

Project Planning & Work Effort

DayTrading Inc. will engage in no project planning or work effort enforcement activities. It is left to the individual design teams to structure and enact their own project management plans, including the assignment of individual tasks to group members.

Complete project plans must be submitted to DayTrading Inc. and are to include group development milestone dates as well as an assessment of the risk factors associated with the project.

DayTrading does require each team member to submit individual electronic weekly logbook entries. This entry must specify all the work the team member contributed to the project during the given week, as well as the time devoted to each of the tasks listed.

Supported Component Software

DayTrading Inc. requires that the project be built using Docker Container technologies. All other software choices are open to each group to make, along with the responsibility to install, maintain, come up to speed on, etc. the full technology stack used. All groups must use a full proper code repository for their project.

The only restriction on software and/or coding frameworks that can be used for the project is that frameworks and packaged solutions designed to address or pre-tuned to address system performance requirements cannot be used. If in doubt, your team should check with the course tutorial instructor as to whether any given piece of software and/or software framework can be used as part of the course project.

2. Matching Engine

DayTrading Inc requires the teams to build a custom matching engine that will match customers' orders in real time. The matching engine is the heart of a stock exchange. It receives sell and buy orders from clients and then matches them using a matching algorithm. Although there are several matching algorithms like:

- Price/Time algorithm (or First-in-First-out)
- Pro Rata algorithm

Day Trading Inc requires the FIFO algorithm to be used in which the orders are prioritized according to the time they are received in the matching engine.

At the core of the matching engine lies an orderbook which stores the buy and sell orders separately in different data structures. For example, the buy and sell orders can be stored in two arrays. The arrays need to be sorted based on the price of the order first (if a similar price order comes in, the arrival time will be used to determine the position of the new order) so the best order lies at the start of the array (Here best order means likelihood of an order getting matched), So for buy orders the best order would be the one with the highest price, so the buy orders should be kept in descending order and vice versa for sell orders i.e. ascending order. Please do note that maintaining the orders in arrays will result in O(n) worst case time complexity, it's up to the teams to use more efficient data structures to improve efficiency.

The matching engine's behavior differs based on the type of orders received. Day Trading Inc requires to implement two types of orders:

- Market Order: In this case the client wants to buy/sell at the best available market price. This type of order should be executed immediately, in the case of buy market order, the matching engine will look for the sell order with the best price i.e. cheapest available sell order to match the incoming buy order to.
- Limit Order: In this case the client wants to buy/sell at a particular price which might not be the current market price. For example, a client places a sell limit order more than the current market price, since it is not the best price the order will not get matched to any market buy order on its arrival. This order will need to be put in its correct place in the sell orders data structure. Any future trades might fluctuate the stock price and then this order will be executed.

The price of a stock at any given time will be equal to the best available sell order (or cheapest sell order) present in the sell orders data structure of that stock since that is the order that will be executed when a buy order comes in. Orders can also be partially fulfilled, when two orders get matched the quantity might not be equal and either the buy/sell order will not be fulfilled completely, the unfulfilled order will not be removed from the order book, but the status of the order should be updated to reflect partial fulfillment. See Appendix A for the expected API contract.

Authentication

It is expected by Day Trading that the system will have token authentication using JWT. This token should be generated upon logging into the system with a username and password and should be returned to the browser in the response payload and **not as a cookie**. When this token is being generated user's identity data should be encoded into this token to verify the user. The internal API calls will have this token passed into the header of each HTTP request. More information on this can be found in appendix A.

Transactional Requirements

It is expected by Day Trading that the system uses a single database transaction while inserting/updating database records on the buyer and seller side when an order is matched, if there is an error in any of one the operations the whole trade can be reversed to maintain consistency. It is up to the teams to implement more solutions to improve the robustness of the system.

User Credentials Storage

Day Trading requires that user's passwords should not be leaked in case of a data breach Hence it is required that the passwords should not be stored as plaintext. The password should be hashed using hashing libraries like bcrypt and then be stored in the database. When matching credentials during login the same library can be used to match the incoming plaintext password with the hash stored in the database to identify the user.

Evaluation

The project will be evaluated in three stages with JMeter scripts by the teaching team. After reading this document it is recommended to read SingleUserTest.pdf which contains the test script outline for the Test Run 1 deliverable.

