Team

1. Dante Pasionek → Dpasi314
2. Preston Kelley → pkelley
3. Walker Schmidt → walkyd12

Project

PaperTrader - An alternative to traditional stock market simulators

Project Summary

A python based stock market simulator, web application that allows users to trade in real time with fake currency. Traders are able track their investments and find optimal trading strategies.

Index

User Requirements

While we believe that buying and selling are the two most fundamental concepts of this project, we will include analysis of other concepts of similar importance.

USER REQUIREMENTS		Priority
UCR-001	As an **Administrator** I want to be able to selectively set up which stocks are available to investors to provide a small sample of real-world investing scenarios.	Highest
UCR-002	As a **Trader** I want to be able to view all of my investment history from the current point in time to my first transaction to be able to track my total monetary investments over time.	Normal
UCR-003	As a **Trader** I want to be able to buy available stock in a quantity that I specify	Highest
UCR-004	As a **Trader** I want to be able to sell available stock in a quantity that I specity so I'm able to divest in assets to increase my financial capital.	Highest
UCR-005	As a **Trader** I want to be able to add money to my account so that I'm able to buy stock, similar to an investment I would make in real life.	Normal
UCR-006	As an **Administrator** I want to be able to search for a Trader by name to view their current information.	Low
UCR-007	As a **Trader** I want to be able to view my personal portfolio in order to access information regarding my investments.	Normal
UCR-008	As a **Trader** I want a trading algorithm to help determine which stocks I should invest in based on their current trends to help me make better real-life decisions.	High

NON-**F**UNCTIONAL REQUIREMENTS		Priority
UCNF-001	**Performance:** The site should have at least 90% uptime during active trading hours to help maintain accurate results.	Normal

FUNCTIONAL REQUIREMENTS		Priority
UCF-001	The **System** will prevent Traders from purchasing stock when their balance amount is less than the price of the amount of stock attempting to be purchased.	Normal
UCF-002	The **System** will prevent Traders from selling stock when the quantity they wish to sell is greater than the total quantity in their portfolio	Normal

BUSINESS REQUIREMENTS		Priority
UCB-001	A **User** can be identified by a name and separate static user identification number.	High

Use cases shaded gray indicate our highest priority use cases.

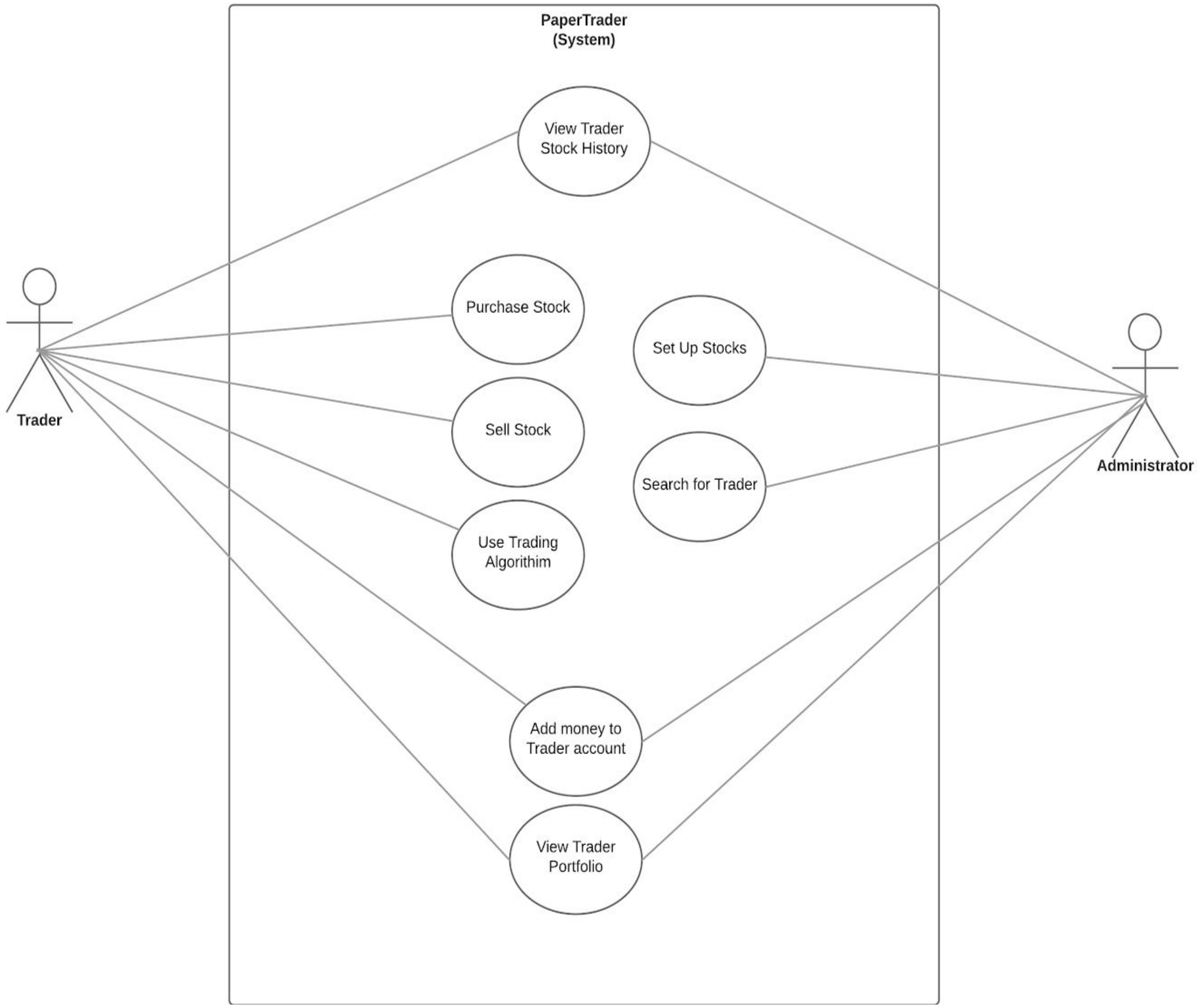
Priority → Highest, High, Normal, Low, Lowest

