



AgriCom Training

Report #3 CSCI 441 - Team B

*David Gladden, Christopher Katz,
David Schiffer, Calvin Ku, Alexis Angel*

Contents

Contents

1. Customer Statement Of Requirements	5
2. Glossary of Terms	8
3. System Requirements	
3.1. Business Goals	9
3.2. Functional Requirements	10
3.3. Non-functional Requirements	11
3.4. User-Interface Requirements	12
4. Functional Requirements Specification	
4.1. Stakeholders	15
4.2. Actors and Goals	15
4.3. Use Cases	17
4.4. System Sequence Diagrams	24
5. User Interface Specification	
5.1. Preliminary Design	26
5.2. User Effort Estimation	30
6. Effort Estimation	
6.1. Background	32
6.2. Unadjusted Actor Weight	33
6.3. Unadjusted Use Case Weight	33
6.4. Technical Complexity Factor	34
6.5. Environmental Complexity Factor	35
6.6. Calculations	36
7. Analysis and Domain Modeling	
7.1. Conceptual Model	37
7.2. System Operation Contracts	42
7.3. Data Model and Persistent Storage	44
7.4. Mathematical Model	46
8. Interaction Diagrams	47
9. Class Diagram and Interface Specification	
9.1. Class Diagram	52
9.2. Data Types and Operation Signatures	52
9.3. Traceability Matrix	57

9.4.	Design Patterns.	58
9.5.	Object Constraint Language	58
10.	System Architecture	
10.1.	Identifying Subsystems.	61
10.2.	Architecture Styles	61
10.3.	Mapping Subsystems to Hardware.	62
10.4.	Network Protocol.	63
10.5.	Global Control Flow.	63
10.6.	Hardware Requirements.	64
11.	Algorithms and Data Structures	
11.1.	Algorithms.	65
11.2.	Data Structures.	65
12.	User Interface Design and Implementation	66
13.	Design of Tests	
13.1.	Test Cases	67
13.2.	Test Coverage	71
13.3.	Integration Testing	72
14.	History of Work, Current Status, and Future Work	
15.	Report 3 Contributions	74
	References	75

Summary of Changes

- ❖ Added duration calculation to effort estimation section.
- ❖ Updated Domain Model diagram to reflect the use of a single Database to store our data.
- ❖ Added more description on Domain Analysis's Tracibility Matrix.
- ❖ Updated User Interface and Design Implementation sections to reflect the current status of the website.
- ❖ Modified DB schema diagram to better show relational structure.
- ❖ Updated description of subsystem breakdown in System Architecture and System Design.
- ❖ Updated Actor goals where necessary.
- ❖ Updated Fully Dressed Use Cases where necessary.
- ❖ Added sections 9.4 and 9.5 to complete full report
- ❖ Class Diagram updated
- ❖ Interaction diagrams and descriptions updated
- ❖ Data Types and Operation Signatures Updated

1 Customer Problem Statement

Introduction

A conundrum with employers is with challenges to the workforce are appropriate credentialing and services on stocks and how companies can give the edge in our competitive market today to extradite a complex series of decisions to streamline our users to make the best decision in the shortest amount of time. A customer requires a centric focus on expounding information from the theory put into practice. The capacity requires taking on calculated risk in a controlled environment to trade in goods worldwide. Fortunately, our solution in mind, a web-based system for training those in various positions, can assist employees and managers alike to incorporate strategies and theories in a structured format to use trades and sales for common metals across broad regions across the globe. Our mission statement is to help customers find the strategies they intend to work for them with our website in a place that can be accessed anywhere at any time with real-time metrics that can correlate to real-world stock reports.

Project Summary

Proper induction into the workforce is the forefront of many company concerns in reducing turnover and providing the necessary means to retain key talent across the marketplace. While on-the-job training is critical for assignment, it predicts that risk is involved when exposing new talent to the workforce that can create an unidentified source of cost to help retain and protect those that are in the interest of procuring a service or product to the company. A company is required to provide a necessary source of interest in the employee to help retain staff but also is required to maintain a safety net of sorts to help prelude new employees to the risks of an ever present and changing marketplace that is high intensity with minimal setbacks.

Our mission is to empower our customers to assist with the on-the-job training process with the scope of our model and to assist through graphic imagery and design processes to target our audience through the website we have created. Our idea is to bring into action to implement learning and bring learning-support aids to assist customers and managers of sales through agent networks that bring theory and practice alike together to incorporate in our website design as a medium to present a design philosophy to life that those in finance may implement to better understand the environment that trades with common metals to better incorporate a schema to include when required in such a demanding environment.

A system that may identify detailed, accurate, and up-to-date product information to inform potential buyers effectively and efficiently for purchases for sellers and to allow a streamlined means to reduce processing in supervisory or managerial positions to define and address needed issues from available sources for an on-the-go interaction for faster resolution times. Our intention is to provide the necessary dynamic, real-time. Available anywhere with internet access, customers may be allowed access to the website with appropriate credentials

for those with access. Our website can include training for traders and managers alike who would require assisting with employees in a manner that can expedite the process for future processes when assisting customers and employees alike.

The intention is one to include stocks of ongoing trades in a controlled environment to help buffer the potential impacts that decision makers can perform while operating freely as though they were to perform actual trades. Along with providing a model to introduce the usage of common metals, we also incorporate inventory and account billing to mirror stock trading to its finest with an ongoing, updated database system to utilize for one's leisure. The usage of common metals extends beyond the traditional format of a 2-dimension format and includes regional changes in currency to better understand and incorporate a dynamic viewpoint into regional differences to compare prices and incorporate the necessary depth to better allow traders the opportunity to understand real-world implications when working with such a broad range of materials.

Overview

The main overview of the survey of documentation is to assist with product ordering by customers, agents, and management alike. Our website is more of a training ground for those who are looking for a chance to get involved in the stock market with valued goods and to train into a medium that can assist on what to look for and review when it comes to split second decision making. The ability to determine and value where a decision is to make maximum impact is what our website is meant to create and build from when reviewing and determining what would be an ideal selling point when trading. The main selling points are those from the traders and the managers when looking into the parameters and what our product is meant to entail when reaching the market.

The primary duties of the website include direct access to up-to-date information and decision-making ability to buy or sell goods with customized offerings to allow differing access to information dependent on the user while order information and product information to managers as learning aids. As mentioned previously, working in a wide regional area marked by large trading centers across the globe (UK, EU, USA, etc.) will show differences in pricing and market shares to dictate worldwide costs and to include differences in pricing via regional currencies, transfer rates, and impact to overall costs when transferred to one region to another. Differing currencies are to be in place to ensure that an accurate account of balances is to dictate and infer upon upcoming trades.

Proposition

The proposition is one that can help create the necessary groundwork to allow employees to operate in an artificial environment prior to being released into the workforce. One where we can ensure that demands are being met with minimal risk in allowing employees the chance to grow into the role to reduce the cost of on-the-job incidents and to create stronger employment with higher retention rates across the board. Our interest is to help ensure that traders and managers alike are able to produce the highest possible result in a competitive

environment while still reducing the impact of a chaotic environment due to a lack of experience brought about by a high barrier of entry that can serve as a deterrent to prospective employees that can predict poor outcomes to companies that can contribute to an overall lack of quality that can be noticeable over a period of time. To provide a secure means to allow employees and managers to remain engaged prior to being released from training is an integral part of the onboarding process to allow departments the rationale to pontificate a means to reduce and streamline the workload into an applicable part of the hiring process.

Solution

For the trader, eliminating the delay between buying or selling with ongoing prices for metals in the market may reduce errors by current market trends and eliminate redundancies with trades while incorporating a medium to include that can foster an environment to dictate decisive action and to include into necessary resources, tasks, and tutelage to better evolve into a more effective means to allow for more yields when trading metals and other common goods. Our product can also allow for necessary distribution of identifying materials to be correctly exchanged based on specific regions to be able to subtract out fees, materials with similar or better purchasing power, and to extract interest when rates make changes throughout the period that the trader is in training. With online resources that are available,

For managers, our website can allow a necessary opportunity to allow managers to reduce decision making by ensuring that interaction with traders is done in an expedite process to include necessary reaction time for input to be included to help retain staff during the training process and to allow managers to interact with traders to ensure that processes remain in place. Quick time intervention to trader concerns and resources to include into the decision making process is a priority to better implement a standardized practice to maintain focus on stock trading, a key priority during the lifecycle of onboarding for new employees.

Conclusion

In summary, a training program to allow traders and managers the opportunity to operate with relative freedom to their positions with material present to educate and reform bad or misinformed practices prior to be released is a necessary inclusion to any onboarding process to the employee that can dictate a higher level output and retain talent to include into the HR hiring process that any respectable organization is to require to maintain a standard that can reduce costs over a long period of time. The inspection of a higher quality assurance is necessary to allow retention to be maintained and continued for future training programs. It should be noted that training is a priority for staff and to include a methodology that can reduce the risk associated with newer employees is a key focus to remain in high-standing and productive positions in trades.

2 Glossary of Terms

- ❖ **Account total** - Also known as Net Worth or Total Equity. It is the user account's value in cash plus its total holdings value.
- ❖ **Actors** - Any external entity that interacts with the system.
- ❖ **Agricultural Commodities** - Plant, animal products, and their by-products, such as crops and meat products.
- ❖ **Cash total** - The user account's total available cash. Cash can be spent by submitting an order ticket to buy commodities. Cash can be gained by submitting an order ticket to sell owned commodities.
- ❖ **Commodity** - Raw material or agricultural product that can be bought and sold.
- ❖ **Holdings** - The user account's owned commodities.
- ❖ **Manager** - An user with administrative privileges. This user can manage over multiple Trainee users and have access to their profiles and transactions.
- ❖ **Order Ticket** - A form filled by the user to submit a transaction.
- ❖ **Order Type** - A transaction can either be a "Sell Order" or a "Buy Order" type.
 - Sell Orders exchanges a specified amount of the account's commodity holdings to its cash value.
 - Buy Orders exchanges a specified amount of the account's cash value to its equivalent value in commodity.
- ❖ **Portfolio** - A detailed grouping of financial information and assets consisting of available cash balance and current commodity holdings at their current and purchased value.
- ❖ **Revenue Stream** - the different sources of money for a company generated from the sale of goods and services
- ❖ **Trader** - A person who engages in the buying and selling of commodities.
- ❖ **Trainee** - An user with standard privileges. This is the user who will act as a Trader.
- ❖ **Transaction** - An exchange between a commodity and cash currency.

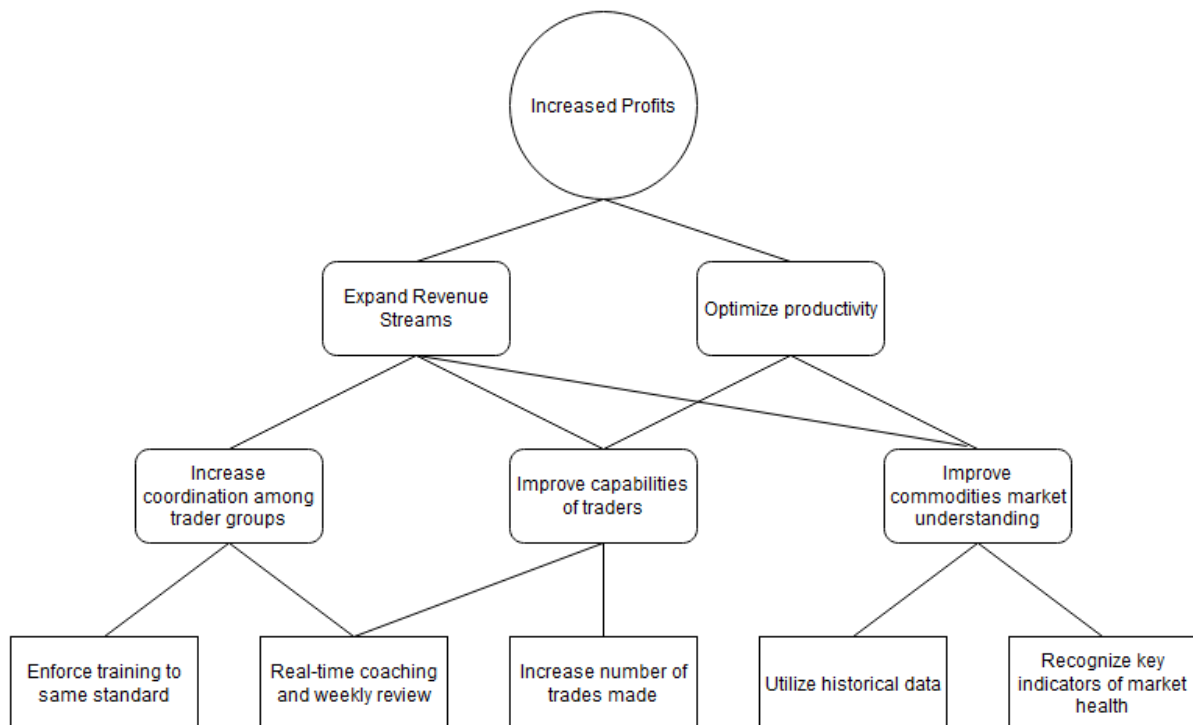
3 System Requirements

3.1 Business Goals

Increase profitability by improving the ability to recognize key indicators of market health leading to less mistakes and taking advantage of opportunities as they present themselves.

Increase productivity of the trader workforce that is more knowledgeable and proficient in trading on the commodities market.

Expand revenue streams by increasing the capabilities of the traders who in the past would be locked down to a small portfolio, but now will be able to manage a wide spread of commodities and improve cash flow by orders of scale



3.2 Functional Requirements

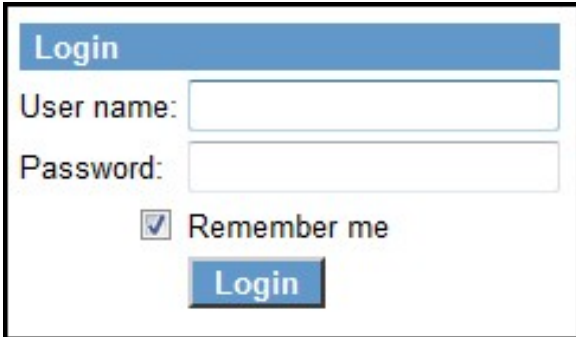
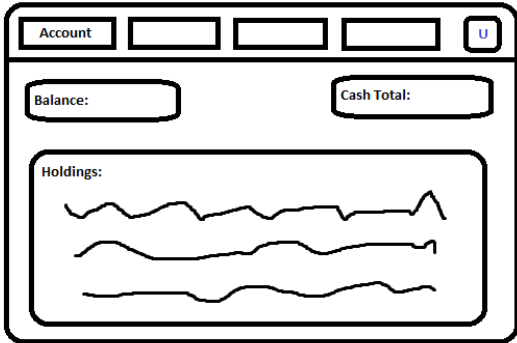
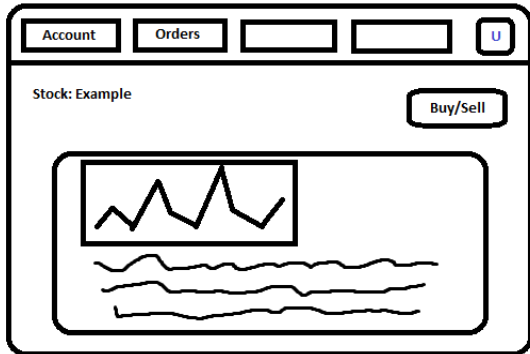
ID	PW	REQUIREMENT
REQ-1	5	The system shall allow new users to register an account by providing a real-world email, which shall be external to our system. Required information shall include an unique login ID and a password that conforms to guidelines, as well as the user's first and last name. The user shall be able to specify if the new account will be a Manager account or a Trainee account at registration. Upon successful registration, the system shall set up an account with a \$10,000 starting balance.
REQ-2	5	The system shall support order placement by filling out a form known as "order ticket". The order ticket shall include the account's information, order type (buy/sell), quantity, and commodity. The order ticket shall be placed in an order queue to be processed.
REQ-3	5	The system shall constantly review the queued orders and for each order ticket in the queue check the following: <ul style="list-style-type: none">• If the order type is a buy order, cancel the order ticket if there is not sufficient cash in the account.• If the order type is a sell order, cancel the order ticket if there is not sufficient commodity holding in the account.• Else, execute the trade instantly at the current price.
REQ-4	5	The system shall maintain a database of user accounts, portfolios, commodities price history, transaction history, and educational information.
REQ-5	5	The system shall periodically update and process market data to provide close-to-real-time information for each commodity, including: <ul style="list-style-type: none">• Prices and graph• Fundamental Indicators such as 52 week highs and lows• Latest news• New education information
REQ-6	5	The system shall maintain each Trainee account with a portfolio of commodities, account total, cash total, holdings, and transaction history information. The account total shall be updated with the latest value upon access.
REQ-7	4	The system shall allow Manager accounts to find Trainee accounts by their account login ID and add/remove them to their profile for supervision. The system shall allow Managers to view their supervising Trainee's profile and transaction history.
REQ-8	4	The system shall allow Trainee accounts a month's time frame for their training in which order tickets can be placed within that frame. The time frame can be reset by the account holder at the end or by a Manager.