Each group will be provided a linux based server VM where they will need to maintain a running version of the trading application. This server will be used for collaboration between the team members and evaluation by the teaching team so the application needs to be stable on it by the deliverable deadlines, failure to do so will result in loss of grade for the applicable deliverable.

The weekly logbooks and individual contribution reports will be taken into account while grading to ensure fair grading among the team members. It is of utmost importance that each member document their contributions in the logbook and fill the contribution report honestly, any fraudulence will not be tolerated and will result in failure of the course. The sample documents for both can be found on Bright Space.

All documentation (including the web application and presentation text) must be clearly and concisely written. Marks will be deducted based on the quality and professionalism of the submitted material.

Grade Distribution And Deliverables

Project Task	Weightage	Deadline
Report-1, Individual Contribution Report	5.00%	Jan 31st 11:59 PST
Test Run 1 (Single User Workload)	10.00%	Feb 15th 11:59 PST
Report-2 , Test Run 2 (10,000 Users), Individual Contribution Report	2.00%, 8.00%	Feb 28th 11:59 PST
Final Report, Test Run (150,000 Users), ,Final Individual Contribution Report	5.00%, 10.00%	March 19th 11:59 PST
	40.00%	

Log Book Deliverables

Assignment Name	Time Period	Deadline
Logbook_1	20th – 24th Jan	24th Jan 2025 11:59 PST
Logbook_2	27th – 31st Jan	31st Jan 2025 11:59 PST
Logbook_3	3rd – 7th Feb	7th Feb 2025 11:59 PST
Logbook_4	10th -14th Feb	14th Feb 2025 11:59 PST
Logbook_5	17th – 21st Feb	21st Feb 2025 11:59 PST
Logbook_6	24th – 28th Feb	28th Feb 2025 11:59 PST
Logbook_7	3rd – 7th March	7th March 2025 11:59 PST
Logbook_8	10th – 14th March	14th March 2025 11:59 PST
Logbook_9	17th – 21st March	21st March 2025 11:59 PST
Logbook_10	24th – 28th March	28th March 2025 11:59 PST

Appendix A

Required APIs

All request and response bodies should be in JSON format.

Expected Unsuccessful Response: {“success”:false,”data”:null,”message”:”<error message>”}

- Unauthenticated APIs

Http Method	Endpoint Name	Request Body/ Query Params (If any)	Expected Response (JSON)
POST	<url>/authentication/register	{"user_name":"test","password":"test","name":"Test User"}	{ "success": "true", "data": null }
POST	<url>/authentication/login	{ "user_name": "test", "password": "test" }	{ "success": true, "data": { "token": "your-token" } }

Note

1. In /register endpoint username should be unique for each user. If username already exists for a different user appropriate error message should be returned.
2. The login API should return a JWT token encoded with user identity information (for example, username or userID which could be the primary key in a database)

- Authenticated APIs

The next set of APIs required user authentication through logging in. The JWT token needs to be passed in the header of each HTTP request with key 'token'. On server side these protected routes will first need to verify the token with an appropriate library and decode the user identity data which will then be used for further processing. If a tampered or expired token is passed to these the API should immediately return 401 (Unauthorized) HTTP Code. It is recommended that the teams do this on a middleware level of the API so it can be reused.

Http Method	Endpoint Name	Request Body/ Query Params (If any)	Expected Response (JSON)
GET	<url>/transaction/getStockPrices	-	{"success":true,"data":[{"stock_id":1,"stock_name":"Apple","current_price":100},{"stock_id":1, "stock_name":"Google","current_price":200}]}
GET	<url>/transaction/getWalletBalance	-	["success":true,"data":{"balance":100}}
GET	<url>/transaction/getStockPortfolio	-	{"success":true,"data":[{"stock_id":1,"stock_name":"Apple","quantity_owned":100},{"stock_id":2,"stock_name":"Google","quantity_owned":150}]}
GET	<url>/transaction/getWalletTransactions	-	{"success":true,"data":[{"wallet_tx_id":"628ba23df2210df6c3764823","stock_tx_id":"62738363a50350b1fbb243a6","is_debit":true,"amount":100,"time_stamp":"2025-01-12T15:03:25.019+00:00"}, {"wallet_tx_id":"628ba36cf2210df6c3764824","stock_tx_id":"62738363a50350b1fbb243a6","is_debit":false,"amount":200,"time_stamp":"2025-01-12T14:13:25.019+00:00"}]} //is_debit = true; deduction transaction
GET	<url>/transaction/getStockTransactions	-	{"success":true,"data":[{"stock_tx_id":"62738363a50350b1fbb243a6","stock_id":1,"wallet_tx_id":"628ba23df2210df6c3764823","order_status":"COMPLETE

