TEAM 3

WEB BASED STOCK FORECASTERS

https://sites.google.com/site/group3stockforecasting/

REPORT 2

Peter Zhang
Vincent Chen
Robert Adrion
Syedur Rahman
Robin Karmakar
Mohammed Latif
Manoj Velagaleti

Responsibility Matrix

	TEAM MEMBERS						
RESPONSIBILITIES	Robert Adrion	Vincent Chen	Robin Karmakar	Mohammed Latif	Syedur Rahman	Manoj Velagaleti	Peter Zhang
Interaction Diagrams	20%	20%		20%	10%	20%	10%
Diagrams descriptions	20%	20%		20%	20%	20%	
Alternate solution descriptions	20%	20%		20%	20%	20%	
Class Diagram						100%	
Data Types and Operation Signatures					50%	50%	
Architectural Styles							100%
Identifying Subsystems		50%		50%			
Mapping Subsystems to Hardware	100%						
Persistent Data Storage		50%		50%			
Network Protocol, Global Control Flow, Hardware Requirements	50%						50%
Algorithms and Data Structures	100%						
Appearance					50%		50%
Description					50%		50%
Design of Tests			100%				
Merging the Contributions from Individual Team Members	8%	7%	2%	52%	12%	8%	11%
Project Coordination and Progress Report		100%					
Plan of Work			100%				
TOTAL POINT ALLOCATION	14.38	14.27	14.22	14.22	14.32	14.38	14.21