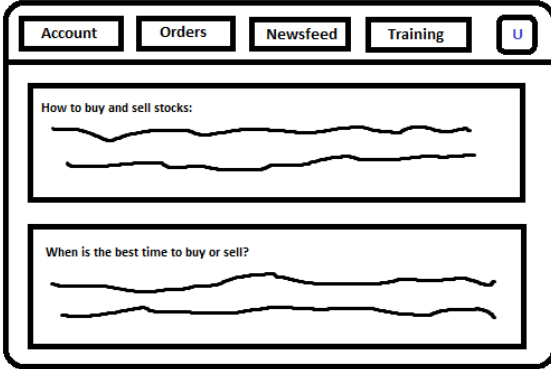
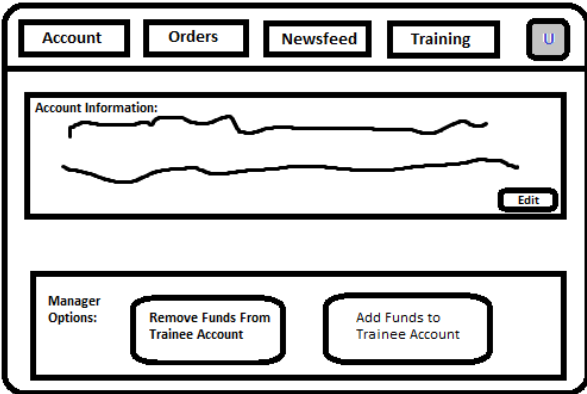
REQ-9	3	The system shall allow Manager accounts to reset the balance and training timeframe of their Trainee's account.
REQ-10	2	The system shall allow Trainee accounts and their Manager to comment on each Trainee account's transaction.
REQ-11	1	The system should have an area that allows Trainee accounts to access educational information about trading commodities.

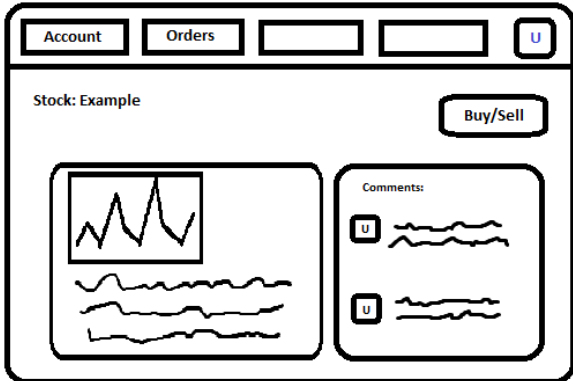
3.3 Nonfunctional Requirements

REQ-12	5	The system shall be simple to use and have a minimal learning curve. Data shall be presented in such a way that the user's focus is automatically drawn to them when the user views each page. The user shall have less than 5 clicks to navigate to any page.
REQ-13	5	All user data shall be stored in the system's database. No user information shall be stored on the user's device. No user shall have access to directly modify any data.
REQ-14	3	Market data shall be updated as real-time as possible. Data latency shall not be more than 30 minutes at any given time.
REQ-15	3	The system shall be platform independent and should appear and run the same across Windows, Mac, and Linux systems.
REQ-16	2	The system shall require minimal maintenance. The system shall require maintenance at most once a week.

3.4 User Interface Requirements

ID	PW	REQUIREMENT
REQ-17	5	<p>GUI must have a log-in page.</p> 
REQ-18	5	<p>GUI must have a page that shows current account information such as current balance, cash total, and holdings.</p> 
REQ-19 a,b	5	<p>GUI must have a page to submit orders. GUI must show the history of stock prices and consistently update.</p> 

REQ-20	5	<p>GUI will have a newsfeed showing notable events on account</p> 
REQ-21	4	<p>GUI should have a training page that holds valuable learning information for trainees.</p> 
REQ-22	5	<p>GUI must have a page for account management. Managers will be able to control funds on trainee accounts.</p> 

REQ-23	3	<p>GUI will allow comments to be made on orders</p> 
--------	---	--

4 Functional Requirements Specifications

4.1 Stakeholders

There are two main groups of stakeholders for this proposal including internal and external stakeholders. Internal stakeholders are defined as those within the organization that will approve requirements, use the platform, give feedback, and measure performance. External stakeholders are defined as those outside the organization that have an effect on this platform that is outside the organization's control.

1. Internal Stakeholders

- a. Owners: an organization that has paid for the right to use this software through our licensing agreement.
- b. Managers: a person with administrator privileges necessary to maintain the training environment and the performance of trainees.
- c. Trainees: any employee of the organization that will be using this training software to improve their understanding of the commodities market

2. External Stakeholders

- a. Government: The government of the United States involved in setting policy and laws related to the legal transactions on the commodities market
- b. Competitors: Any company offering similar products that involve teaching the basic concepts of the commodities market and allowing the user to practice in a safe environment
- c. Commodities Market: The public market available for the sale and purchase of commodities between multiple entities.

4.2 Actors and Goals

- ❖ **Guest:** User who has not yet registered or been authenticated by the system.
 - **Type:** Initiating
 - **Goals:**
 - Register an account.

- Login with valid account information.
- ❖ **Manager:** User who has been registered as a manager account.
 - **Type:** Initiating
 - **Goals:**
 - Search for trainee accounts.
 - Add trainee accounts to their supervision profile.
 - View trainee accounts profiles and transaction history.
 - Reset or add funds to trainee accounts portfolios.
 - Comment on trainee accounts transactions.
- ❖ **Trainee:** User who has been registered as a trainee account.
 - **Type:** Initiating
 - **Goals:**
 - View profile of commodities, account totals, cash total, and holdings.
 - View account information.
 - View educational information related to commodity trading.
 - Submit purchase and sell orders for their accounts.
- ❖ **Browser:** Allows interaction between user and system.
 - **Type:** Participating
 - **Goals:**
 - Presents data to the user.
 - Accepts data from the user.
- ❖ **Commodity API:** Provides up-to-date commodity prices for the system.
 - **Type:** Participating
 - **Goals:**
 - Provide up-to-date commodity price information when fetched.
 - Provide latest educational information when fetched.
 - Provide latest commodity news when fetched.
 - Updates system database accordingly for historical information.
- ❖ **Database:** Holds all information of current user accounts and their portfolios.
 - **Type:** Participating
 - **Goals:**
 - Save account information for new users.

- Save and maintain portfolio balance and holdings.
- Save and maintain transactional history.
- Maintain commodities price history.
- Save and maintain manager comments.

❖ **TimerTask:**

- **Type:** Initiating/Participating
- **Goals:**
 - Fetch information from commodity API at regular set intervals.
 - Store fetched information into the database.
 - Set a Trainee account's training time frame of 1 month. Will trigger a performance report at the end of the regiment.

4.3 Use Cases

Casual Description

❖ **UC-1: Buy Commodity**

- **Actor:** Trainee (*Initiating*), Commodity API (*Participating*), Database (*Participating*)
- **Goal:** To buy a commodity. This involves the Trainee filling out and submitting a buy order ticket of a specified commodity and its market price queried from Commodity API. The order ticket will be processed and its data saved in the Database. Changes to the Trainee's portfolio will be reflected.

Related Requirements: REQ-2, REQ-3, REQ-4, REQ-5, REQ-6

❖ **UC-2: Sell Commodity**

- **Actor:** Trainee (*Initiating*), Commodity API (*Participating*), Database (*Participating*)
- **Goal:** To sell a commodity. This involves the Trainee filling out and submitting a sell order ticket of a specified commodity at the current market price queried from Commodity API. The order ticket will be processed and its data saved in the Database. Changes to the Trainee's portfolio will be reflected.

Related Requirements: REQ-2, REQ-3, REQ-4, REQ-5, REQ-6

❖ **UC-3: Register as Trainee**

- **Actor:** Guest (*Initiating*), Database (*Participating*)
- **Goal:** To register an account as a Trainee. This involves the Guest to fill out the Trainee registration form with their user information. The Database will check to make sure that the user name submitted is unique and create an account for the Guest.

Related Requirements: REQ-1, REQ-4, REQ-6

❖ **UC-4: View Portfolio**

- **Actor:** Trainee/Manager (*Initiating*), Commodity API (*Participating*), Database (*Participating*)
- **Goal:** To view the current status of an account's portfolio. This involves the User selecting an account to display the current portfolio information retrieved from the Database. The Database will update the holdings and account total with the latest values retrieved from the Commodity API.

Related Requirements: REQ-4, REQ-5, REQ-6

❖ **UC-5: View Transaction History**

- **Actor:** Trainee/Manager (*Initiating*), Database (*Participating*)
- **Goal:** To view a list of transactions made. This involves the User selecting the transaction history section of a profile to view the logged transactions made by order type, time, and price.

Related Requirements: REQ-4, REQ-6

❖ **UC-6: Submit Comment**

- **Actor:** Trainee/Manager (*Initiating*), Database (*Participating*)
- **Goal:** To submit a comment on a Transaction. This involves the User submitting a comment on through the transaction history to provide feedback for the Manager/Trainee.

Related Requirements: REQ-4, REQ-6, REQ-10

❖ **UC-7: View Comment**

- **Actor:** Trainee/Manager (*Initiating*), Database (*Participating*)
- **Goal:** To view a comment made on a Transaction. This involves the User selecting a transaction from the transaction history to view the saved comments.

Related Requirements: REQ-4, REQ-6, REQ-10

❖ **UC-8: View Educational Information**

- **Actor:** Trainee/Manager (*Initiating*), Commodity API (*Participating*), Database (*Participating*)
- **Goal:** To view education information. This involves the User selecting to view the educational information section saved in the Database. New educational information will be periodically retrieved from the Commodity API and saved in the Database.

Related Requirements: REQ-4, REQ-5, REQ-11

❖ **UC-9: View Commodity**

- **Actor:** Trainee/Manager (*Initiating*), Commodity API (*Participating*), Database (*Participating*)
- **Goal:** To search ticker symbols and view market information for specified commodities. Information will include prices, charts, fundamentals, news articles, etc. Information will be queried from Commodity API.

Related Requirements: REQ-4, REQ-5

❖ **UC-10: Register as Manager**

- **Actor:** Guest (*Initiating*), Database (*Participating*)
- **Goal:** To register an account as a Trainee. This involves the Guest to fill out the Manager registration form with their user information. The Database will check to make sure that the user name submitted is unique and create an account for the Guest.

Related Requirements: REQ-1, REQ-4

❖ **UC-11: Add Trainee to manage**

- **Actor:** Manager (*Initiating*), Database (*Participating*)
- **Goal:** To add a Trainee account for management by a Manager. This involves the Manager to search Trainee's by user name and select to add them to their management profile so the User will gain access to controlling and viewing the Trainee's account.

Related Requirements: REQ-4, REQ-6, REQ-7

❖ **UC-12: Remove Trainee from management**

- **Actor:** Manager (*Initiating*), Database (*Participating*)

- **Goal:** To remove a Trainee account for management by a Manager. This involves the Manager to search Trainee's by user name and select to remove them from their management profile.

Related Requirements: REQ-4, REQ-6, REQ-7

❖ **UC-13: Reset Trainee's account**

- **Actor:** Manager (*Initiating*), Database (*Participating*)
- **Goal:** To reset the training timeframe or account balance of a managed Trainee. This involves the Manager to find the Trainee's profile and selecting to either reset the Trainee's training timeframe or account balance if their Trainee commits a significant mistake.

Related Requirements: REQ-4, REQ-7, REQ-9

❖ **UC-14: Provide end of training feedback (Considered for future work)**

- **Actor:** Manager (*Initiating*), TimerTask (*Participating*), Database (*Participating*)
- **Goal:** To comment on the performance report of a managed Trainee at the end of their training regiment. This involves the Manager to access and review a Trainee's performance report as well as their transaction history to provide feedback for the Trainee.

Related Requirements: REQ-4, REQ-6, REQ-7, REQ-8

Use Case Diagram

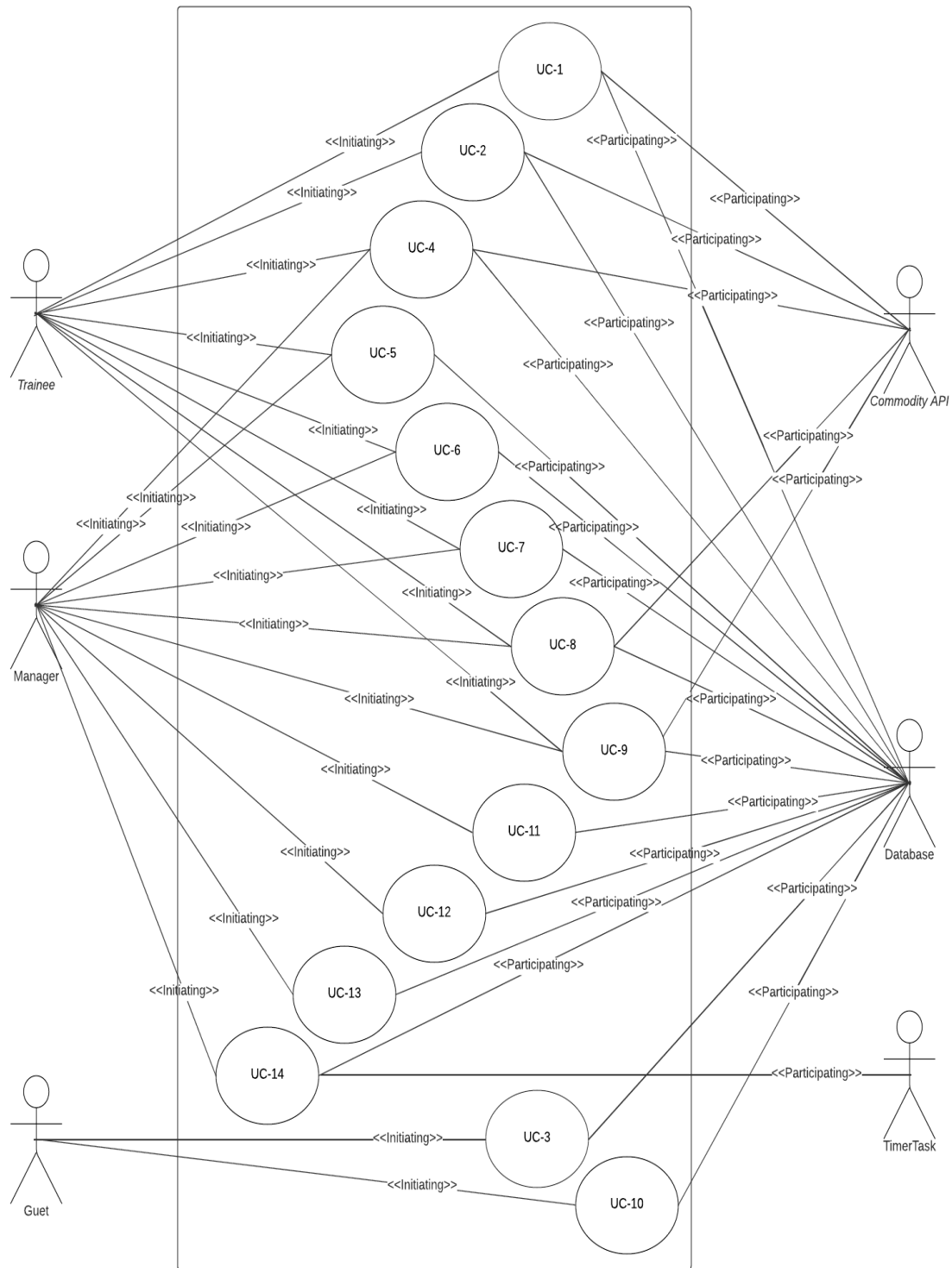


Figure 4.1: Use Case Diagram

Traceability Matrix

REQ	PW	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC13	UC14
REQ-1	5			X							X				
REQ-2	5	X	X												
REQ-3	5	X	X												
REQ-4	5	X	X	X	X	X	X	X	X	X	X	X	X	X	X
REQ-5	5	X	X		X				X	X					
REQ-6	5	X	X	X	X	X	X	X				X	X		X
REQ-7	4											X	X	X	X
REQ-8	4														X
REQ-9	3													X	
REQ-10	2						X	X							
REQ-11	1								X						
Max PW		5	5	5	5	5	5	5	5	5	5	5	5	5	5
Total PW		25	25	15	15	10	12	12	11	10	10	14	14	12	18

Fully Dressed Descriptions

Use Case UC-1	Buy Commodity
Related Requirements:	REQ-2, REQ-3, REQ-4, REQ-5, REQ-6
Initiating Actor:	Trainee
Actor's Goal:	To buy a commodity
Participating Actors:	Commodity API, Database
Preconditions:	Trainee is logged in and shown an option to "Trade".
Postconditions:	System has notified trainee of purchase order outcome. History and portfolio updated.
Flow of Events for Main Success Scenario	
→ Trainee clicks on "Trade" link.	
← System prompts for order type, commodity name/symbol and amount.	
→ Trainee enters/selects order type, valid commodity name/symbol and amount.	
← System queries market price from Commodity API and returns total cost.	
→ Trainee confirms order.	
← System saves the order, the purchased amount, and the updated total funds to the database.	
← System returns notification to trainee that the transaction has been completed.	
Flow of Events for Alternate Scenario 1	
→ Trainee enters/selects invalid commodity name/symbol.	
← System returns notification to trainee that the name/symbol is invalid.	
Flow of Events for Alternate Scenario 2	
→ Trainee attempts to purchase using more funds than are available.	
← System returns notification to Trainee that there is not enough cash to complete purchase.	