Use Case Documents & Diagram

Use Case Diagram - Single diagram depicting all use cases

PAPERTRADER | USE CASE DIAGRAM

Dante Pasionek, Preston Kelley, Walker Schmidt



Use Case Documents

Use Case ID:	UCR-003
Use Case Name:	Traders can purchase stock
Description:	Traders are able to select which stocks they'd like to invest their money in

Actors:	Trader
Pre-Conditions:	Trader is logged in and viewing the stock page
Post-Conditions:	Trader's money is invested into specific stock, their investment will now be tracked.
Frequency of Use:	Sporadic

Flow of Events:		**Actor Action**	**System Response**
	1	Click Stock name	Display verbose stock information
	2	Enter amount to invest	
	3	Click Add	Get existing stock object
	4		Compare user balance to total amount of stock objects, purchasing if there is enough money
	5		Remove money from user's account
	6		Add stock to user's portfolio
	7		Add investment into user's investment history
	8		Display message
	9	Click Finish	

Variations:	
Exceptions:	**4.** User does not have enough money; Jump to **6** display error message
Notes:	

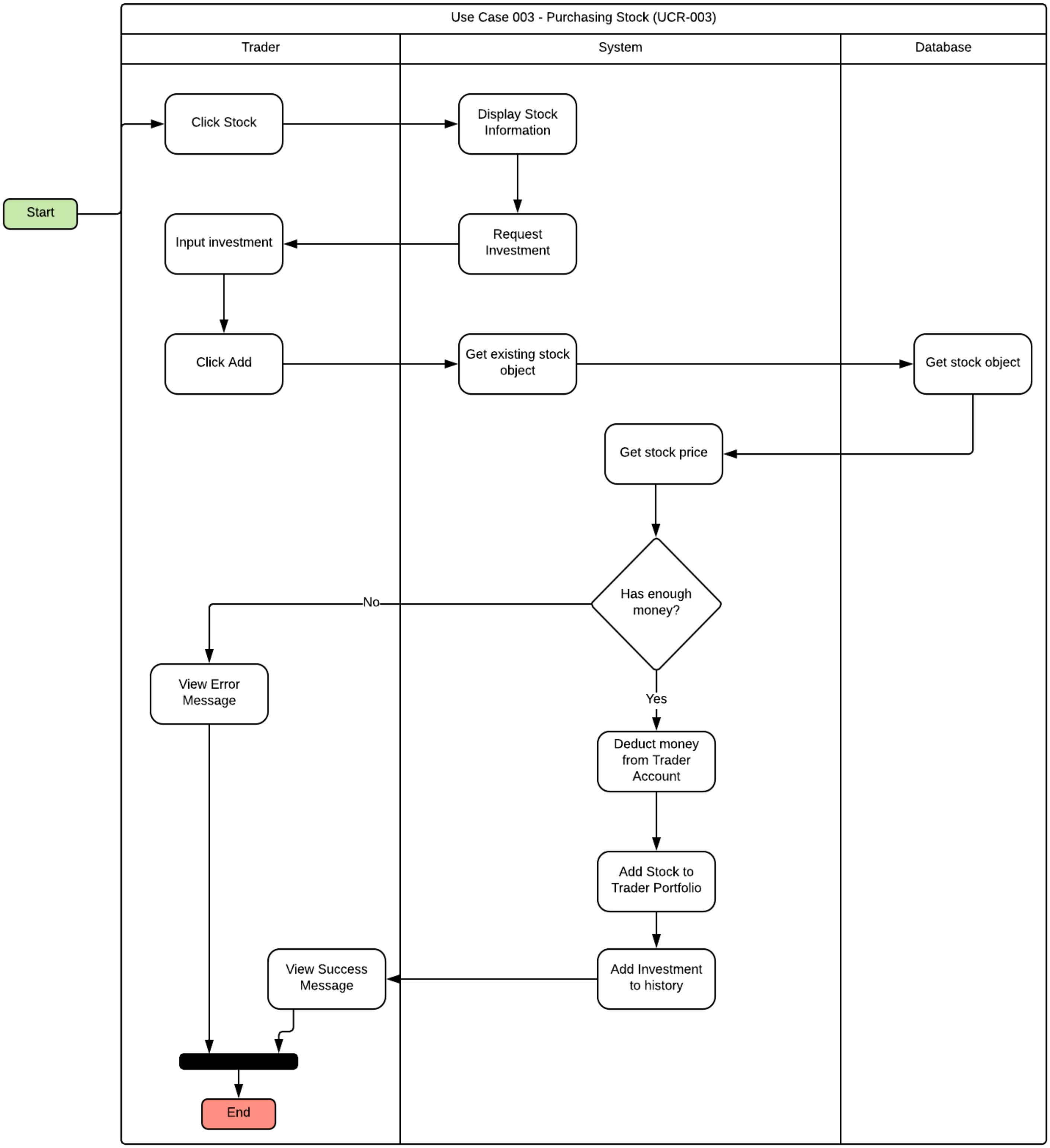
Use Case ID:	UCR-004
Use Case Name:	Traders can sell stock
Description:	Traders are able to sell stocks in which they already have money invested