Allocation of Points

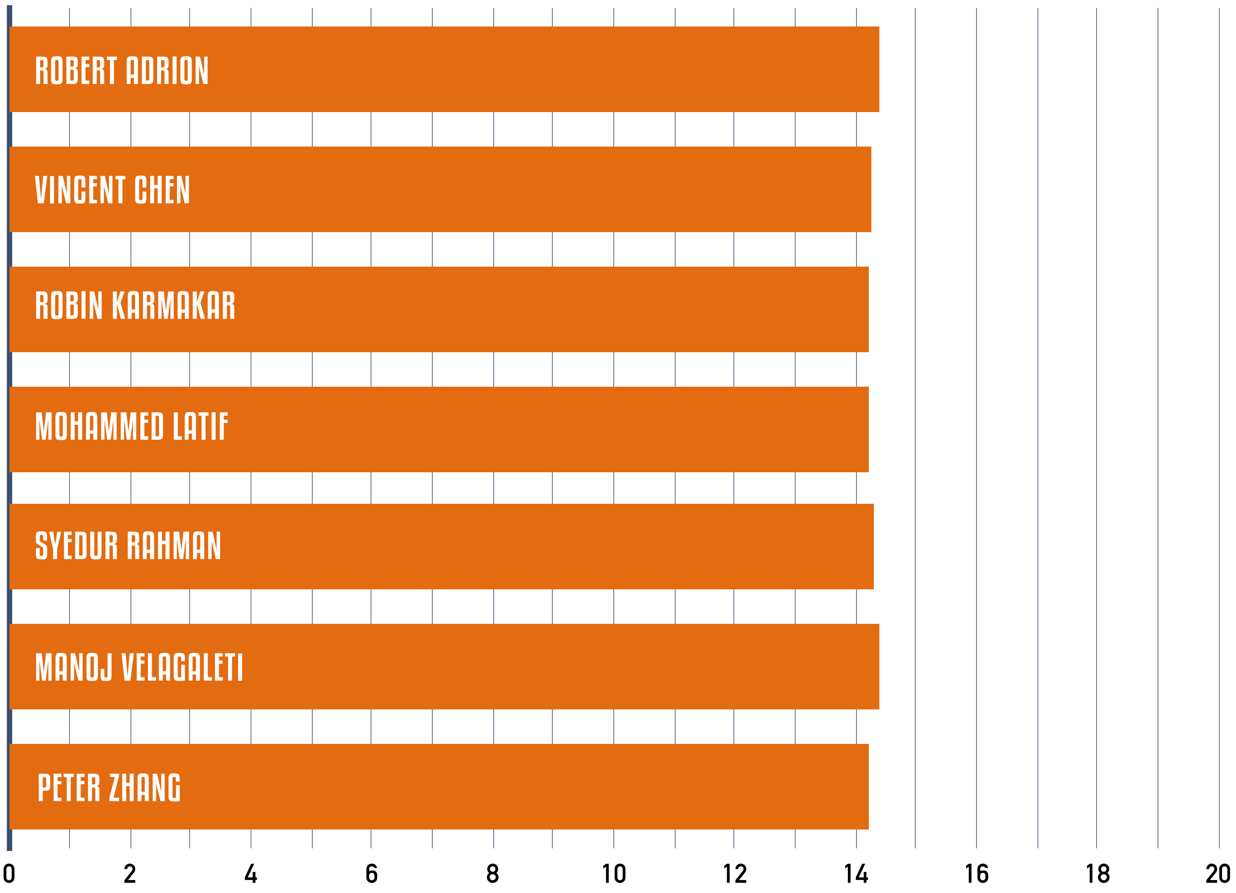


Table of Contents

1. Interaction Diagrams

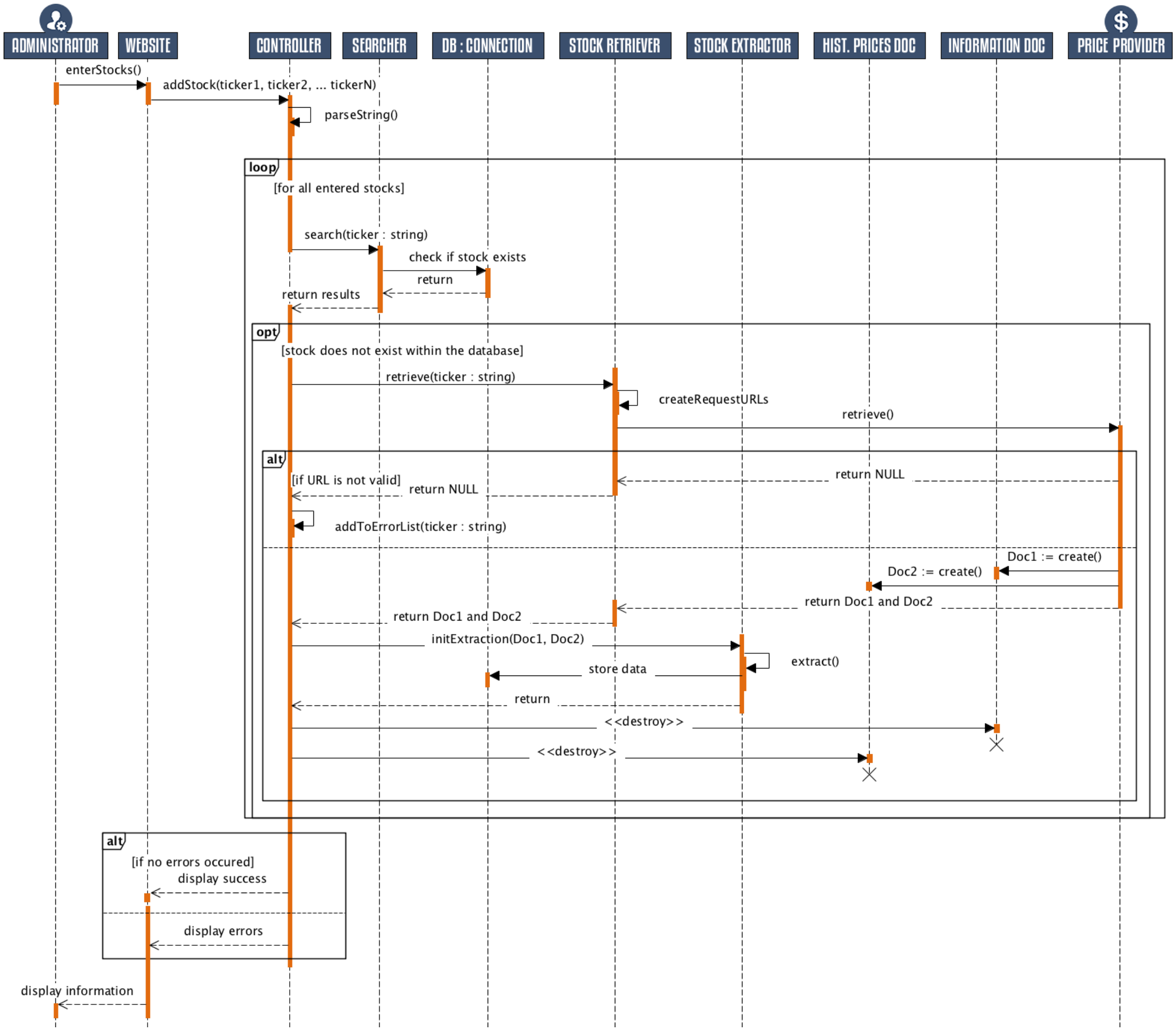

UC-2: AddStock

Figure-18: Design sequence diagram for UC-2: AddStock

Figure-18 shows the design sequence diagram for Use Case 2: AddStock. Once the administrator enters a sequence of comma separated ticker symbols the controller parses that string into an array of ticker symbols. A main loop is then entered for each ticker. To prevent duplicates, the searcher verifies that the stock does not already exist within the database. Once the stocks uniqueness is verified, two request URLs are created by the Stock Retriever to retrieve the necessary stock data. If the administrator made