Use Case UC-2	Sell Commodity
Related Requirements:	REQ-2, REQ-3, REQ-4, REQ-5, REQ-6
Initiating Actor:	Trainee
Actor's Goal:	To sell a commodity
Participating Actors:	Commodity API, Database
Preconditions:	Trainee is logged in and shown an option to "Trade".
Postconditions:	System has notified trainee of sell order outcome. History and portfolio updated.
Flow of Events for Main Success Scenario	
→ Trainee clicks on "Trade" link. ← System prompts for order type, commodity name/symbol and amount. → Trainee enters/selects order type, valid commodity name/symbol and amount. ← System queries market price from Commodity API and returns total return. → Trainee confirms order. ← System saves the order, the sell amount, and the updated total funds to the database. ← System returns notification to trainee that the transaction has been completed.	
Flow of Events for Alternate Scenario 1	
→ Trainee enters/selects invalid commodity name/symbol. ← System returns notification to trainee that the name/symbol is invalid.	
Flow of Events for Alternate Scenario 2	
→ Trainee attempts to sell more shares than are available. ← System returns notification to Trainee to enter an available amount of share to sell.	

Use Case UC-3	Register as Trainee
Related Requirements:	REQ-1, REQ-4, REQ-6
Initiating Actor:	Guest
Actor's Goal:	To register an account as a Trainee
Participating Actors:	Database
Preconditions:	An unregistered Trainee (Guest) visits the system.
Postconditions:	The guest has been successfully registered as a Trainee, or an error message has been provided.
Flow of Events for Main Success Scenario	
→ Guest clicks on "Sign Up" link. ← System prompts Guest for their user information via Trainee registration form. → Guest enters user information in Trainee registration form. ← System queries database to verify user name is unique. ← System saves new Trainee user into database. ← System notifies Trainee (Guest) that account creation is successful.	
Flow of Events for Alternate Scenario 1	
→ Trainee enters an existing user name. ← System returns notification to Guest to enter a different user name.	

Use Case UC-4	View Portfolio
Related Requirements:	REQ-4, REQ-5, REQ-6
Initiating Actor:	Trainee/Manager
Actor's Goal:	To view current portfolio
Participating Actors:	Commodity API, Database
Preconditions:	Trainee/Manager is logged in.
Postconditions:	Account portfolio is displayed for Trainee/Manager
<p align="center">Flow of Events for Main Success Scenario</p> <p>→ (If manager) Manager selects which managed Trainee account to display.</p> <p>→ Trainee/Manager clicks on the "Portfolio" link.</p> <p>← System queries market prices from Commodity API and updates database.</p> <p>← System pulls selected account portfolio information from database.</p> <p>← System displays current account portfolio information to Trainee/Manager.</p>	

Use Case UC-14	Provide end of training Feedback
Related Requirements:	REQ-4, REQ-6, REQ-7, REQ-8
Initiating Actor:	Manager
Actor's Goal:	To comment on Trainee performance
Participating Actors:	TimerTask, Database
Preconditions:	Manager is logged in, has Trainees to manage, and performance report is ready for review.
Postconditions:	Comments are saved and available for Trainee to review
<p align="center">Flow of Events for Main Success Scenario</p> <p>→ Manager selects which managed Trainee account to display.</p> <p>→ Manager clicks on "View Performance Report" link.</p> <p>← System pulls selected account portfolio information and transaction history from database.</p> <p>← System displays performance report information to Manager.</p> <p>→ Manager clicks on "Provide feedback" link.</p> <p>← System prompts Manager for Trainee performance feedback.</p> <p>→ Manager enters feedback for Trainee and clicks "Submit".</p> <p>← System saves feedback to database.</p> <p>← System returns notification to Manager that feedback has been successfully saved.</p>	

(UC-14: Considered for future work)

4.4 System Sequence Diagrams

In the sequence diagram listed below, we are to describe the interactions and how they respond in tow to our design philosophy. The interactions listed are dynamic and meant to infer an interaction to expedite time-sensitive sales for maximum impact. As such, communication from server to trader is to deliver the information to the user to maximize the decision making process.

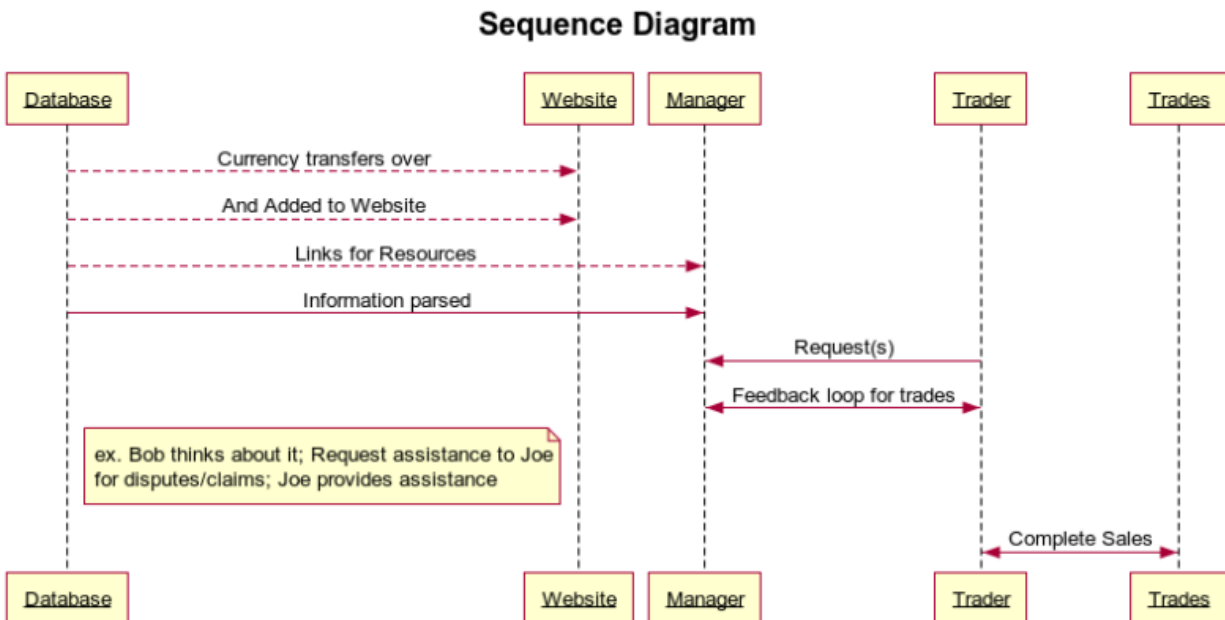


Figure 4.2: Process to determine accuracy in sales begins in the database and trickles down to where the Manager and Trader are able to pull from resources to ensure that the onboarding process to determine accuracy remains in high integrity during the training process. Our indication of success goes back to the feedback loop between Manager and Trader to determine sales and an upward mobility in being able to accurately indicate and determine efficacy of trades.

5 User Interface Specification

5.1 Preliminary Design

The user interface (UI) for AgriCom Training will be a command center for users and managers to interact with their portfolio, manage their or trainees balance, conduct orders on products, and become a center of learning on how stock trading works with commodities. The Portfolio page, changed from Account status originally, (Figure 5.1) will act as the home view for users to show what status their account is at by being able to view their balance, cash total and holdings all at once.

The UI should be lightweight so as to easily run on restrictive platforms such as mobile or tablet browsers. The color scheme should also be easy on the viewer and hold a basic pallet of colors that are web-supported.

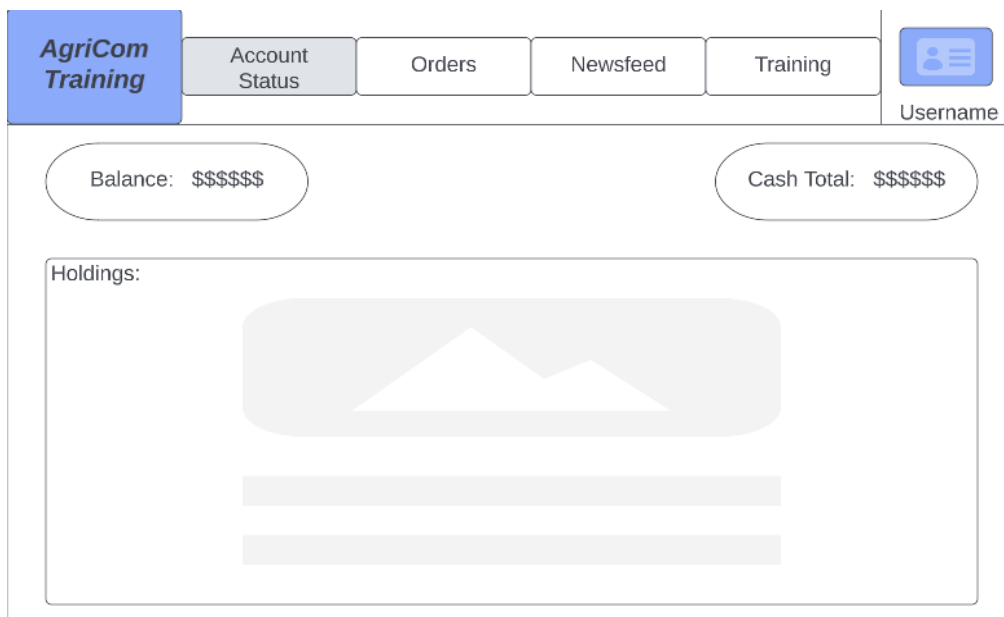


Figure 5.1 Preliminary design for Account Status page.

Landing Page and Login

While logging into AgriCom Training, a user has the ability to choose whether they are making a manager or trainee account. This initial page (Figure 5.2) also does not have any of the header navigation as the features of the website do require user authentication in order to proceed.



Figure 5.2: Preliminary design of AgriCom login page.

Orders

The Orders page (see Figure 5.3) is the most fluid in the preliminary stage as it has the most information on it and is the most subject to change visually as we code the system into working. It is our goal to have everything listed on the figure however the actual page may not look like this in the finished product. Stock details will be listed and the ability to buy/sell a commodity is found on this page.

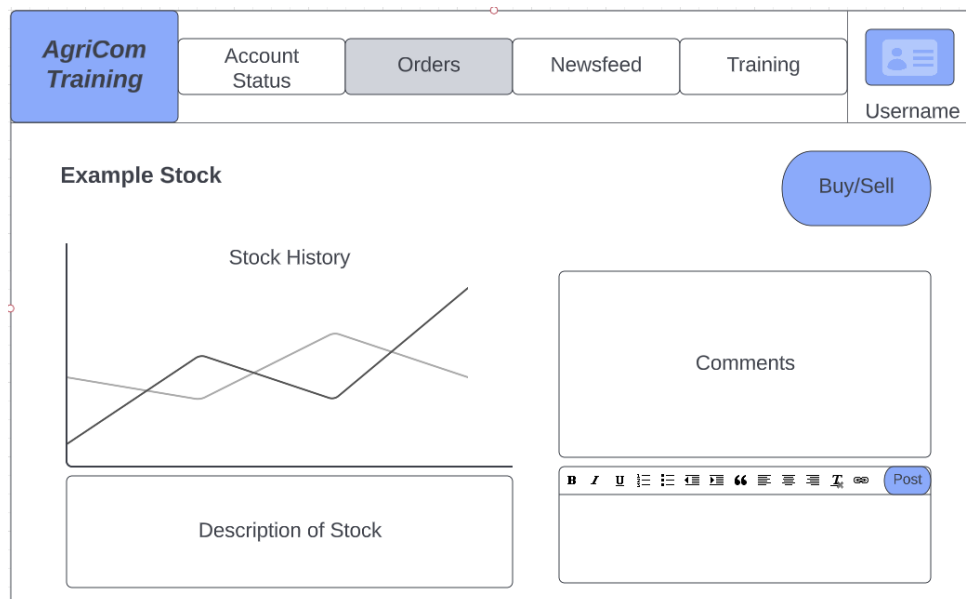


Figure 5.3: Preliminary design for Orders page, subject to change during coding.

Newsfeed

The Newsfeed page (Figure 5.4) gives information about what is happening on the account and gives a history of the events that have occurred.

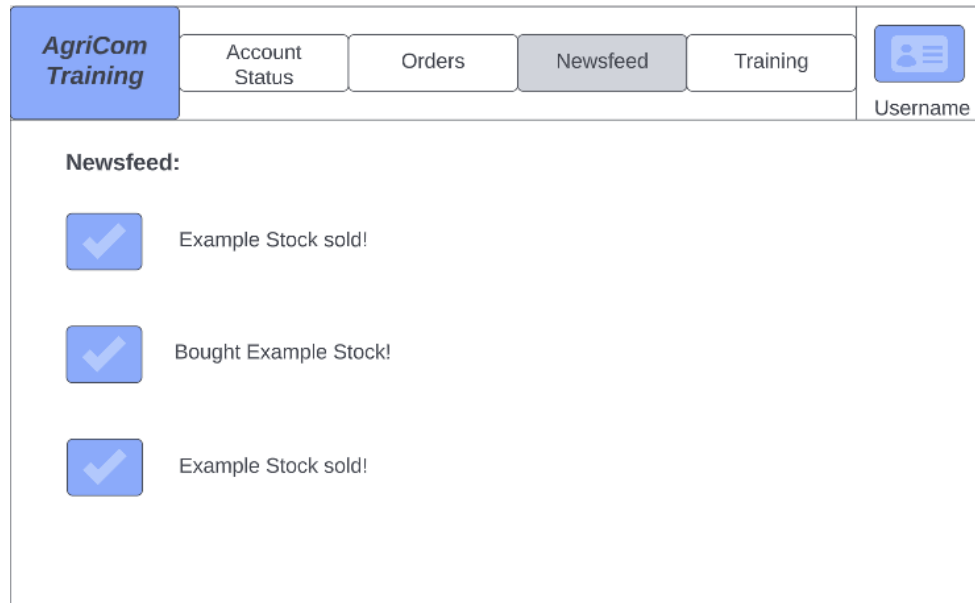


Figure 5.4: Preliminary design of Newfeed showing example events.

Training

Training (Figure 5.5) is a learning asset for all users to be able to gain more information about how to buy and sell commodities. This page is supplemental and useful to those who are just learning how to use the stock market. As mentioned above it may be implemented in the future that you may not need to login to access this page as it is purely an informational section beneficial to all who would access the site.