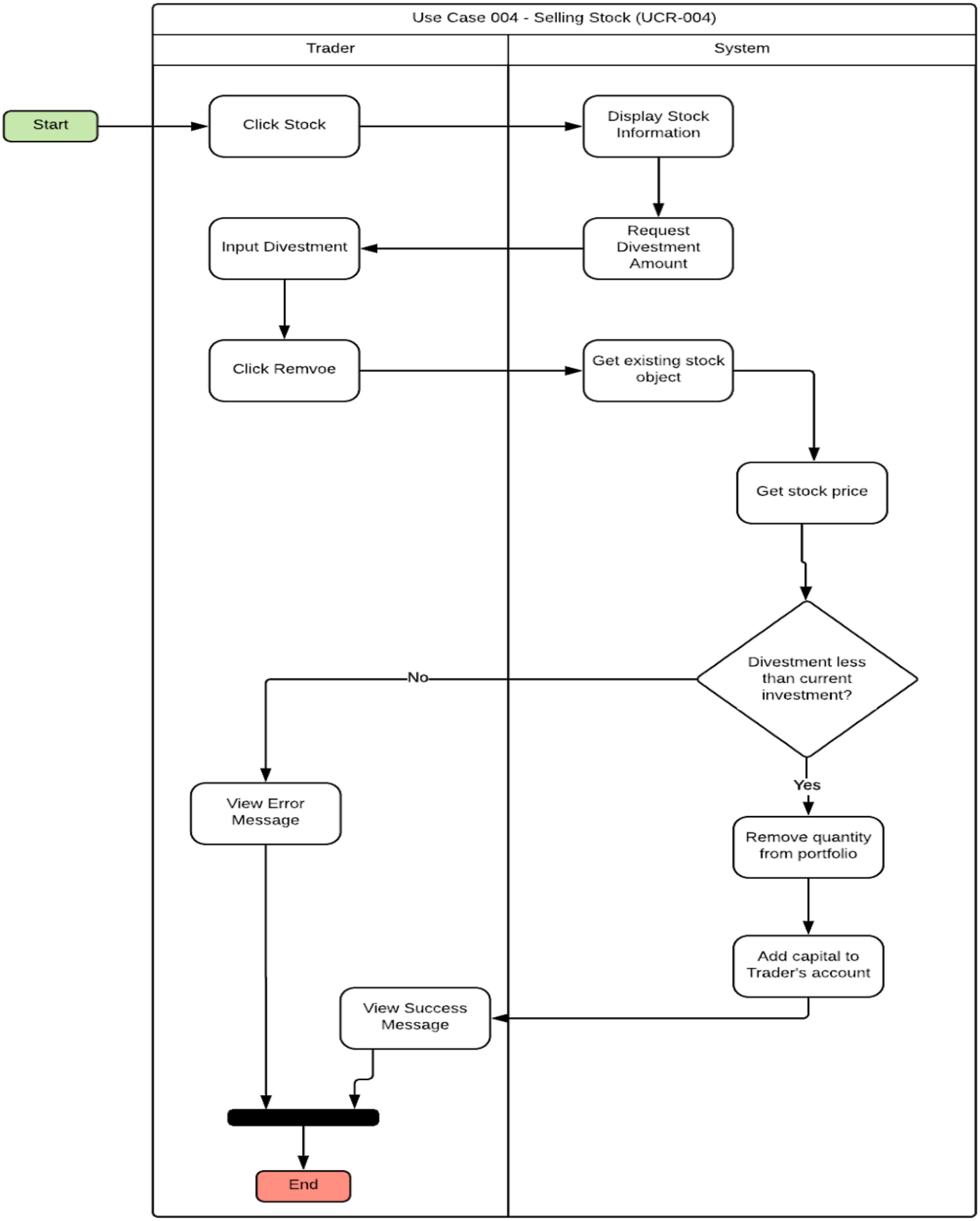
Actors:	Trader
Pre-Conditions:	Trader is logged in and viewing the stock page, trader is already invested in stock
Post-Conditions:	Trader’s investment in stock is removed
Frequency of Use:	Sporadit

Flow of Events:		**Actor Action**	**System Response**
	1	Click Stock name	Display verbose stock information
	2	Enter amount to divest	
	3	Click Finish	Get trader portfolio
	4		Check if amount entered to divest is less than or equal to current total asset amount
	5		Remove stock or quantity from user portfolio
	7		Return capital to Trader’s account
	8		Display Message