a typo or for some reason the URL is invalid, the failed stock is added to an error list and the loop continues onto the next stored ticker. Otherwise, the price provider will create and return two documents: a file containing historical prices and a file containing current stock information such as industry, sector, and current prices. The relevant data is then extracted from the documents and stored within the database by the Stock Extractor. Once the loop has completed, the administrator is notified of errors that occurred if any.

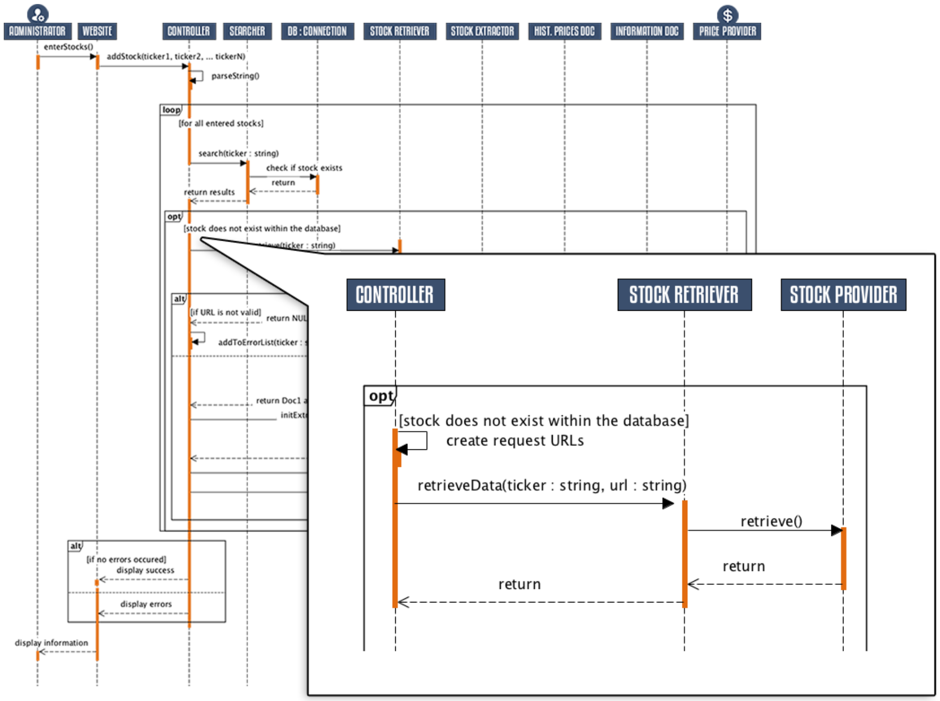


Figure-19: Alternate design sequence diagram for UC-2: AddStock

The question of who should create the request URLs can lead to many solutions. The expert doer principle suggest that this should be the controller since it first gets the knowledge of whether a stock already exists within the database or not (and thus whether a request URL must be created for it). Figure-19 demonstrates this alternate design where the controller first constructs the URLs and passes them to the stock retriever. It should be noted that to simplify this diagram, unrelated intermediate function calls have been removed. The high cohesion principle, on the other hand, would dictate that the stock retriever should be responsible for creating the request URL's. Since the controller's sole responsibility is to manage communication between concepts and nothing more, the initial design in Figure-18 is chosen.