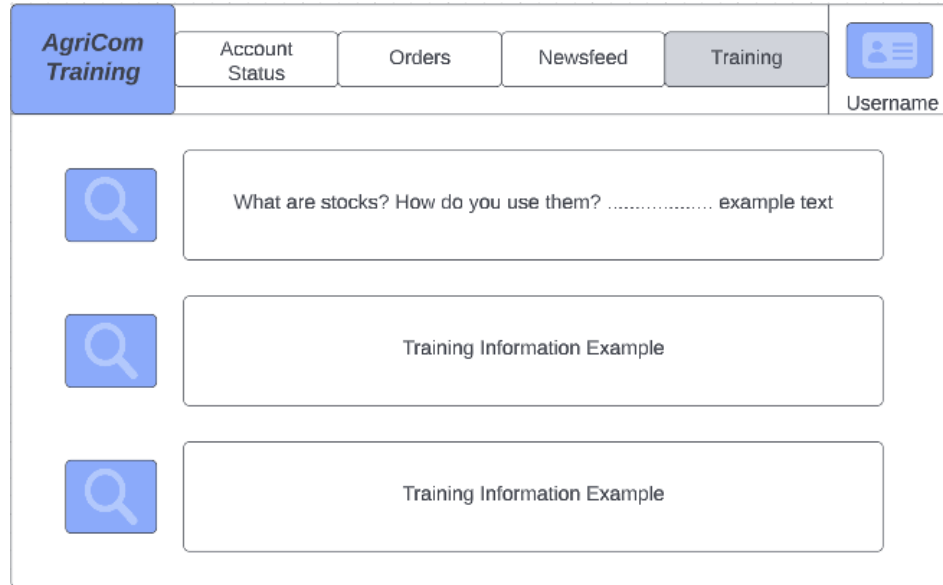


Figure 5.5: Preliminary design of Training page that will show useful information on how to buy and sell trading commodities.

User Settings

User Settings (Figure 5.6) lets the user have the ability to edit information on their account and change their password if needed. This is also the page where managers are able to add/remove funds from trainee accounts they are managing. The top left will show an actual username in the functional website.

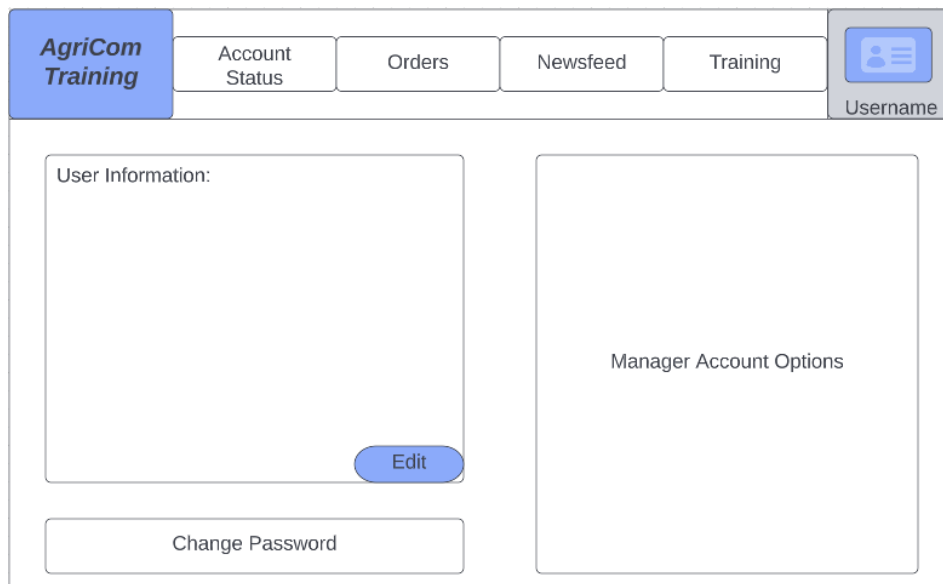


Figure 5.6: Preliminary design of User Settings where a user can change their basic information or a manager can set trainee account funds.

5.2 User Effort Estimation

Some of the most common use case scenarios for AgriCom Training:

Usage Scenario	Clicks	Keystrokes
Login to Existing Account	1 - 3	2 - 50
Register New Account	4 - 5	2 - 50
Buy/Sell Commodity	6	2 - 76
Create Comment on Stock	4	2 - 550

Login to Existing Account

Assume the user has already navigated to the domain and wishes to login if already a registered user:

❖ Data Entry

- Click in the username textbox to enter registered username, 1 click.
- Enter registered username, 1-25 keystrokes.
- Click or tab into the password textbox to enter password, 0-1 click.
- Enter registered password, 1-25 keystrokes.
- Hit enter or click login, 0-1 click.

Register New Account

Assume the user has already navigated to the domain and wishes to create a new user account:

❖ Navigation

- Click on New User button

❖ Data Entry

- Click in the username textbox to enter a new username, 1 click.
- Enter a username, 1-25 keystrokes.
- Click in the password textbox or hit tab to enter a new password, 1 click or keystroke.
- Enter a password, 1-25 keystrokes.
- Select from dropbox whether trainee or manager account, 2 clicks.
- Click Register, 1 click.

Buy/Sell Commodity

Assume the user has navigated to the domain and has logged into an already existing account:

❖ Navigation

- Click on Orders header, 1 click

❖ Data Entry

- Click on the search bar to enter commodity/stock name, 1 click.
- Enter commodity/stock name, 1-50 keystrokes.
- Hit enter or click on the search button, 1 click or keystroke.
- Click on Buy/Sell button, 1 click.
- Click on the amount textbox, 1 click.
- Enter the amount of money the user wants to buy/sell, 1-25 keystrokes.
- Click on the order button, 1 click.

Create Comment on Stock

Assume the user has navigated to the domain at the Order header and is logged into an already existing account:

❖ Data Entry

- Click on the search bar to enter commodity/stock name, 1 click.
- Enter commodity/stock name, 1-50 keystrokes.
- Hit enter or click on the search button, 1 click or keystroke.
- Click on the comment textbox, 1 click.
- Enter comment, 1-500 keystrokes.
- Click post, 1 click.

6 Effort Estimation

We have employed the “Use Case Points” system of estimating the effort necessary to create the system. This is done due to the need to have a metric on the complexity of the design of the project in order to properly delegate resource allocation, with the acceptance that the created metric is necessarily subjective and arbitrary.

6.1 Background

The estimation of “Use Case Points” or UCP is calculated based on the team’s perception of the project’s complexity and the team’s efficiency. Each variable in the computation is defined and computed separately using weighted values, subjective values, and constraining constants. The equation for computing UCP is as follows:

$$UCP = UUCP \times TCF \times ECF$$

In this equation, Unadjusted Use Case points (UUCP) are calculated as a sum of two components:

- ❖ Unadjusted Actor Weight (UAW) which is based on the combined complexity of the actors in all the use cases.
- ❖ Unadjusted Use Case Weight (UUCW) which is based on the number of activities contained in the use case scenarios.

There are also two complexity factors in this equation: technical and environmental. The technical complexity factor represents the challenge of implementing nonfunctional requirements of a system. There are two constants used with the technical complexity equation, the first being $C_1 = 0.6$ and the second being $C_2 = 0.01$. Meanwhile the environmental complexity represents the miscellaneous factors such as experience and time management. There are also two constants used for the environmental complexity equation which are $C_1 = 1.4$ and $C_2 = -0.03$. The complexity factor equation is as below:

$$CF = c_1 + c_2 \sum_{i=1}^{13} w_i F_i$$

6.2 Unadjusted Actor Weight

Actor Name	Description	Complexity	Weight
Guest	User who has not yet registered or been authenticated by the system	Simple	1
Manager	User who has been registered as a manager account	Complex	3
Trainee	User who has been registered as a trainee account	Complex	3
Browser	Allows interaction between user and system	Simple	1
CommodityAPI	Provides up-to-date commodity prices for the system	Average	2
Database	Holds all information of current user accounts and their portfolios	Average	2
TimerTask	Task that executes after a certain time period has been met	Simple	1

6.3 Unadjusted Use Case Weight

Use Case	Description	Complexity	Weight
Buy Commodity UC - 1	To buy a commodity. This involves the Trainee filling out and submitting a buy order ticket of a specified commodity and its market price queried from Commodity API. The order ticket will be processed and its data saved in the Database. Changes to the Trainee's portfolio will be reflected.	Complex	15
Sell Commodity UC - 2	To sell a commodity. This involves the Trainee filling out and submitting a sell order ticket of a specified commodity at the current market price queried from Commodity API. The order ticket will be processed and its data saved in	Complex	15

	the Database. Changes to the Trainee's portfolio will be reflected.		
Register as Trainee UC - 3	To register an account as a Trainee. This involves the Guest to fill out the Trainee registration form with their user information. The Database will check to make sure that the user name submitted is unique and create an account for the Guest.	Simple	5
View Portfolio UC - 4	To view the current status of an account's portfolio. This involves the User selecting an account to display the current portfolio information retrieved from the Database. The Database will update the holdings and account total with the latest values retrieved from the Commodity API.	Average	10
End of Training Feedback UC - 14	To comment on the performance report of a managed Trainee at the end of their training regiment. This involves the Manager to access and review a Trainee's performance report as well as their transaction history to provide feedback for the Trainee.	Simple	5

6.4 Technical Complexity Factor

Technical Factor	Description	Weight	Perceived Complexity	Calculated Factor
Distributed System	Distributed system between end users having access through web and main server.	2	5	10
System Performance	System performance expected to be good but nothing exceptional.	1	3	3
User Efficiency	End users expect efficiency but no exceptional demands.	1	3	3
Reusability	No requirements for the system to be reusable.	0.5	0	0

Ease of Use	Ease of use for users is imperative.	0.5	3	1.5
Ease of Change	System should only change marginally, so ease of change is a low priority.	1	1	1
Concurrent Use	Concurrency is an issue due to users having access to activity, history, and the system has to poll data.	2	5	10
Security	Security is important but does not require extreme efforts.	1	2	2
Training Requirements	System should be easy to use, but basic tutorials are available to users.	1	1	1

6.5 Environmental Complexity Factor

Environmental Factor	Description	Weight	Perceived Complexity	Calculated Factor
Application Experience	Some novices to the field of finance and some have experience.	0.5	0	0
Development Experience	Some competent with UML-based development and construction processes as well as some beginners.	1.5	1.5	2.25
Motivation	Motivation is high to finish project but fluctuates over the semester	1	3	3
Stable Requirements	Requirements are defined but only approximate.	2	3	6
Time Management	All developers are working very few hours a week due to jobs and other classwork.	-1	5	-5
Language	Developers are using a collection of modern languages where some are familiar and some are not.	-1	2	-2

6.6 Calculations

$$UAW = \# \text{ of Simple} \times 1 + \# \text{ of Average} \times 2 + \# \text{ of Complex} \times 3 = 3 \times 1 + 2 \times 2 + 2 \times 3 = 13$$

$$UUCP = \# \text{ of Simple} \times 5 + \# \text{ of Average} \times 10 + \# \text{ of Complex} \times 15 = 2 \times 5 + 1 \times 10 + 2 \times 15 = 50$$

$$TCP = 0.6 + 0.01 * (\text{Technical Factor Total}) = 0.6 + 0.01 * (31.5) = 0.915$$

$$ECP = 1.4 - 0.03 * (\text{Environmental Factor Total}) = 1.4 - 0.03 * (4.25) = 1.2725$$

$$UCP = (UAW + UUCW) \times TCF \times ECF = (13 + 50) \times 0.915 \times 1.2725 = 73.35$$

$$\text{Duration} = UCP * PF \text{ (productivity factor)} = 73.35 * 28 = 2053.80$$

7 Analysis and Domain Modeling

7.1 Conceptual Model

Concept Definitions

The Domain Model Concepts are derived from the elaborated Use Cases described in Report #1.

Responsibility	Type	Concept
R1: Let new users create a trainee account to partake in exercises for stock trading training. Login existing user.	D	Account Controller
R2: Initialize new accounts with a fixed balance.	D	Account Controller
R3: View the user's portfolio and display info about stocks, trades, and balance.	K	User Controller
R4: Retrieve information about stocks, trades, and commodities.	K	Commodity API
R5: Buy/Sell Stock trades and record into the database.	D	System Controller
R6: Display Home screen to create an account or login	K	Home View
R7: Display end of training feedback to user	K	User Controller

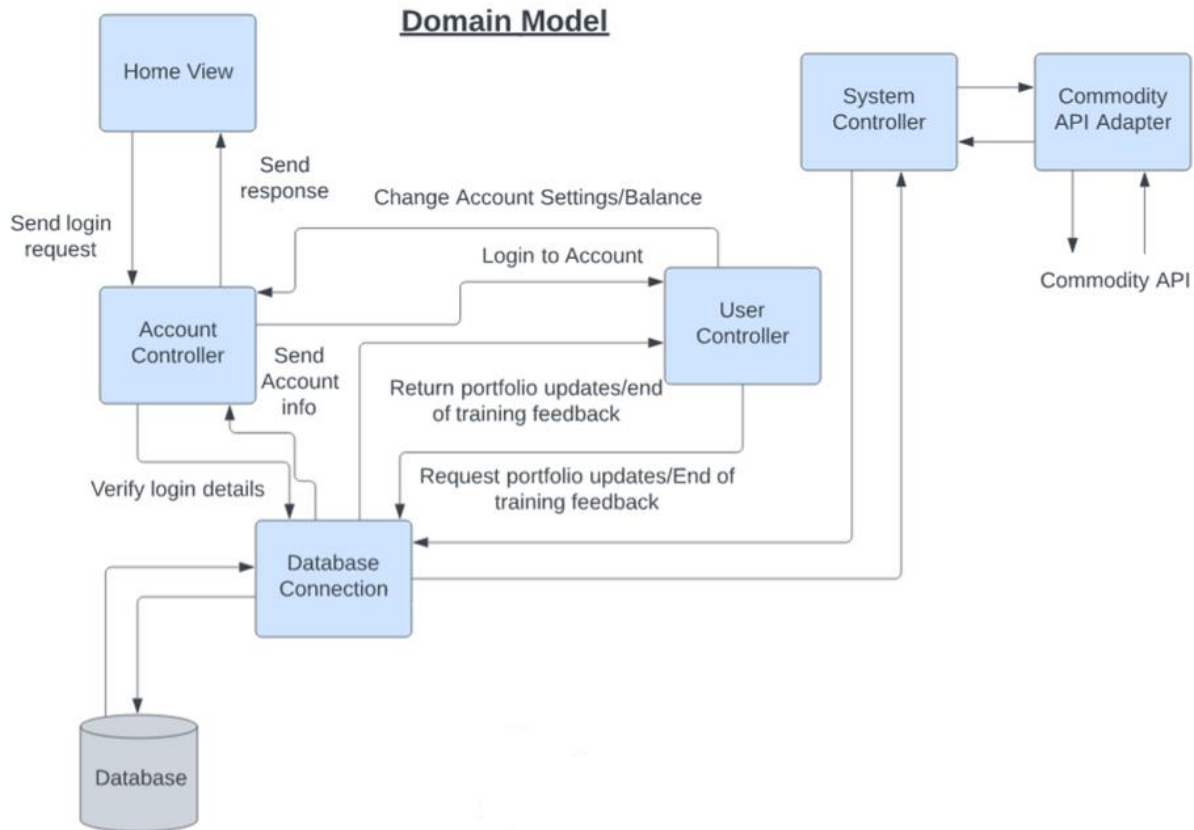


Figure 7.1 The Domain Model

Account Controller

In order to access this system at all, one has to make an account. Account creation will happen on the login screen and will check to see if an account has already been made with those credentials and if there if not it will proceed to make either a trainee or manager account depending on the preference. These details are then stored into the database.

User Controller

The User Controller accesses users' portfolios and displays relevant information such as stock, balance, and trade history. User Controller should also pull end of training feedback once training has been completed by a trainee. It will query the Commodity API to pull relevant information about any searched stocks as needed.

Commodity API

The Commodity API provides the almost real time stock data that AgriCom Training is dependent on for training new stock market users. We will have to access the market API through an adaptor that serves as a translator between our server and the API. If we did not have access to this data, the stock market training would not be realistic as it would not have real time information.

System Controller

Any order that is placed will have to go through the System Controller as it will have to record the order and input it into the database so we have a history to pull later and are able to make calculations based on account balances. This system will also have to communicate with the Commodity API as it will require current stock prices in order to record accurate account balances and if the trade is allowed under how much funds the account has.

Home View

The Home View displays a UI that allows the user to create an account or login with existing credentials. This will send a request to the Account Controller to pull or add to the database and if it fails it will reflect this on the Home View or if it succeeds the user will proceed to the Portfolio.