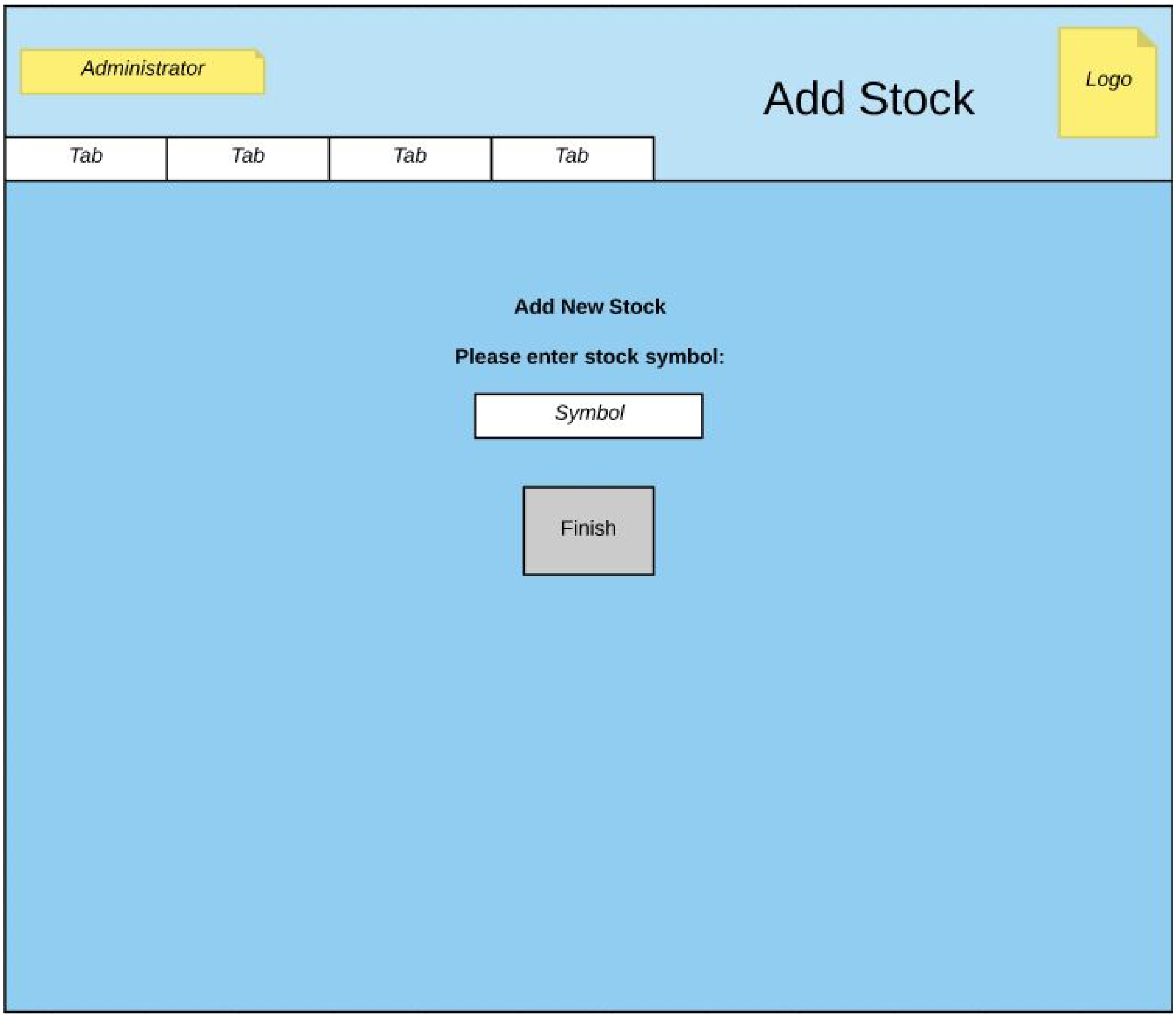
Variations:	
Exceptions:	**4.** Divestment amount is larger than the current investment amount; Jump to **6** display error message
Notes:	**Investment -** Refers to current amount of money a Trader currently has spent purchasing stock. **Divestment -** Refers to the desired amount of money returned from an investment already made.