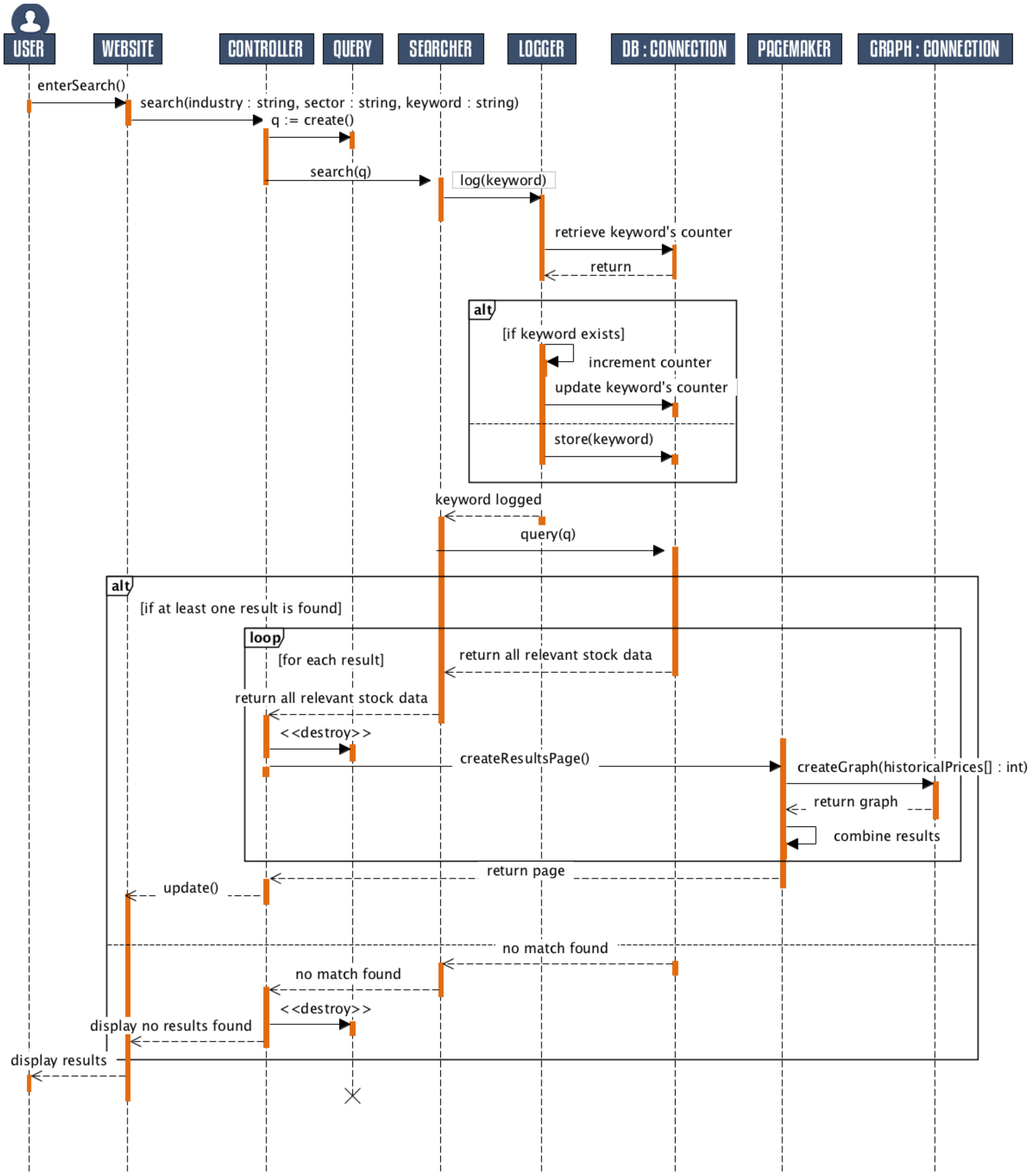

UC-6: Search

Figure-20: Design sequence diagram for UC-6: Search

Figure-20, shows the design sequence diagram for Use Case 6: Search. The controller creates a query using a user's search parameters which includes a keyword, industry, and sector. If a keyword is entered it is logged by the Logger for analytics and then the database is queried by the Searcher using the previously constructed query. For every result, a chart of historical prices is created. The pagemaker combines all relevant information for every search result into a page which is returned to the controller. The controller will then update the website. If no matches are found, the website is simply updated to show this.

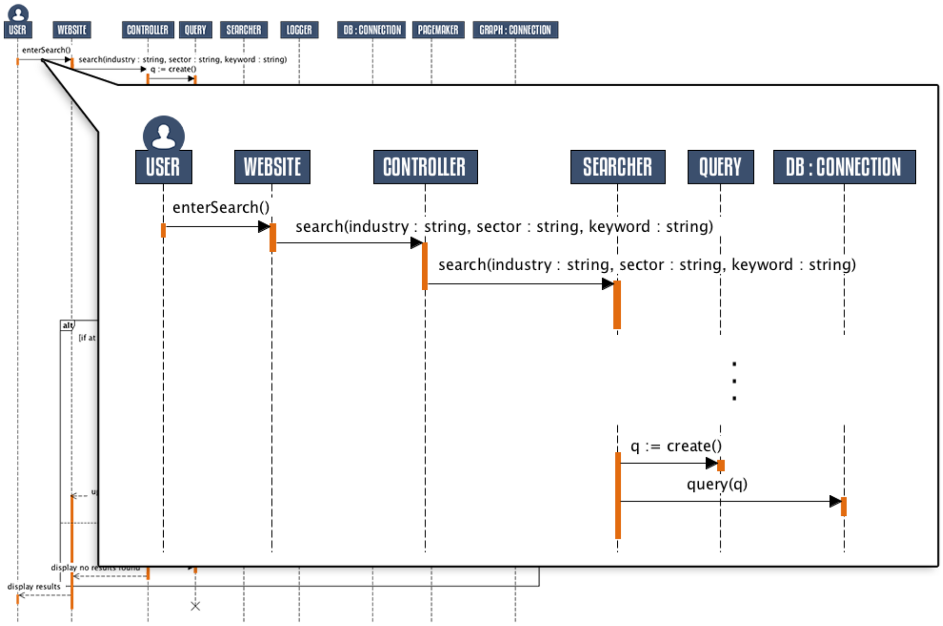


Figure-21: Alternate design for UC-6: Search

Figure-21 displays an alternate design to this sequence where the controller, instead of creating the query itself, passes on the relevant information to the searcher which would then be responsible for creating the query. It should be noted that unrelated intermediate function calls have been removed to further simplify this diagram. This design, again follows the high cohesion principle. The controller's responsibilities are limited to that of just coordinating tasks between concepts and nothing more. Furthermore, this method allows for the query to be created right before its use unlike in Figure-20 where the keyword must first be logged before the query is even utilized. Thus, the data is less prone to corruption. Therefore the method in Figure-21 will be followed.