Association Definitions

Concept Pair	Association Description	Association Name
Home View <> Account Controller	Home View sends a login request. Account controller responds with success or failure.	Sends
Account Controller <> Database Connection	Account Controller sends user login details. Database sends account information or login failure.	Sends

Account Controller <> User Controller	Account controller shows that user logs into account and is able to access information and make changes. User controller makes those changes and sends them back to the Account Controller.	Updates
User Controller <> Database Connection	User Controller requests portfolio updates from database or end of training review. Database sends required information over to User controller.	Sends, Updates
System Controller <> Database Connection	System Controller Buy/Sell stocks and send information to Database for storage.	Sends, Updates
System Controller <> Commodity API	System Controller sends requests to API about stock prices. API returns stock information.	Sends

The associations of domain concepts are derived from the table above. First, the Account Controller takes the information that a user enters in the Home View and verifies it with the Database to see if the login is valid. Once that is validated, the Account Controller can change the view over to the Portfolio based on information that is stored in the database. The User Controller allows them to navigate the site and request information located on the site and updates from the database as needed. When a user makes a trade, the System Controller steps in and Buy/Sells stocks and sends that information to the database for storage while also pulling current stock information and prices from the Commodity API for use. The System Controller must see if a user has sufficient funds however for a Buy/Sell action.

Attribute Definitions

Responsibility	Attribute	Concept
R9: Know if a user login failed	LoginFailed	Home View

R10: Know if a user is logged in	isLoggedIn	Portfolio
R11: User Name	userName	Portfolio
R12: Account Balances	Account Summary	Portfolio
R13: Stock Bought	buyStock	System Controller
R14: Stock Sold	sellStock	System Controller
R15: Manager Account	isManager	Account Controller
R16: Trainee Account	isTrainee	Account Controller
R17: Reached end of training	endOfTraining	User Controller

At the Home View it is very important to know if a login attempt failed as we can notify the user that way. When the login attempt is successful it will set the attribute of isLoggedIn in the Account Controller which then interacts with the database to pull all relevant user information to populate the user Portfolio.

At the Portfolio view a user may attempt things such as making a buy/sell order, view account history, or view account information. The Account Summary will show the user balances on the Portfolio view and the System Controller will store previous stocks bought or sold for the database to be able to pull here for the history.

Once the user has made it this far into the system they are ready to make an order. The System Controller will interact with the Commodity API and retrieve data for a particular stock such as its current price or price history and then report it back to the System Controller so it can perform the action of Buy/Sell on the stock.

The Database Connection also should have a retrieveData attribute to be able to read data from the database. There should also be the inverse of that attribute so that it can store data about the user and their profiles such as writeData.

Traceability Matrix

Use Case	PW	Account Controller	User Controller	System Controller	Commodity API	Home View	Database Connection
UC 1	5		X	X	X		X
UC 2	5		X	X	X		X
UC 3	4	X				X	X
UC 4	5	X	X				X
UC 9	4		X	X	X		X
UC 14	3		X				X
Max PW		5	5	5	5	5	5
Total PW		9	22	14	14	4	26

Figure 7.2 Traceability Matrix

We can see here that the Database Connection is applicable to the majority of Use Cases and thus have the highest Power Weight. The next most important is the User Controller as it handles the user's account's portfolio and its price and holdings. The next are System Controller and Commodity API, as they will be integral to making the ordering system work.

7.2 System Operation Contracts

UC-1 Buy Commodity	
Preconditions:	<ul style="list-style-type: none"> Account Controller's LoginStatus = Trainee Logged In Page Renderer's ViewType = Commodity View Order System Controller's OrderType = Buy Database Connection's IsConnected = Success Commodity API Handler's RetrievalStatus = Success
Postconditions:	<ul style="list-style-type: none"> Order System Controller's ValidOrder = Success/Fail Database is updated if ValidOrder = Success.

UC-2 Sell Commodity	
Preconditions:	<ul style="list-style-type: none"> Account Controller's LoginStatus = Trainee Logged In Page Renderer's ViewType = Commodity View Order System Controller's OrderType = Sell Database Connection's IsConnected = Success Commodity API Handler's RetrievalStatus = Success
Postconditions:	<ul style="list-style-type: none"> Order System Controller's ValidOrder = Success/Fail Database is updated if ValidOrder = Success.

UC-3 Register as Trainee	
Preconditions:	<ul style="list-style-type: none"> An unregistered Trainee (Guest) visits the system. Account Controller's LoginStatus = NULL or Logged Out Page Renderer's ViewType = Login View Database Connection's IsConnected = Success
Postconditions:	<ul style="list-style-type: none"> Account Controller's AccountCreationStatus = Success/Fail Database is updated if AccountCreationStatus = Success.

UC-4 View Portfolio	
Preconditions:	<ul style="list-style-type: none"> Account Controller's LoginStatus = Trainee Logged In Database Connection's IsConnected = Success Commodity API Handler's RetrievalStatus = Success
Postconditions:	<ul style="list-style-type: none"> Account information is retrieved from the Database and updated with information from Commodity API Handler. Page Renderer's ViewType = Trainee Profile View

UC-14 Provide end of training Feedback	
Preconditions:	<ul style="list-style-type: none"> Account Controller's LoginStatus = Manager Logged In Database Connection's IsConnected = Success Database's ManagedTraineeID is not NULL Database's Trainee account's TimerTask is expired
Postconditions:	<ul style="list-style-type: none"> Database updates comments for the trainee to review

7.3 Data Model and Persistent Storage

Data is a fundamental component of the AgriCom Training System and therefore will utilize a series of SQL relational database tables to persist a majority of the required user and trading data. Currently this data will exist in set 10 different structured SQL tables. Each table can be used for multiple portions of the training system as detailed below. The relational data diagram below details the different tables and includes the field names and data types:

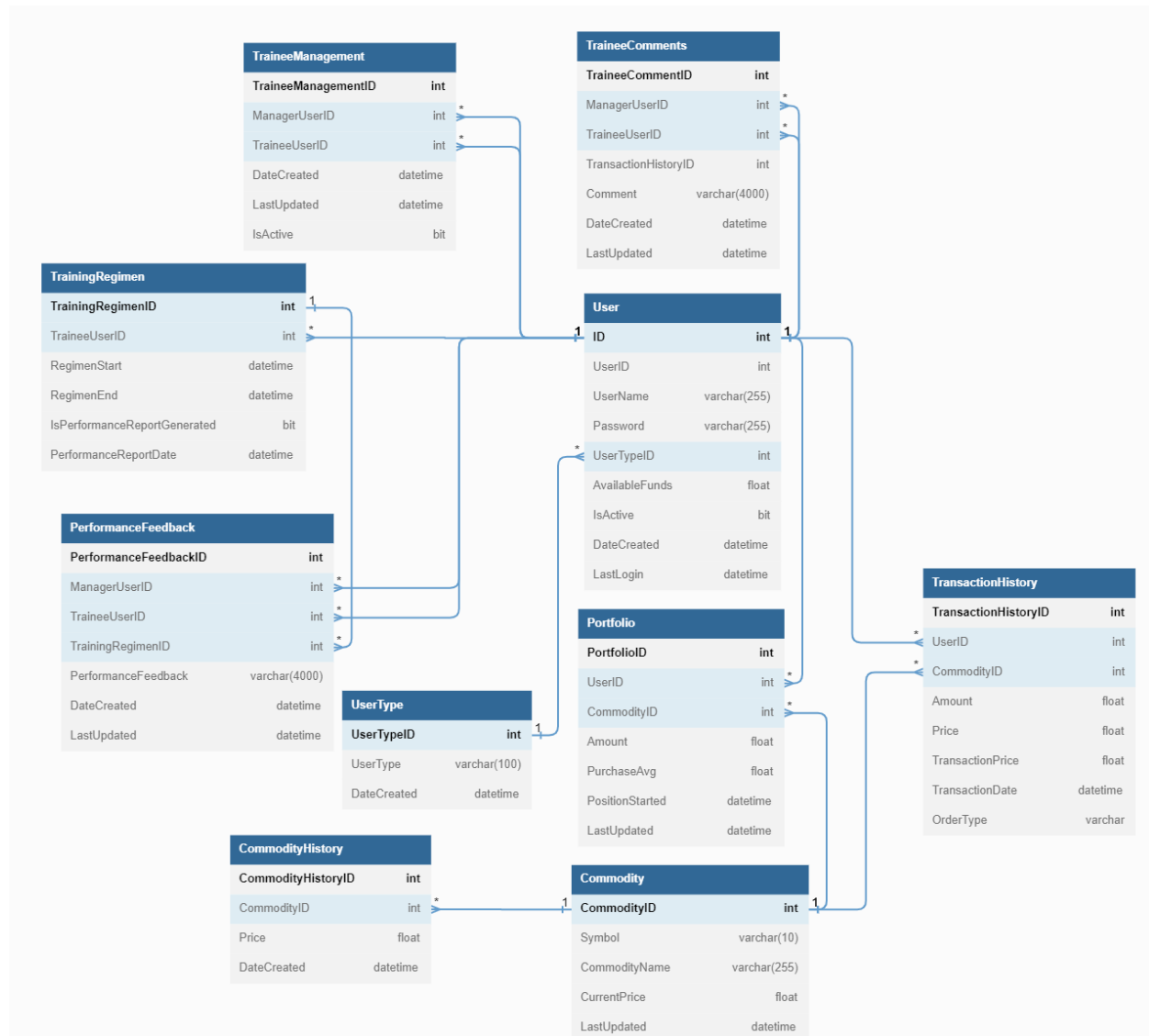


Figure 7.3 Database Tables

There are 2 central tables within this database schema that will be utilized as the core of the data system. These main tables are the User and Commodity tables. The User table will house all of the registered user information including their username , password, user type,

current available funds, the date the user was created, the last login date, and if they are currently active. The Commodity table will store the various commodities that can be traded. This includes the symbol, name, current listed price, and last updated date. When the commodity API is called to pull the current price information the Commodity table is where this data will be stored.

The remaining tables outside of the central tables above are the satellite tables that store a variety of different data from configurations, user relationships, comment information, and history data. The UserType table will store the different types of user accounts to be associated with the User table. Currently the types planned at this time are trainee and manager but having a lookup table will allow many other user types to be created without impacting the established schemas.

Each user will have their current account holdings stored in the Portfolio table. This table will maintain the commodity id, amount currently held, the purchase average, data the position was started, and the last updated date for the commodity in their portfolio. This table, along with the Commodity and User table, will also be used to calculate a given user's current account balance from the product of each held commodities amount * current price in addition to the users' current available funds.

TrainingRegimen will store the trainee users' start and end date for their monthly training regimen. This also includes a flag for if their performance report has been generated for the given regimen as well as the date this report has been generated.

The TraineeManagement table will store the relationships between trainee users and manager users. As managers add trainees to their list of users to manage, this table will be updated with their manager user id, the trainee user id, the date the relationship was established, if the relationship is currently active, and the date the relationship was last updated. This will allow the system to retain previous manager to trainee relationships even if a trainee is no longer being monitored by a manager user going forward. This setup also allows for multiple manager accounts to be monitoring the same trainee accounts rather than only allowing one trainee to one manager relationship.

There are two tables created for the storage of comments and feedback at both a transactional and performance level. The TraineeComments table will hold comments from either manager and/or trainee accounts that are created for transactions that the trainee account has made. TransactionHistoryID is included in this table to maintain the relation to the TransactionHistory table. The PerformanceFeedback table, however, will store managerial comments for a trainee at the end of their monthly training regimen. TrainingRegimenID is included in this table to maintain the relationship back to the TrainingRegimen table.

As far as history tables are concerned, there are two different tables created for retaining historical information, CommodityHistory and TransactionHistory. CommodityHistory will contain previous commodity price information so that as new prices are pulled into the system the previously retrieved values will be stored. TransactionHistory will include all the transactions for

a given user. Every transaction that a user places will be stored in the TransactionHistory table and will include the commodity id, amount, transaction price, and transaction date.

7.4 Mathematical Model

Commodity Prices

There are no complicated mathematical models behind how the commodity prices are determined on our platform. The market prices that are retrieved from Commodity API Handler are the current price in the given update timeframe allowed by the API provider.

End-of-Training Regiment Report

The report at the end of a training regiment has two mathematical models. There are no complicated algorithms behind how the report is calculated. The report will provide a simple statistical summary of the Trainee's account and transactions, such as percentage gain or loss, average account balance, and average gain or loss per transaction.

Percentage gain or loss

$$(End\ Total - Start\ Total) \div Start\ Total$$

Average account balance

$$\sum_{i=1}^{Number\ of\ Days} \div Number\ of\ Days$$

Average gain or loss per transaction

$$(End\ Total - Start\ Total) \div Number\ of\ Trades$$

8 Interaction Diagrams

Use Case 1

The workflow for how to buy a commodity is shown in the diagram for UC-1. A trainee that is logged into the system can click the "Trade" link which will call the System Controller to prompt the trainee for the order type, commodity name/symbol, and amount wished to be purchased. The trainee will then select a valid commodity and enter in an amount which is returned to the System Controller and the system then calls the Commodity API to query the current market price and returns the total cost so that the trainee can confirm the order. Once confirmed this information will be saved into the database via the DB connection and a notification is returned to the trainee from the System Controller that the transaction has been completed.

Not shown are workflows for two alternate scenarios which cause an error notification to return to the trainee. In alternate scenario 1 the trainee attempts to enter an invalid commodity name or symbol and the system returns a notification that the name or symbol is invalid. In alternate scenario 2 the trainee attempts to complete a purchase using more funds than the trainee currently has in their portfolio. This again is returned with a system notification alerting the trainee that there is not enough funds to complete the purchase.

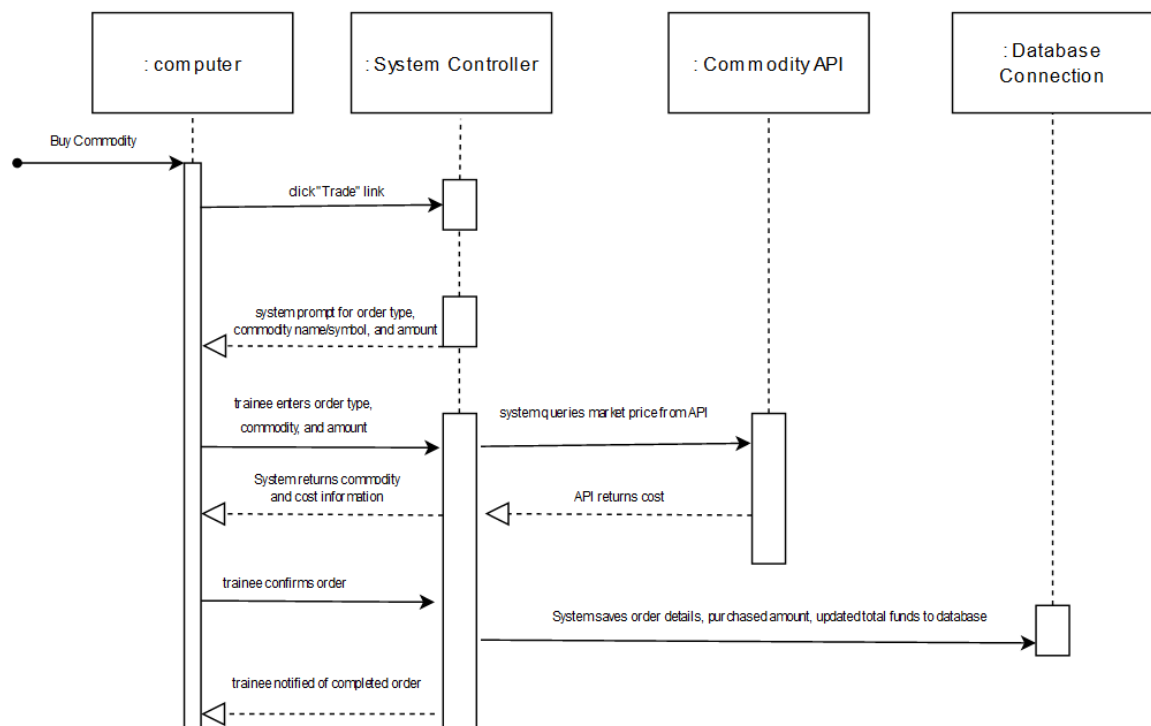


Figure 8.1: Use Case 1 Diagram

Use Case 2

The workflow for how to sell a commodity is shown in the diagram for UC-2. A trainee that is logged into the system can click the "Trade" link which will call the System Controller to prompt the trainee for the Order Type, Commodity, and amount wished to be sold. The trainee will then select a valid commodity and enter in an amount wished to be sold. This is returned to the System Controller and the system then calls the Commodity API to query the current market price. The projected total return from the sale is returned so that the trainee can confirm the order. Once confirmed this information will be saved into the database via DB Connection and a notification is returned to the trainee from the System Controller that the transaction has been completed.