Activity Diagrams

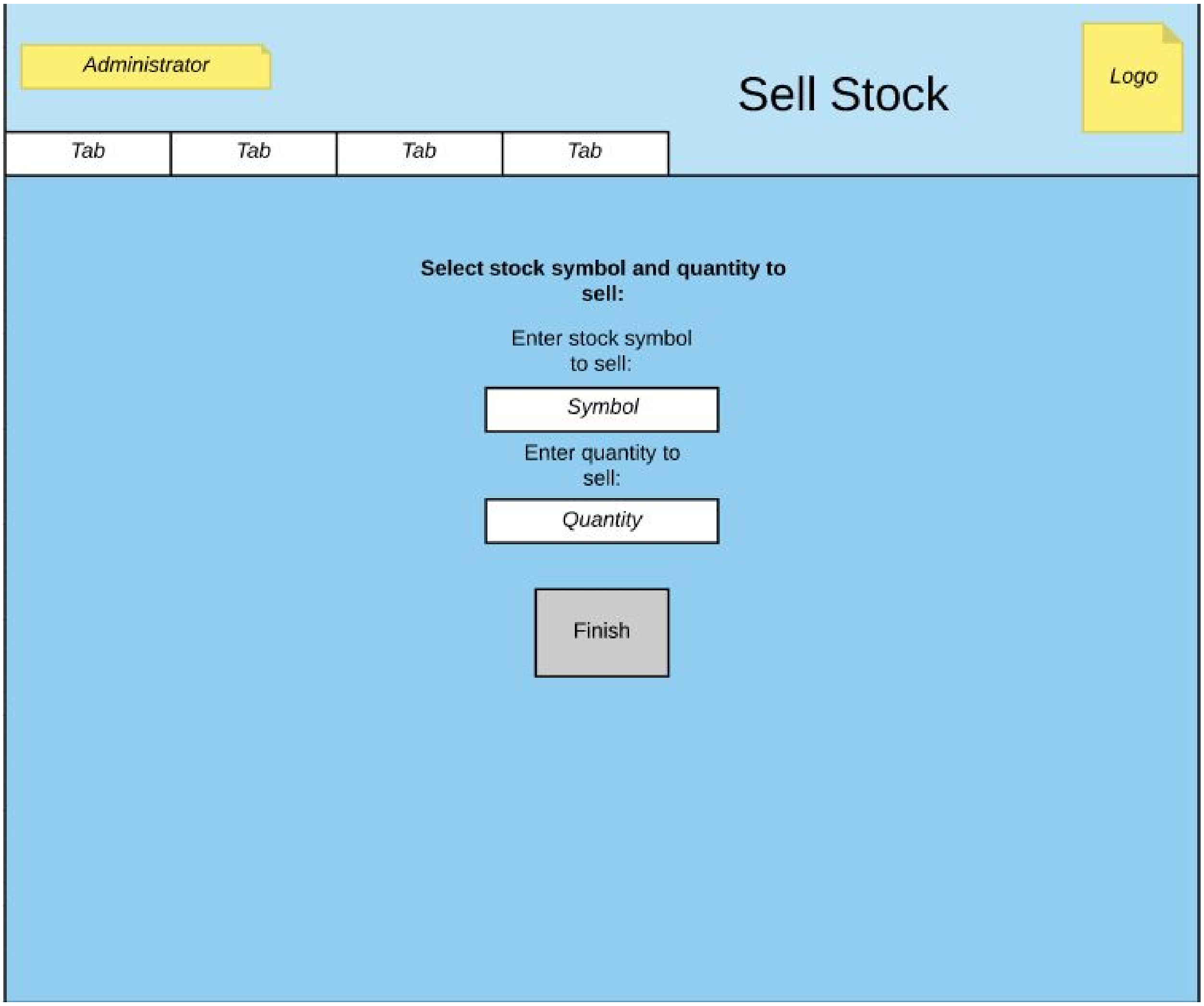

Use Case 003 - Purchasing Stock (UCR-003)
Trader
System
Database
Start
Click Stock
Display Stock Information
Input investment
Request Investment
Click Add
Get existing stock object
Get stock object
Get stock price
Has enough money?
No
Yes
View Error Message
Deduct money from Trader Account
Add Stock to Trader Portfolio
View Success Message
Add Investment to history
End


Use Case 004 - Selling Stock (UCR-004)
Trader
System
Start
Click Stock
Display Stock Information
Request Divestment Amount
Input Divestment
Click Remvoe
Get existing stock object
Get stock price
Divestment less than current investment?
No
Yes
View Error Message
Remove quantity from portfolio
Add capital to Trader's account
View Success Message
End