			D","is_buy":true,"order_type":"LIMIT", "stock_price":50,"quantity":2,"parent_tx_id":null,"time_stamp":"2025-01-12T15:03:25.019+00:00"}, {"stock_tx_id":"62738363a50350b1fbb243a6","stock_id":1,"wallet_tx_id":"628ba36cf2210df6c3764824","order_status":"COMPLETED","is_buy":false,"order_type":"MARKET","parent_tx_id":null, "stock_price":100,"quantity":2,"time_stamp":"2025-01-12T14:13:25.019+00:00"}]}

Note

- The above endpoints doesn't take in any request body so it might be confusing how to identify for which user we are requesting the data for as APIs are stateless, the token which is passed in the header will give the userId or user_name which will be used to query the database for information.
- The sell stock transactions prior to execution and completion should have order_status as 'IN_PROGRESS' first, if partially matched first the status should change to "PARTIALLY_COMPLETE" and then it should be changed to "COMPLETED". See Appendix B flowcharts for buy market and sell limit order transaction flow.

Partial Order Fulfillment And Order Cancellation

In case of partial fulfillment a pair of wallet and stock transactions should be inserted for each individual order, the status of original stock transaction should be changed to 'PARTIALLY_COMPLETE' as soon as the first partial order is completed. The stock transactions has a key parent_tx_id which will have the original stock transaction id of the order. So it is possible to have multiple child transactions if the original stock transaction quantity is large.

In case of limit orders they can be cancelled if the status is not 'COMPLETED'. If an order is cancelled, change the status of stock transaction to 'CANCELLED' and refund either the wallet/ stock depending on buy or sell. Please reference the flow charts in appendix B.

Http Method	Endpoint Name	Request Body/ Query Params (If any)	Expected Response (JSON)
POST	<url>/transaction/addMoneyToWallet	{"amount":100}	{"success":true, "data":null}

POST	<url>/engine/placeStockOrder	{"stock_id":1,"is_buy":true,"order_type":"LIMIT","quantity":100, "price":80}	{"success":true, "data":null}
POST	<url>/engine/cancelStockTransaction	{“stock_tx_id”:” 62738363a50350b1fbb243a6”}	{"success":true, "data":null}

Request body format for placeStockOrder:

- BuyMarket: {“stock_id”:1,”is_buy”:true,”order_type”:”MARKET”,”quantity”:10,”price”:null}
- Sell Limit: {“stock_id”:1,”is_buy”:false,”order_type”:”LIMIT”,”quantity”:10,”price”:80}

- Possible values for order_type: MARKET, LIMIT
- Possible values for is_buy: true,false
- When market order is being placed price should be passed as null.
- If the API is fed any other values for the above keys, it should return an appropriate response in error message.

Http Method	Endpoint Name	Request Body/ Query Params (If any)	Expected Response (JSON)
POST	<url>/setup/createStock	{"stock_name":”Google”}	{"success":true,"data":{“stock_id”:”your_

Note:
The above two APIs are meant for initialization of system state.
/addStockToUser should not be used in any other part of the system other than initialization as it is entirely separate from the normal buy sell order flow and doesnt involve wallet and stock transactions. It is just meant to give an account initial number of stocks and kickstart trading.
A postman collection can be found on Bright Space in Project Documents with all the APIs mentioned in this document.

APPENDIX B

System Initialization Flow

- Create a company account.
- Create a stock using /createStock endpoint and get stock_id of the created stock
- Add n number of stocks to this account with /addStockToUser endpoint.
- Place a sell order inside the matching engine with this account, we are placing a LIMIT order here because we want to set an initial price of the stock. Now since there is a valid order inside the matching engine we can create more users, add money in their wallets and start trading.