Not shown are workflows for two alternate scenarios which cause an error notification to return to the trainee. In alternate scenario 1 the trainee attempts to enter an invalid commodity name or symbol and the system returns a notification that the name or symbol is invalid. In alternate scenario 2 the trainee attempts to sell a larger amount of shares than the trainee currently has in their portfolio. This again is returned with a system notification alerting the trainee to enter in an available amount of shares to be sold.

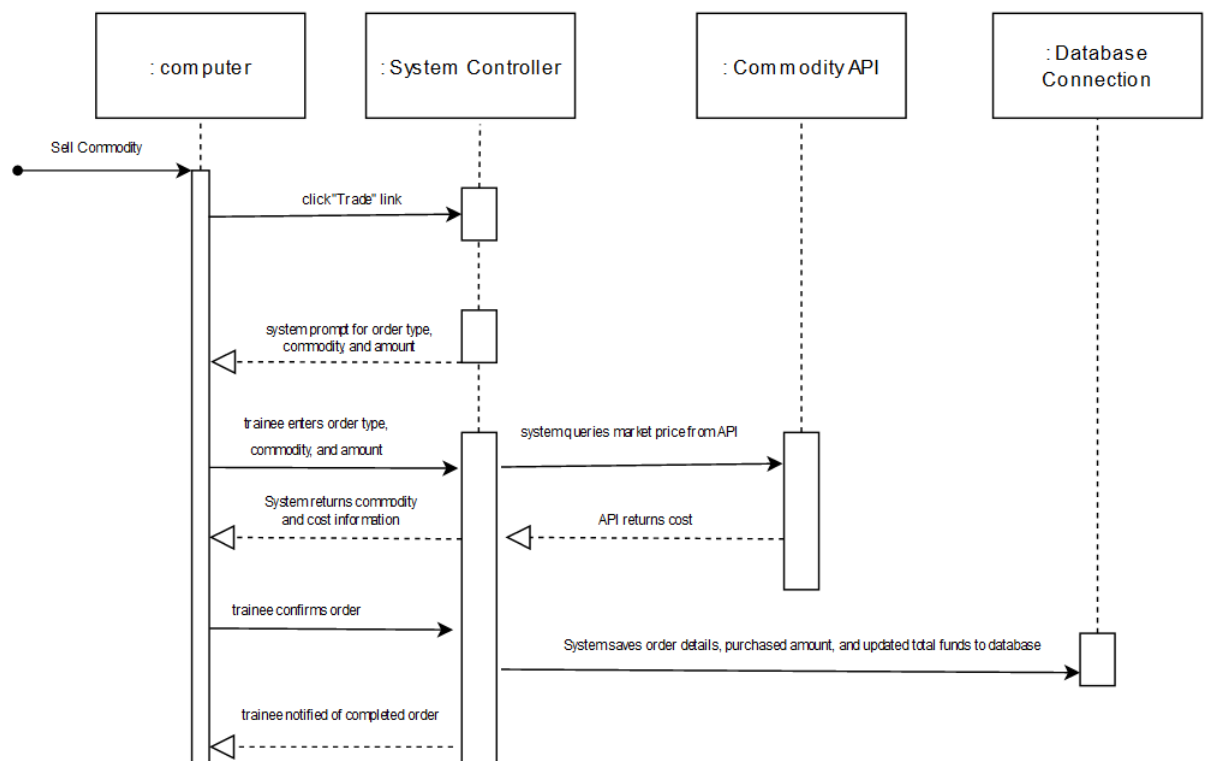


Figure 8.2: Use Case 2 Diagram

Use Case 3

The workflow for how to register as a trainee is shown in the diagram for UC-3. As a guest within the system, the guest user will click on the "Sign Up" link which calls the System Controller and a Trainee Registration Form is returned to the guest for their user information. The guest then enters the information into the Trainee registration form and once submitted, the System Controller will verify that the user name entered is unique via DB Connection. If unique, the System Controller will save the new user into the database via DB Connection and provide a notification to the now registered trainee that the account was successfully created.

Not shown in the workflow is an alternate scenario which causes an error notification to return to the guest attempting to register. In alternate scenario 1 the guest attempts to create a trainee account with a user name that already exists in the database. The system then returns a notification for the guest to attempt to register with a different user name.

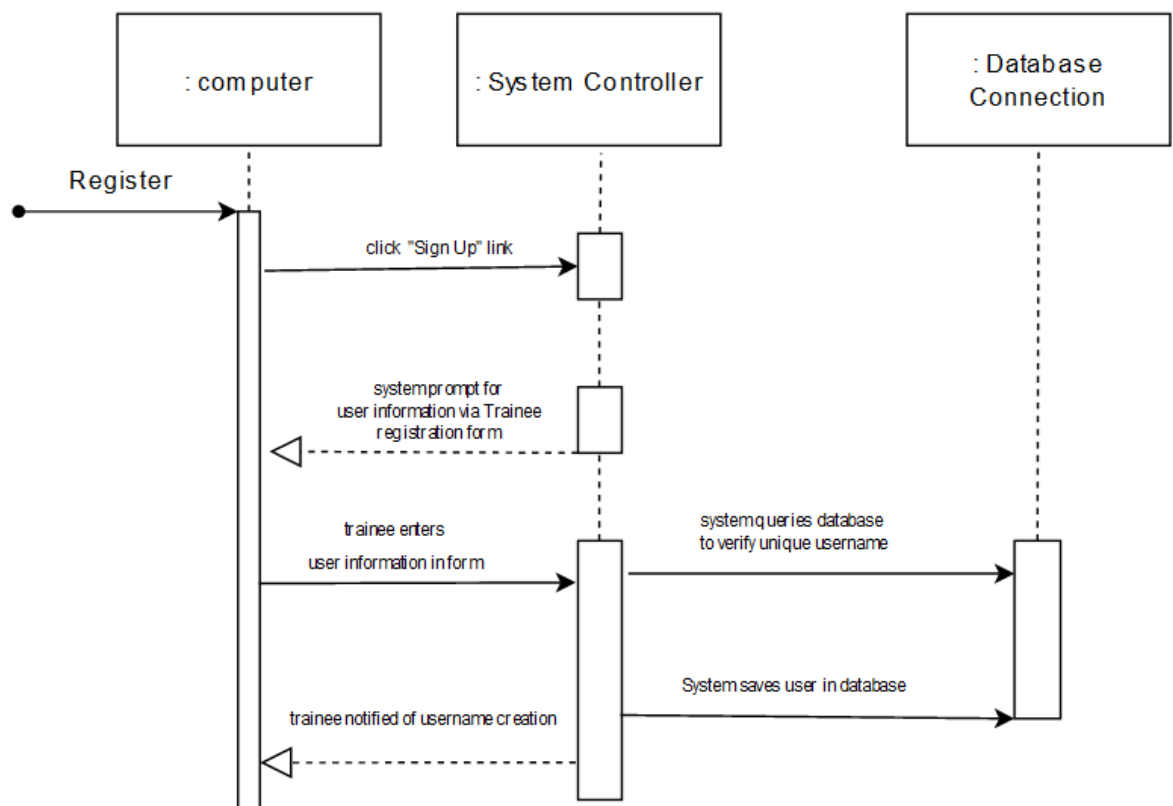


Figure 8.3: Use Case 3 Diagram

Use Case 4

The workflow for how to view a portfolio is shown in the diagram for UC-4. A trainee that is logged into the system can click on the "Portfolio" link. If the user is a manager then they will first select the managed trainee account they wish to view before clicking on the "Portfolio" link. In both scenarios the "Portfolio" link will call the System Controller which will then call the Commodity API to query the most recent commodity prices. This is returned to the System Controller which stores this information in the database via DB Connection. Afterwards the trainee, or selected trainee, account information will be pulled from the database and the System Controller will display the current portfolio information back to the trainee or manager user.

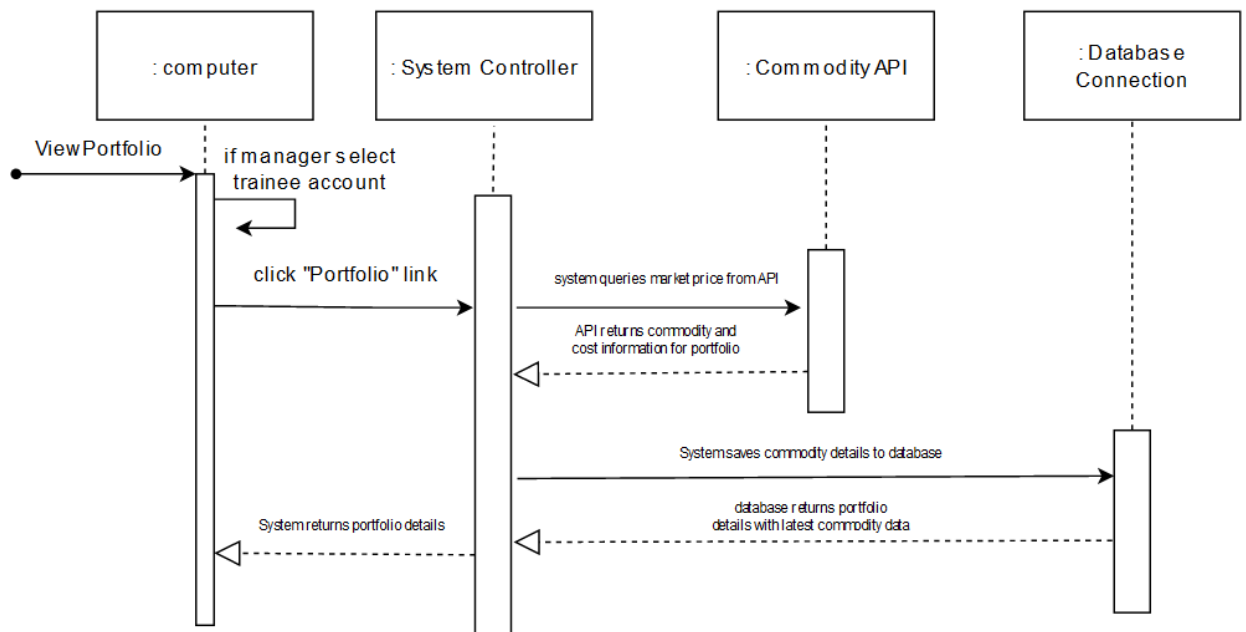


Figure 8.4: Use Case 4 Diagram

Use Case 14

The workflow for how end of training feedback is provided is shown in the diagram for UC-14. A manager that is logged into the system will select a trainee account from their management list and click on the "View Performance Report" link. The System Controller will then query for the selected account's portfolio and historical transactions from the database via DB Connection which is returned back to the System Controller which then returns the performance report information back to the manager user. After reviewing this information, the manager is then able to click the "Provider Feedback" link and the System Controller will prompt for feedback on the trainee accounts performance. The manager user will enter in their feedback and click "Submit" which calls the System Controller to save the feedback into the database via DB Connection and then returns a system notification to the manager that the feedback has been successfully saved.

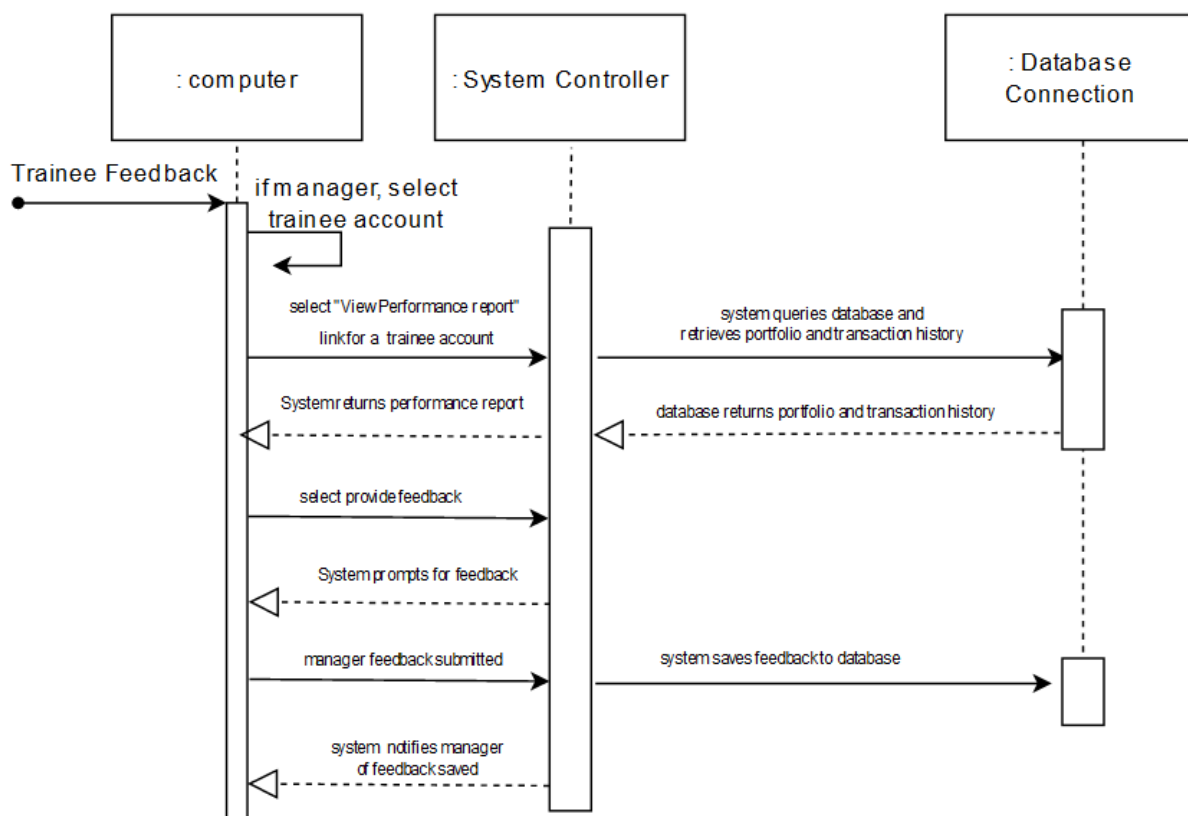


Figure 8.5: Use Case 14 Diagram

9 Class Diagram and Interface Specification

9.1 Class Diagram

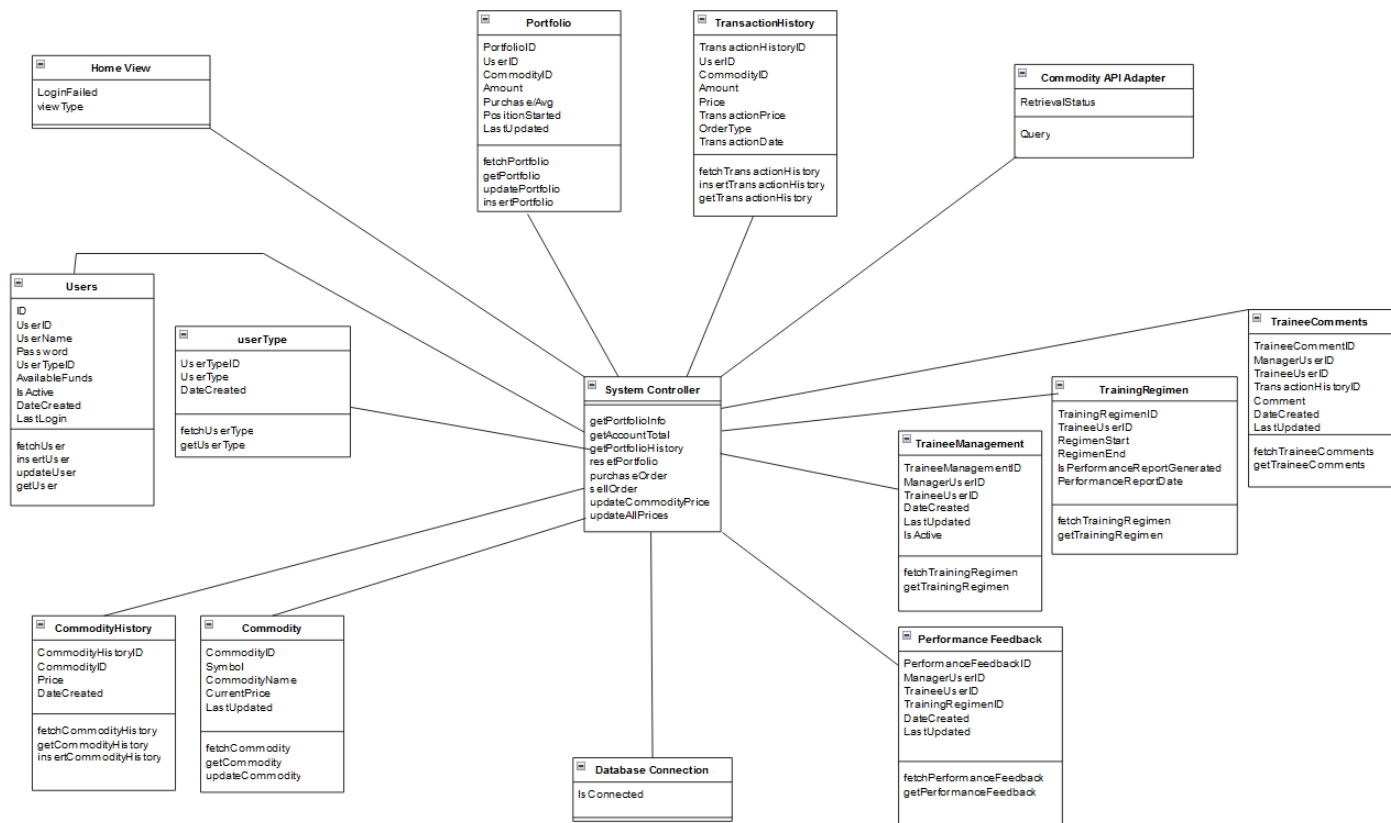


Figure 9.1: Class Diagram

9.2 Data Types and Operation Signatures

❖ Home View

➤ Attributes

■ LoginFailed : Boolean

- Lets the user know if a login attempt failed

■ viewType : string

- tells the page renderer which view to display choosing between the login screen, commodity screen, management screen, or trainee profile.