UI Mockups



An example of the Add Stock screen

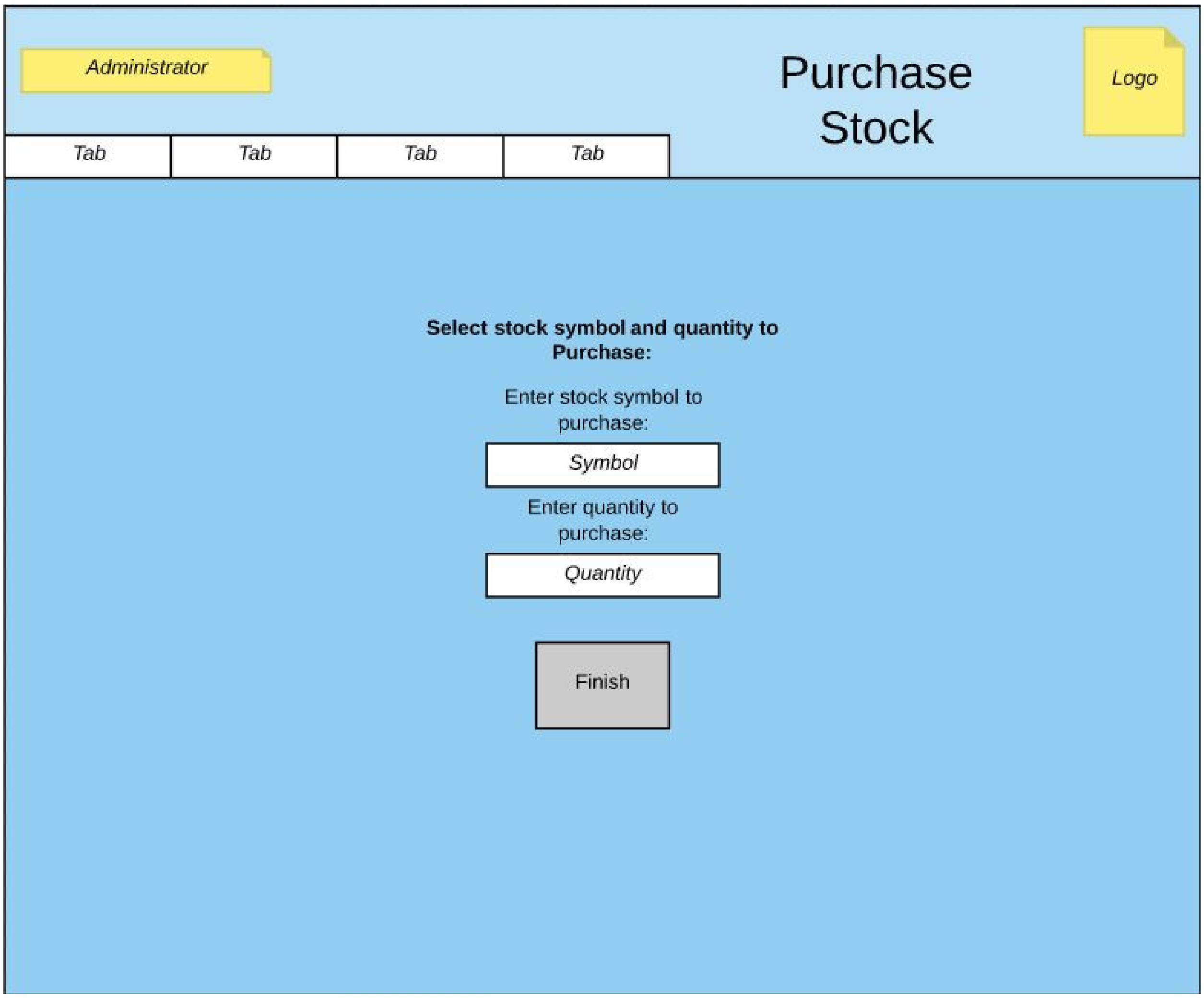


Select stock symbol and quantity to sell:

Enter stock symbol to sell:

Enter quantity to sell:

An anticipated initial version of the Sell Stock page



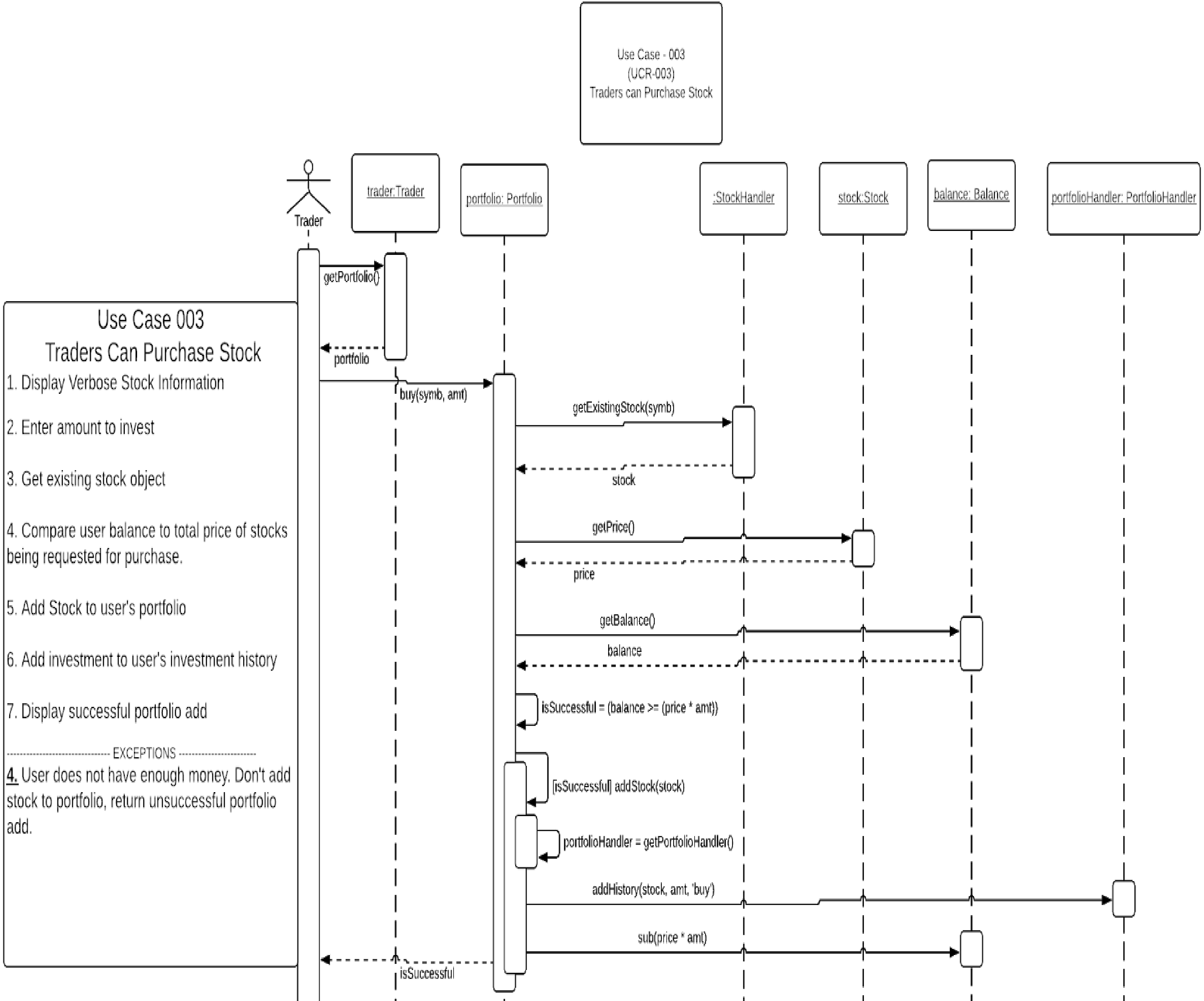
Select stock symbol and quantity to Purchase:

Enter stock symbol to purchase:

Enter quantity to purchase:

A prototype of the Purchase Stock page

User Interactions



Use Case 003
Traders Can Purchase Stock

1. Display Verbose Stock Information

2. Enter amount to invest

3. Get existing stock object

4. Compare user balance to total price of stocks being requested for purchase.

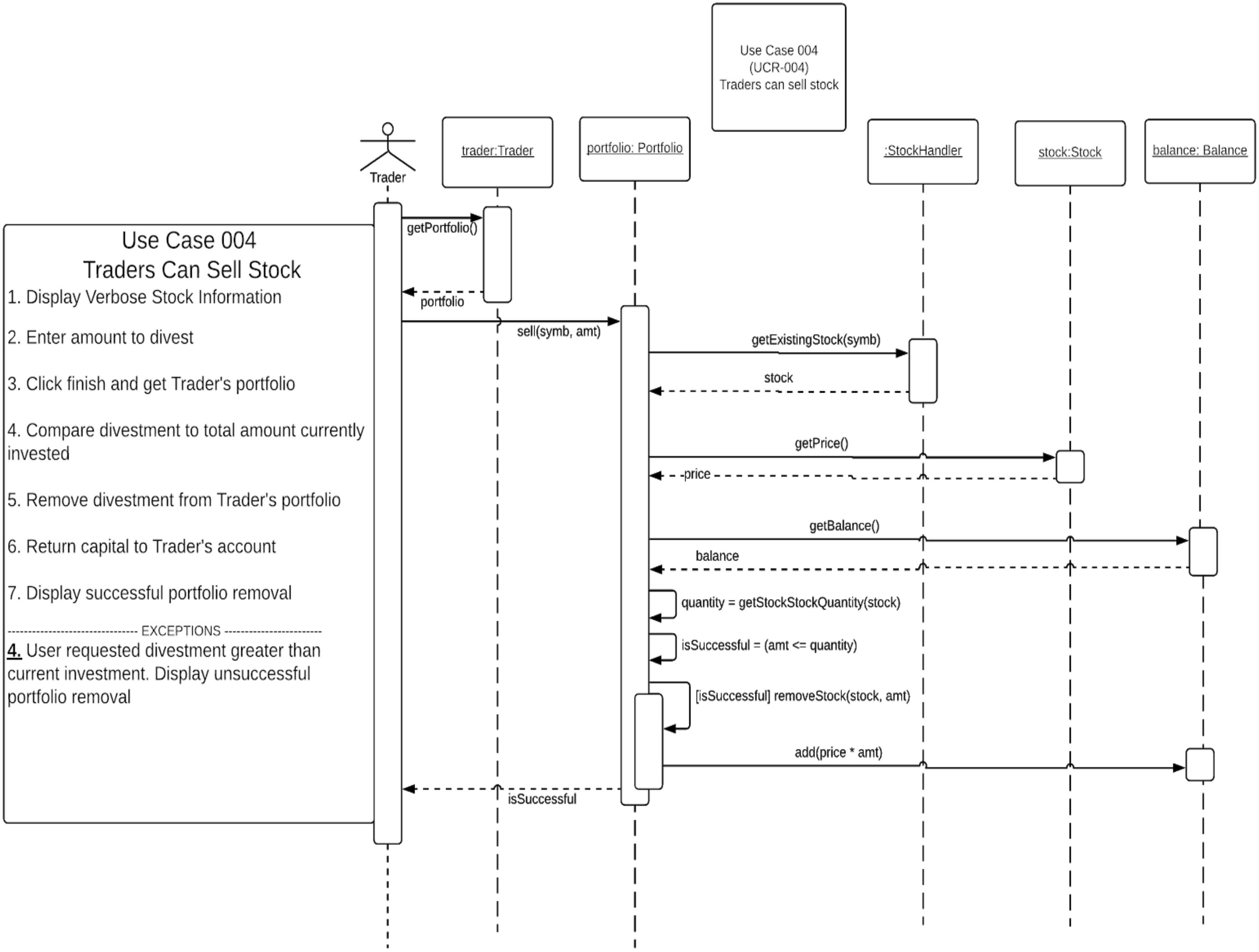
5. Add Stock to user's portfolio

6. Add investment to user's investment history

7. Display successful portfolio add

---------------------------- EXCEPTIONS ----------------------

4. User does not have enough money. Don't add stock to portfolio, return unsuccessful portfolio add.



Use Case 004
Traders Can Sell Stock

1. Display Verbose Stock Information

2. Enter amount to divest

3. Click finish and get Trader's portfolio

4. Compare divestment to total amount currently invested

5. Remove divestment from Trader's portfolio

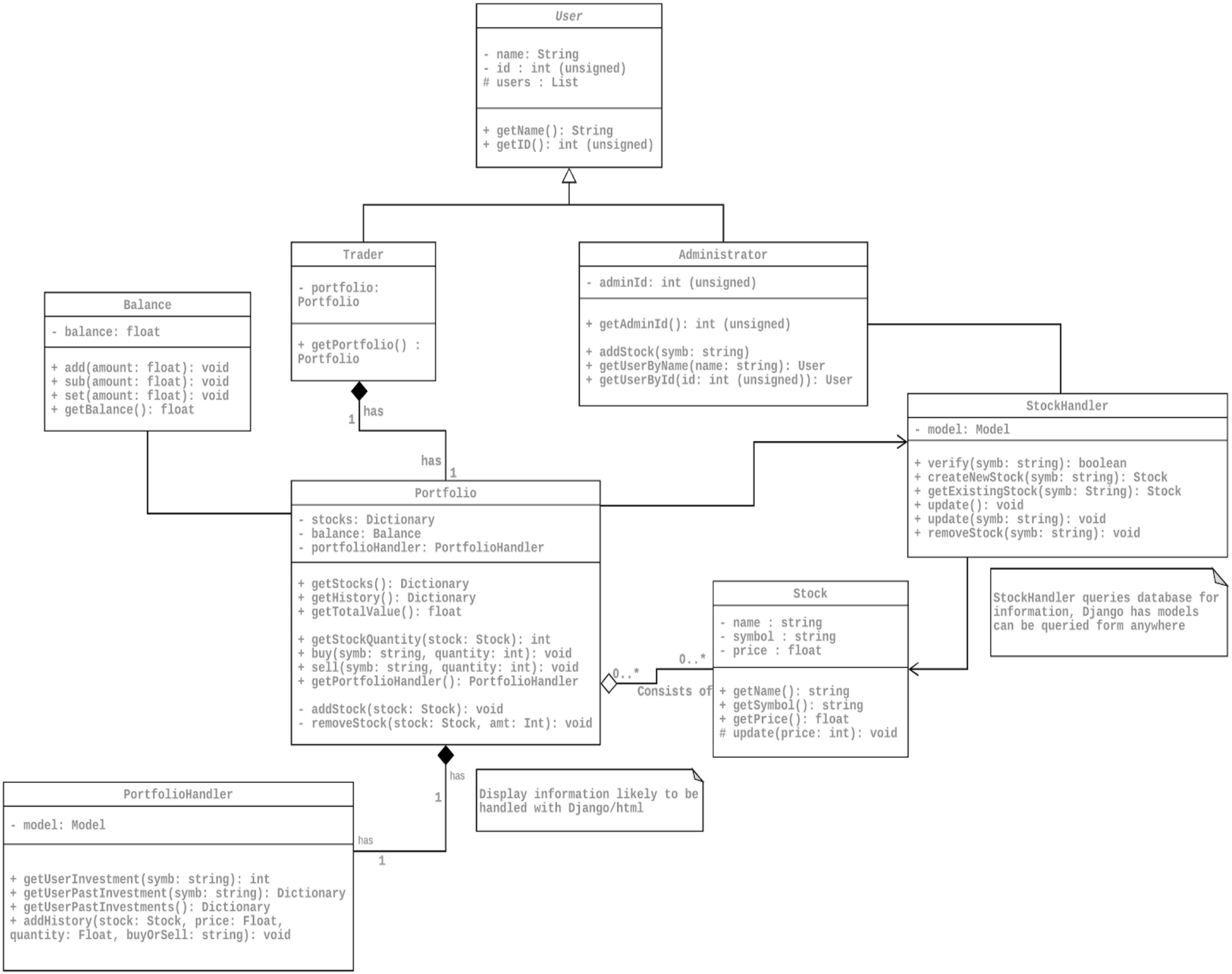
6. Return capital to Trader's account

7. Display successful portfolio removal

------------------------------ EXCEPTIONS ----------------------

4. User requested divestment greater than current investment. Display unsuccessful portfolio removal

Class Diagram


User
- name: String
- id : int (unsigned)
# users : List
+ getName(): String
+ getID(): int (unsigned)
Trader
- portfolio: Portfolio
+ getPortfolio() : Portfolio
Administrator
- adminId: int (unsigned)
+ getAdminId(): int (unsigned)