❖ systemController

➤ Operations

- **getPortfolioInfo(Integer: UserID, String: Symbol, String: CommodityName, Integer: Amount, Double: PurchaseAvg, Double: CurrentPrice, Double: TotalValue)**
 - pull the portfolio data with the total value
- **getAccountTotal(Portfolio) : PortfolioTotal**
 - with portfolio, return total value of available funds
- **getPortfolioHistory(Portfolio) : PortfolioHistory**
 - with portfolio, return portfolio history
- **resetPortfolio(Portfolio) : Portfolio**
 - with portfolio, reset portfolio
- **purchaseOrder(User, Portfolio, Commodity, Transaction) : TransactionHistory**
 - execute a purchase order and update transaction history
- **sellOrder(User, Portfolio, Commodity, Transaction) : TransactionHistory**
 - execute a sell order and update transaction history
- **updateCommodityPrice(Symbol, Commodity) : CommodityPricing**
 - update the commodity pricing for a particular commodity using the symbol
- **updateAllPrices(Commodity) : CommodityPricing**
 - update all commodity pricing

❖ **Commodity**

➤ **Attributes**

- **CommodityID : Integer**
- **Symbol : String**
- **CommodityName : String**
- **CurrentPrice : Double**
- **LastUpdated : Date**

➤ **Operations**

- **fetchCommodity(): CommodityData**
 - returns all commodities
- **getCommodities(String: Commodity): CommodityData**
 - return specific commodity
- **updateCommodity(String: Commodity): Commodity Price**
 - update pricing for a commodity

❖ **CommodityHistory**

➤ **Attributes**

- **CommodityHistoryID : Integer**
- **CommodityID : Integer**
- **Price : Double**
- **DateCreated : Date**

➤ **Operations**

- **fetchCommodityHistory(): CommodityData**
 - returns all history

- **getCommodityHistory(String: Commodity) : CommodityData**
 - return history of specific commodity
 - **insertCommodityHistory(String: Commodity) : CommodityData**
 - find the current price information for commodity and insert new data into history table
- ❖ **PerformanceFeedback**
 - **Attributes**
 - **PerformanceFeedbackID**
 - **ManagerUserID : Integer**
 - **TraineeUserID : Integer**
 - **TrainingRegimenID : Integer**
 - **DateCreated : Date**
 - **LastUpdated : Date**
 - **Operations**
 - **fetchPerformanceFeedback() : CommodityData**
 - returns all performance feedback
 - **getPerformanceFeedback(Integer: PerformanceFeedbackID, Integer: ManagerUserID, Integer: TraineeUserID) : PerformanceFeedback**
 - returns feedback for specific id, manager, and trainee
- ❖ **Portfolio**
 - **Attributes**
 - **PortfolioID : Integer**
 - **UserID : Integer**
 - **CommodityID : Integer**
 - **Amount : Integer**
 - **PurchaseAvg : Double**
 - **PositionStarted : Double**
 - **LastUpdated : Date**
 - **Operations**
 - **fetchPortfolio() : Portfolio**
 - return all portfolio
 - **getPortfolio(Integer: UserID, Integer: PortfolioID) : Portfolio**
 - returns portfolio of specific user
 - **updatePortfolio(Integer: UserID, Integer: PortfolioID, Double: Amount, Double : PurchaseAvg, Double: PositionStarted, Date: LastUpdated) : Portfolio**
 - updates existing portfolio of a user
 - **insertPortfolio(Integer: UserID, Integer: PortfolioID, Double: Amount, Double : PurchaseAvg, Double: PositionStarted, Date: LastUpdated): Portfolio**
 - Enters a new portfolio object into the database
- ❖ **TraineeComments**
 - **Attributes**
 - **TraineeCommentID : Integer**

- ManagerUserID : Integer
 - TraineeUserID : Integer
 - TransactionHistoryID : Integer
 - Comment : String
 - DateCreated : Date
 - LastUpdated : Date
- Operations
 - fetchTraineeComments() : TraineeComments
 - return all trainee comments
 - getTraineeComments(Integer: TraineeCommentID, Integer: ManagerUserID: Integer, Integer: TraineeUserID)
 - return trainee comments for a specific person
- ❖ TraineeManagement
 - Attributes
 - TraineeManagementID : Integer
 - ManagerUserID : Integer
 - TraineeUserID : Integer
 - DateCreated : Date
 - LastUpdated : Date
 - IsActive : Boolean
 - Operations
 - fetchTraineeManagement() : TraineeManagement
 - return all trainees under all managers
 - getTraineeManagement(Integer: TraineeManagementID, Integer: ManagerUserID, Integer: TraineeUserID)
 - Get trainees listed under manager
- ❖ TrainingRegimen
 - Attributes
 - TrainingRegimenID : Integer
 - TraineeUserID : Integer
 - RegimenStart : Date
 - RegimenEnd : Date
 - IsPerformanceReportGenerated : Boolean
 - PerformanceReportDate : date
 - Operations
 - fetchTrainingRegimen() : TrainingRegimen
 - return all training regimens for all users
 - getTrainingRegimen(Integer: TrainingRegimenID, Integer: TraineeUserID)
 - return training regimen for a specific trainee
- ❖ TransactionHistory
 - Attributes
 - TransactionHistoryID : Integer
 - UserID : Integer

- CommodityID : Integer
 - Amount : Integer
 - Price : Double
 - TransactionPrice : Double
 - OrderType : String
 - TransactionDate : Date
- Operations
 - fetchTransactionHistory() : TransactionHistory
 - return all transaction history for all users
 - insertTransactionHistory(Integer: UserID, Integer: CommodityID, Integer: Amount, Double: Price, Double: TransactionPrice, String: OrderType, Date: TransactionDate) : TransactionHistory
 - insert a new line into transaction history for a specific user
 - getTransactionHistory(Integer: UserID, Integer: CommodityID, Integer: Amount, Double: Price, Double: TransactionPrice, String: OrderType, Date: TransactionDate) : TransactionHistory)
 - return transaction history for a specific user
- ❖ userType
 - Attributes
 - UserID : Integer
 - UserType : String
 - DateCreated : Date
 - Operations
 - fetchUserType() : UserType
 - Returns all user types
 - getUserType(Integer: UserID, String: UserType, Date: DateCreated)
 - returns user type for a particular user type id
- ❖ Users
 - Attributes
 - ID : Integer
 - UserID : Integer
 - UserName : String
 - Password : String
 - UserID : Integer
 - AvailableFunds : Double
 - IsActive : Boolean
 - DateCreated : Date
 - LastLogin : Date
 - Operations
 - fetchUser() : user
 - returns all users
 - insertUser(Integer: UserID, String: UserName, String: Password, Integer: UserID, Double: AvailableFunds, Boolean: IsActive)

- insert new user into the database
- updateUser(String: UserName, String: Password, Integer: UserID, Double: AvailableFunds, Boolean: IsActive, Date: LastLogin, Integer: UserID)
 - update an existing user in the database
- getUser(Integer: ID, Integer: UserID, String: UserName, String: Password, Integer: UserID, Double: AvailableFunds, Boolean: IsActive, Date: DateCreated, Date: LastLogin)
 - retrieve an existing user details from the database
- ❖ Database Connection
 - Attributes
 - isConnected : Boolean
 - This value holds the success/fail value if the database connected properly
- ❖ Commodity API Adapter
 - Attributes
 - RetrievalStatus
 - This value holds the success/fail value if the API connected properly
 - Operations
 - Query(String: commodity):CommodityData
 - This function retrieves the market commodities data from the internet

9.3 Traceability Matrix

REQ	PW	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC13	UC14
REQ-1	5			X							X				
REQ-2	5	X	X												
REQ-3	5	X	X												
REQ-4	5	X	X	X	X	X	X	X	X	X	X	X	X	X	X
REQ-5	5	X	X		X				X	X					
REQ-6	5	X	X	X	X	X	X	X				X	X		X
REQ-7	4											X	X	X	X
REQ-8	4														X
REQ-9	3													X	
REQ-10	2						X	X							
REQ-11	1								X						
Max PW		5	5	5	5	5	5	5	5	5	5	5	5	5	5
Total PW		25	25	15	15	10	12	12	11	10	10	14	14	12	18

9.4 Design Patterns

Our project AgriCom Training has used a few design patterns throughout. The most consistently used ones are the Command pattern and the Publisher-Subscriber pattern. The largest one with the Pub-Sub pattern is the systemController as it has many operations that are subscribers to one controller that receives the key for the subscribers to check. Other examples would be Users as there are also many functions that listen for one key that is sent to the controller. For the Command pattern, this is seen in the sell order as the user chooses when to make the order but does not change anything available. This is also the case in the user login sessions. Another design pattern used is the Proxy pattern, this is used when a session login id is needed on the site and grants permissions based on the login id type registered.

9.5 Object Constraint Language (Contracts)

context homeView::fetchUser() : boolean

pre: User is not currently logged in

post: User has successfully logged in.

context systemController::getPortfolioInfo() : Array

pre: Data exists in Portfolio for User

post: Array of portfolio items is returned

context systemController::getAccountTotal() : int

pre: Data exists in Portfolio for User

post: Total value of portfolio is summed with current funds

context systemController::updateAllPrices() : boolean

pre: Valid CommodityAPI access key is available

post: All commodity prices are updated in the database

context commodity::fetchCommodity() : Commodity

pre: Commodity table contains values

post: Commodity object, or array of Commodity objects is returned

context commodity::updateCommodity() : Commodity

pre: Valid CommodityAPI access key is available, Commodity exists in Commodity table, Commodity object contains valid values

post: Commodity price is updated in the database and Commodity object is returned

context commodityHistory::fetchCommodityHistory() : CommodityHistory

pre: CommodityHistory table contains values

post: CommodityHistory object, or array of CommodityHistory objects is returned

context portfolio::fetchPortfolio() : Portfolio

pre: Portfolio table contains values

post: Portfolio object, or array of Portfolio objects is returned

context portfolio::insertPortfolio() : boolean

pre: Portfolio object contains valid values

post: Portfolio object is stored in the database

context portfolio::updatePortfolio() : boolean

pre: Portfolio exists in Portfolio table and Portfolio object contains valid values

post: Portfolio records updated in the database

context portfolio::getPortfolio() : Portfolio

pre: Valid array is passed

post: Portfolio object is returned from array values

context traineeComments::fetchTraineeComments() : TraineeComments

pre: TraineeComments table contains values

post: TraineeComments object, or array of TraineeComments objects is returned

context traineeComments::getTraineeComments() : TraineeComments

pre: Valid array is passed

post: TraineeComments object is returned from array values

context traineeManagement::fetchTraineeManagement() : TraineeManagement

pre: TraineeManagement table contains values

post: TraineeManagement object, or array of TraineeManagement objects is returned

context traineeManagement::getTraineeManagement() : TraineeManagement

pre: Valid array is passed

post: TraineeManagement object is returned from array values

context transactionHistory::fetchTransactionHistory() : TransactionHistory

pre: TransactionHistory table contains values

post: TransactionHistory object, or array of TransactionHistory objects is returned

context transactionHistory::insertTransactionHistory() : boolean

pre: TransactionHistory object contains valid values

post: TransactionHistory object is stored in the database

context transactionHistory::getTransactionHistory() : TransactionHistory

pre: Valid array is passed

post: TransactionHistory object is returned from array values

context users::fetchUser() : User

pre: User table contains values

post: User object, or array of User objects is returned

context users::insertUser() : boolean

pre: User object contains valid values

post: User object is stored in the database

context users::updateUser() : boolean

pre: User exists in Users table and User object contains valid values

post: User records updated in the database

context users::getUsers() : User

pre: Valid array is passed

post: User object is returned from array values

context userType::fetchUserType() : UserType

pre: UserType table contains values

post: UserType object, or array of UserType objects is returned

10 System Architecture

10.1 Identifying Subsystems

On a high-level layer, our platform exists with a front-end system and a back-end system. But on a much deeper level, we can see that each of these subsystems can be broken down into still greater detail. Front end systems typically involve user interface, and object interactions with the user. Back-end will refer to all database schema, implementation and interactions with relevant hardware. Also included are non-associative items which are necessary to the success of our system.

The front-end system will have to maintain constant communication with the back-end system to maintain consistency and retrieve data regularly. It needs to be able to successfully communicate information from commands given by the user and communicate them to the back end. The back end system will retrieve necessary data and information and return the data to the UI and user to project the page or information requested.

Our back end system will be broken down further and is easily considered the most important part of our infrastructure. The back end system is to be broken down into controller and database subsystems. Additionally, we will have the price retrieval system and commenting system as previously outlined. Thus, the bulk of the command processing is handled by our back-end subsystem. The back-end system must not only communicate among the subsystems within itself, but it must also communicate with the front-end UI system to respond to commands and also communicate with the non-associate systems as well.

Breaking down the subsystem further, we highlight the importance of the price retrieval system. The financial retrieval system will communicate with commodities-api.com to retrieve relevant information as requested by the controller (whenever the controller receives input from the front-end user). The success of these modules, the success of the entire back-end system, and the success of communication amongst the systems will be crucial for the overall success of the software.

10.2 Architecture Styles

To efficiently use our software, we must couple several software tools and principles into our design. The following architecture types will be expanded in detail below to reflect functionality in both the general sense as well as the functionality of the system as a whole. The systems that we include will be (and perhaps expand upon in the future) Client-Server access, Data-Centric Design, and RESTful design. Each of these architecture styles will be serving a part of the whole system.

Client-Server Access

All interactions are occurring on a client-server basis with our program as the client is constantly interacting with the interface. The user is and always will be the primary client and because of this must always be able to interact with the other subsystems. AgriCom Training must be completely accessible to the user. The infrastructure must be able to communicate as well with API's for stock information.

Data-Centric Design

Data is absolutely essential in the functionality of AgriCom Training, without it the system would not work as intended at all. The database will store bouts of data that is necessary for all aspects of the program. It will need to store important information from both user specific data as well as information from API's so that stocks can be regularly updated to reflect proper information when placing orders. Each time the user logs in, the system will have to have stored a host of personal data including but not limited to the user's portfolio, account holdings, settings, and history of transactions.

Representational State Transfer

When using a Client-Server Access system, a REST system is inherently implied. The RESTful design principles state that in addition to having a Client-Server Access system, the system is uniform, stateless, and cacheable as well as having a scalability of components. By using this interface it allows both the users as well as the designers to have streamline interactions with the interface. Each time the user is going to do an action on the webpage it is quite clear what is going to happen. If we were to want to make this compatible with mobile browsers the RESTful implementation would also help streamline the process.

10.3 Mapping Subsystems to Hardware

This system will be fairly lightweight with access to the application interface being provided through the client's web browser. The manager and trainee users can use their web browser loaded onto their PC. The backend services such as the Account Controller, Order System Controller, Page Renderer, API Handler, Notifier, and Database will be handled by the server. The server side subsystems will handle the inputs from the application interface and display the validated outputs after running through the logic programming while communicating with the database and API feed.