+ addStock(symb: string)
+ getUserByName(name: string): User
+ getUserById(id: int (unsigned)): User
Balance
- balance: float
+ add(amount: float): void
+ sub(amount: float): void
+ set(amount: float): void
+ getBalance(): float
has
1
has
1
Portfolio
- stocks: Dictionary
- balance: Balance
- portfolioHandler: PortfolioHandler
+ getStocks(): Dictionary
+ getHistory(): Dictionary
+ getTotalValue(): float
+ getStockQuantity(stock: Stock): int
+ buy(symb: string, quantity: int): void
+ sell(symb: string, quantity: int): void
+ getPortfolioHandler(): PortfolioHandler
- addStock(stock: Stock): void
- removeStock(stock: Stock, amt: Int): void
StockHandler
- model: Model
+ verify(symb: string): boolean
+ createNewStock(symb: string): Stock
+ getExistingStock(symb: String): Stock
+ update(): void
+ update(symb: string): void
+ removeStock(symb: string): void
StockHandler queries database for information, Django has models can be queried form anywhere
Stock
- name : string
- symbol : string
- price : float
+ getName(): string
+ getSymbol(): string
+ getPrice(): float
# update(price: int): void
0..*
0..*
Consists of
has
1
Display information likely to be handled with Django/html
PortfolioHandler
- model: Model
+ getUserInvestment(symb: string): int
+ getUserPastInvestment(symb: string): Dictionary
+ getUserPastInvestments(): Dictionary
+ addHistory(stock: Stock, price: Float, quantity: Float, buyOrSell: string): void
has
1