10.4 Network Protocol

In accordance with standards on a typical web-based application, AgriCom Training will use the standard Hypertext Transfer Protocol (HTTP). HTTP structures text which is used in hyperlinks to communicate messages between nodes. It is not particularly unique however it is still the primary protocol in how the user and software communicate between each other. From any browser medium, a user is able to access multiple links and web pages throughout the AgriCom Training website. Most importantly, they will be able to access through this protocol all relevant stock information, portfolio history, and other relevant information throughout the website.

10.5 Global Control Flow

Execution Order

For the most part, the AgriCom Training System is an event-driven system. The elements and features of the system are largely triggered by the user or by an embedded portion of the system, with the users being in the majority for triggering the events. Viewing portfolio, history, adding comments, etc. are some of the features that are only available for the user to trigger. Alternatively, some of the triggering embedded portions of the system are the behind the scenes processes that run on certain events. A user submitting a buy or sell order will also initiate the system to call the Commodity API to retrieve current market values.

A few functionalities have an established linear order in which the process requires a specific order of events. These functionalities are:

- ❖ Registering in the system: All users must be registered before they can attempt to use any of the systems' functionality.
- ❖ Adding Trainees to manage: All Managers must add Trainees to manage before they can review Trainee accounts, leave comments, or leave feedback.

Time Dependency

The AgriCom Training System is largely a real-time system for most of the functionalities, however, there are some features that are event-driven and not dependent on time as well. There are system features that are dependent on the commodity trading hours which are limited to specific time intervals depending on numerous factors such as time of day, day of the week, commodity trading, national holidays, etc.

- ❖ Commodity trading hours: The commodity market has specific time intervals for when the market is open and closed.

- ❖ TaskTimer: Fetches market information at regular periodic intervals to store into the database.
- ❖ Trainee training regiment: An monthly periodic interval set for Trainee accounts to know when to generate the Trainee performance report and allow for Manager review and feedback.

10.6 Hardware Requirements

- ❖ Disk storage – minimum of 10 GB of available space on hard disk should be more than enough to handle the low requirements of the software package on the desktop
- ❖ Display - minimum resolution of 1280 x 800 color monitor
- ❖ Memory – minimum 2GB of RAM
- ❖ Network – minimum of 100 mbps bandwidth to ensure adequate response times for the real-time market data
- ❖ Operating System – minimum Windows 10 32-bit with latest updates to ensure compatibility
- ❖ Peripherals – keyboard, mouse (or compatible pointing device), and either a wireless or wired network card
- ❖ Processor – minimum CPU with 1 GHz frequency

11 Algorithms and Data Structures

11.1 Algorithms

Most of the functions of AgriCom take user inputs and return outputs with minimal data manipulation other than page rendering. Commodity prices will be obtained through Commodity API Handler and each request will obtain a historic range of data to be able to provide enough information for displaying a graph. A relational database will be used for persistent data storage. Most of the data in the system will be entered into the database and so algorithms for manipulating the data, such as sorting and searching, are handled by the database. Thus search and sorting algorithms are not in the scope of the system and will not be discussed.

11.2 Data Structures

The AgriCom system will utilize various data structure elements to assist in storing and processing system data for different scenarios. These planned data structures are Array, Queue, and Entity Classes based on the database table structures.

Array

Array data structures will be utilized to hold and process data that can contain more than one element for a given item. An example of this usage would be for calculating averages from transaction history. For this example there will be an unknown amount of transactions for a specific user and commodity, so an Array will be used to pull this type of data and as this structure does not have a fixed size and can grow to accommodate the data needed.

Queue

Queue data structure will be used as the foundation for the AgriCom order system. For all order tickets, buy or sell, once placed by the user the order ticket will be placed into a queue for processing based on a FIFO (First-In-First-Out) principle. Since there may be multiple users placing orders during the same time domain, this will enable the system to follow closely with trading in real world conditions.

Entity Class

Entity Classes will be created to represent each table in the SQL relational database structure given that the tables' data will need to be referenced by the system. Tables such as User, Portfolio, Commodity, TransactionHistory, etc. will have entity classes created so the various columns and data elements can easily be pulled from the database and referenced in the application layer. Each entity will contain all of the columns and their corresponding data types, as well as getter and setter methods to access the data within each entity class.

12 User Interface Design and Implementation

12.1 Updated Design

Most of the user interface is completed at this point and small changes will be made to make the website more user friendly and visually appealing. Our mockups from the previous reports and found in section 5 are what we have been basing our design off of and so far it has been going very well.

A few changes that we do plan on making still include: adding an option on the login page to recover a lost password and implementing the accounts page fully. If we were to not include a way to recover a password it would be very difficult for anyone to recover their account and they would unfortunately have to make a new one in order to keep using the training.

12.2 Efficiency

A very important concern is making sure that the website is easy to use and fast for all users who attempt to use our website. For fast loading times we can and will logically break down our website into different elements so that the load time is more efficient especially for those elements that are unchanging and can be cached. For example, we can separate the header and the content of a given page. Since the header is the same across all pages, they only need to be loaded a single time to the client and can be cached on the client side for the duration of the visit to the website.

To be able to show that a user is interacting with the webpage without reloading it, we can take advantage of technologies such as Comet [5]. This will allow us to keep the pages dynamic while reducing load times by caching more information on the client side and perform updates only when needed with lower delay.

In order to reduce client load we will be using HTML[6] and CSS[7] to display our user interface while minimally using pictures if any. Any picture displayed will be contained in appropriate web format and resized to fit the container as needed. Our main audience is those training in the stock market so the user interface is made to be used in browsers. While we will make the pages responsive to browser size it will be optimized for those with a laptop or desktop computer. This also means that it is expected to have a browser web compliant with modern web standards. This should have minimal impact however as most people in our target audience will have those requirements met.

13 Design of Tests

13.1 Test Cases

Test-case Identifier: TC-1 Function Tested: Buy Commodity, SystemController::RequestBuy(Ticket : ticket) : Void Pass/Fail Criteria: The test passes if the request to buy commodity is sent to the DatabaseConnection. The test fails if the request does not go out, due to an incorrect argument.	
Test Procedure	Expected Results
-Call Function (Success)	-Correct data gets sent, correct commodity is added to the investor's portfolio
-Call Function (Fail)	-Data does not get sent, no new commodity was purchased

Test-case Identifier: TC-2 Function Tested: Sell Commodity, SystemController::RequestSell(Ticket : ticket) : Void Pass/Fail Criteria: The test passes if the request to sell stock is sent to the DatabaseConnection. The test fails if the request does not go out, due to an incorrect argument.	
Test Procedure	Expected Results
-Call Function (Success)	-Correct data gets sent, correct commodity is removed from the investor's portfolio
-Call Function (Fail)	-Data does not get sent, no commodity was sold

Test-case Identifier: TC-3 Function Tested: Register as Trainee, AccountController::CreateAccount(UserInfo: userinfo): Boolean Pass/Fail Criteria: The test passes if the test stub requests an account creation and the request is granted.	
Test Procedure	Expected Results
-Request to create an account (Pass)	-AccountController requests account creation and returns true if account creation successful
-Call Function (Fail)	-If the request for account creation is unsuccessful, return false

Test-case Identifier: TC-4 Function Tested: View Portfolio, UserController::RequestPortfolio(String: Investor): Portfolio Pass/Fail Criteria: The test passes if the test stub requests for portfolio data and it is retrieved from the database	
Test Procedure	Expected Results
-Request portfolio data (Pass)	-UserController requests portfolio data and returns it from the database.
-Call Function (Fail)	-If there is an error retrieving the data from the database, it should display an error that no pertinent data was returned.

Test-case Identifier: TC-5 Function Tested: View Transaction History, SystemController::RequestHistory(String : investor) : Void Pass/Fail Criteria: The test passes if the request for the correct investor history is sent to the DataHandler. The test fails if the request does not go out, due to an incorrect argument.	
Test Procedure	Expected Results
-Call Function (Success)	-Correct data gets sent, RequestHistory of SystemController is called
-Call Function (Fail)	-Data does not get sent, RequestHistory of SystemController is not called

Test-case Identifier: TC-6 Function Tested: Submit Comment, SystemController::SubmitComment(TransactionID: transactionID, String: comment): Boolean Pass/Fail Criteria: The test passes if the comment is saved to the transaction in the database. The test fails if the request is not saved.	
Test Procedure	Expected Results
-Call Function (Success)	-Correct data gets sent, Database returns true.
-Call Function (Fail)	-Data does not get sent, Database returns nothing or false.

Test-case Identifier: TC-7 Function Tested: View Comment, SystemController::RequestComment(TransactionID: transactionID) : Void Pass/Fail Criteria: The test passes if the request for the correct comments under a transaction is sent to the DataHandler. The test fails if the request does not go out, due to an incorrect argument.	
Test Procedure	Expected Results
-Call Function (Success)	-Correct data gets sent, RequestComment of SystemController is called
-Call Function (Fail)	-Data does not get sent, RequestComment of SystemController is not called

Test-case Identifier: TC-8 Function Tested: View Educational Information, SystemController::RequestEducation() : Void Pass/Fail Criteria: The test passes if the request is called. The test fails if the request does not go out.	
Test Procedure	Expected Results
-Call Function (Success)	-RequestEducation of SystemController is called
-Call Function (Fail)	-RequestEducation of SystemController is not called

Test-case Identifier: TC-9 Function Tested: View View Commodity, CommodityAPI::Query(String: commodity):CommodityData Pass/Fail Criteria: The test passes if the system queries a commodity and that commodity is returned	
Test Procedure	Expected Results
-Request to query a commodity (Pass)	-CommodityQuery returns the commodity data
-Request to query a commodity (Fail)	-If the attempted query was for a commodity that does not exist, CommodityQuery should return an error code that the commodity does not exist. If commodity information was not attainable, it should display an error that no pertinent data was returned.

Test-case Identifier: TC-10 Function Tested: Register as Manager, AccountController::CreateAccount(UserInfo: userinfo): Boolean Pass/Fail Criteria: The test passes if the test stub requests an account creation and the request is granted.	
Test Procedure	Expected Results
-Request to create an account (Pass)	-AccountController requests account creation and returns true if account creation successful
-Call Function (Fail)	-If the request for account creation is unsuccessful, return false

Test-case Identifier: TC-11 Function Tested: Add Trainee to manage, AccountController::Manage(String: traineeID): Void Pass/Fail Criteria: The test passes if the test stub requests that a Trainee account be added to the Manager's account.	
Test Procedure	Expected Results
-Add Trainee account to the Manager's account. (Pass)	-AccountController adds Trainee account to the Manager's account
-Add Trainee account to the Manager's account. (Fail)	-If unable to add Trainee in database, display error saying that the operation was unsuccessful

Test-case Identifier: TC-12 Function Tested: Remove Trainee to manage, AccountController::UnManage(String: traineeID): Void Pass/Fail Criteria: The test passes if the test stub requests that a Trainee account be removed from the Manager's account.	
Test Procedure	Expected Results
-Remove the Trainee account from the Manager's account. (Pass)	-AccountController removes Trainee account from the Manager's account
-Remove the Trainee account from the Manager's account. (Fail)	-If unable to remove Trainee in database, display error saying that the operation was unsuccessful

Test-case Identifier: TC-13 Function Tested: Reset Trainee's account, AccountController::ResetPortfolio(String: traineeID): Boolean Pass/Fail Criteria: The test passes if the test stub requests to reset a trainee's portfolio settings in the database and is successful. Unsuccessful if settings not changed	
Test Procedure	Expected Results
-Request to reset a Trainee's portfolio from the Manager's account. (Pass)	-AccountController modifies Trainee's portfolio settings and returns true
-Request to reset a Trainee's portfolio from the Manager's account. (Fail)	-AccountController unable to modify portfolio settings, returns false.

Test-case Identifier: TC-14 Function Tested: Provide end of training feedback, AccountController::EndofTraining(String: traineeID): Void Pass/Fail Criteria: The test passes if the test stub calls EndofTraining when the Trainee's training regiment expires.	
Test Procedure	Expected Results
-TimerTask calls function. (Pass)	-Checks with Database to find accounts that is expired and calls EndofTraining function
-TimerTask calls function. (Fail)	-EndofTraining function fails to be called despite having accounts expiring in database

13.2 Test Coverage

The ideal test coverage would be to have a test that covers every edge case of every method. This is not only not feasible, it is impossible since it is not possible to actually know all the edge cases. Because of this we plan to test core functionality to provide a core amount of testing. Through the use of alpha and beta build interactions with end users, we will be able to identify ways that users may interact with the system that were not foreseen.

We can then add additional testing to cover these new edge and use cases which will also help debug and prevent regression in the future.

13.3 Integration Testing

Integration testing will be done on a local developer machine by emulating the server environment. The system may not go live until the current system works in the integration environment.

We accomplish this by having two branches of source code on GitHub, master and dev. dev is the branch that all new work will be done on. From there, it will be pulled down into the local integration machine to be tested and debugged. Once the system has been debugged with detailed logs of any system config changes, the source code will be pushed to master.

Once the source code is pushed to master, any system config changes will be made on the production server in order to accommodate the new branch. Once those changes are made, master will be pulled into the production machine and a second round of integration testing will begin by launching the service on a developer port. If it passes all the tests, then the developer port will be shut down, and the system will relaunch the website on the normal http port.

14 History of Work, Current Status, and Future Work

14.1 History of Work

Project was determined to be successful with the starting completion of a team that could coordinate and distribute workload to mitigate deadlines through stretch goals that could be facilitated through coordinated endeavors to maintain contact and peruse through weak points to dictate where responsive action would be necessary. Weekly meetings were held to hold each other accountable and to maintain integrity with workload with updates by Discord with discussion on problems and issues to be managed out by the team.

Every completion of deadlines was done as a team and similar protocols were in place when starting the next deadline to allow each member to determine what the next goal is to be while allowing others the opportunity to work on strengths and reign in on weaknesses. Teammates were able to develop roles and responsibilities based on skill sets and scheduling. Guidelines were able to devise strategies in place and questions were asked to maintain a high level of quality control throughout the semester. Work was of high quality and continues to be in good standing with completion of deadlines with coding assignments to be nearing its fruition to complete and completion of the semester.

14.2 Current Status

Facilitating the work nearing its completion was a milestone that was accomplished by communication and appropriate time management. As of now, completion of report #3 is another milestone that will be followed up with comments, questions, and concerns to dictate next course of action and to allow members of Group B to settle into the next chapter of the project and to ensure that members are able to maintain scheduling despite setbacks and conflicts due to obligations outlined in our response action plans.

With Demo #2 in the works and the completion of report #3 in the making at the time of writing the report, the current insinuation of our ability to interact with and work towards a common goal is sustainable to allow the completion of the remainder of the assignments to be continued as expected.

14.3 Future Work

With the semester ending, the future will involve the completion of Demo #3 as mentioned previously in the discussion and to follow up with the video presentation while submitting reflective essays on our respective contributions towards the project. Reception towards the ordeal has been positive as an After-Action Review is necessary to see where our strengths and weaknesses lie as a team and towards our individual contributions that can be used towards the completion of each program to use and gain valuable insights into where we shine and where we can improve.

15 Report 3 Contributions

All group members contributed equally to this report.

	Names				
Category	David Gladden	David Schiffer	Alexis Angel	Calvin Ku	Christopher Katz
Report 3	20%	20%	20%	20%	20%

References

- ❖ *CommoPrices API*. API CommoPrices. (n.d.). Retrieved February 16, 2023, from <https://api.commoprices.com/>
- ❖ | *commodities prices and currency conversion JSON API*. Commodities-API. (n.d.). Retrieved February 16, 2023, from <https://commodities-api.com/>
- ❖ Lioudis, N. (2022, December 19). *Commodities trading: An overview*. Investopedia. Retrieved February 22, 2023, from <https://www.investopedia.com/investing/commodities-trading-overview/>
- ❖ Palmer, B. (2023, February 3). *A beginner's Guide to Precious Metals*. Investopedia. Retrieved February 22, 2023, from <https://www.investopedia.com/articles/basics/09/precious-metals-gold-silver-platinum.ap>
- ❖ Wikimedia Foundation. (2022, August 15). *Comet (programming)*. Wikipedia. Retrieved March 24, 2023, from [https://en.wikipedia.org/wiki/Comet_\(programming\)](https://en.wikipedia.org/wiki/Comet_(programming))
- ❖ Wikimedia Foundation. (2023, March 23). *HTML*. Wikipedia. Retrieved March 24, 2023, from <https://en.wikipedia.org/wiki/HTML>
- ❖ Wikimedia Foundation. (2023, March 23). *CSS*. Wikipedia. Retrieved March 24, 2023, from <https://en.wikipedia.org/wiki/CSS#>