UC-13: PredictAndNotify

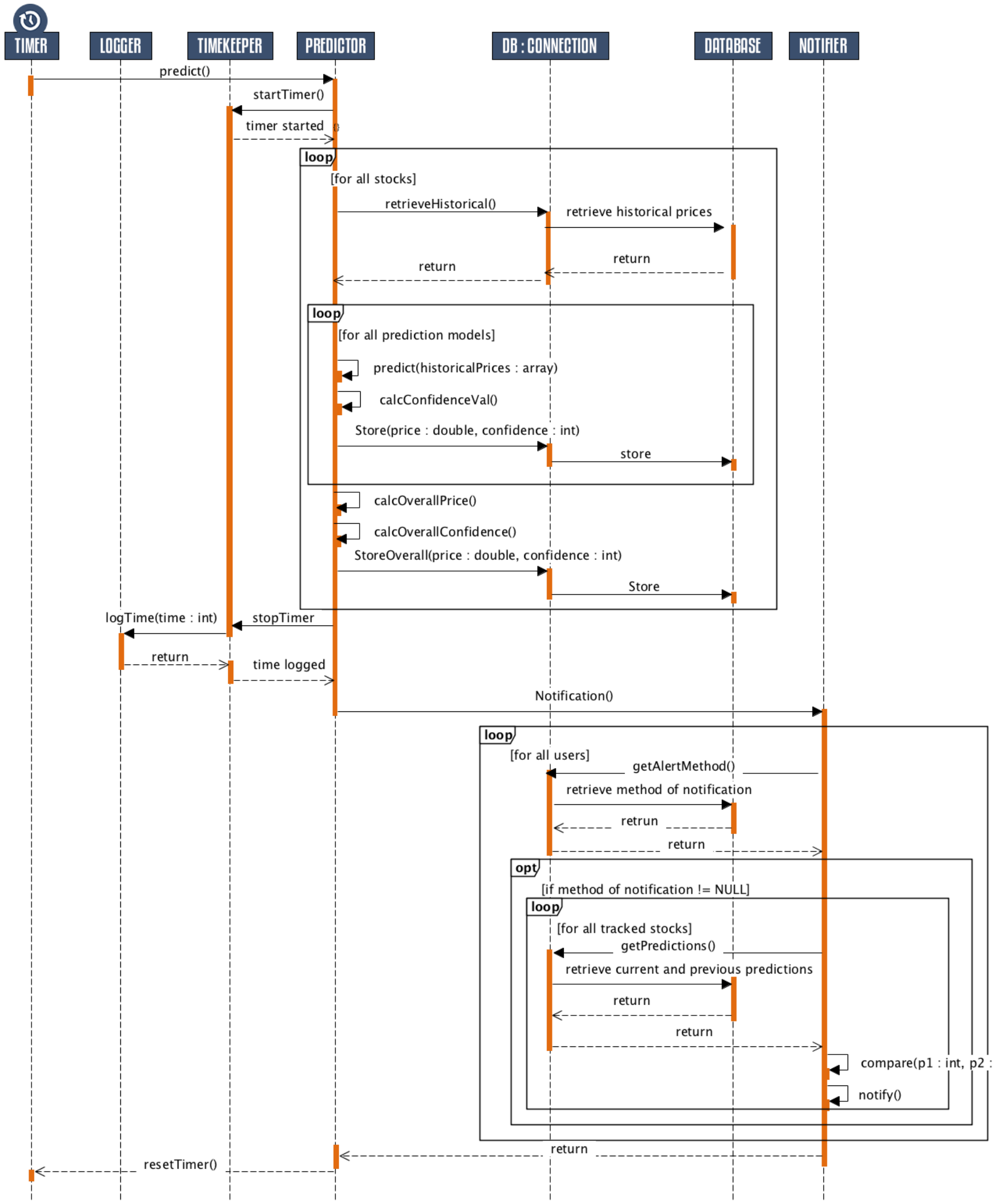


Figure-22: Design sequence diagram for UC-13: PredictAndNotify

Figure-22 shows the design sequence diagram for Use Case 13: PredictAndNotify. A Cronjob will initiate the predictor to start an internal timer within TimeKeeper. The predictor then retrieves historical prices for each stock from the database. For each prediction model, a prediction is made based on those historical prices and a confidence value is calculated, these are then stored into the database. An overall prediction is then made and stored within the database along with its confidence value. The timer is then stopped and the time sent to the logger to be logged. The predictor then calls the Notifier. For each user and for each tracked stock for that user, a comparison is then made between the current prediction and the previous prediction. If a change is detected, the notifier sends out a notification through the users preferred method of notification.

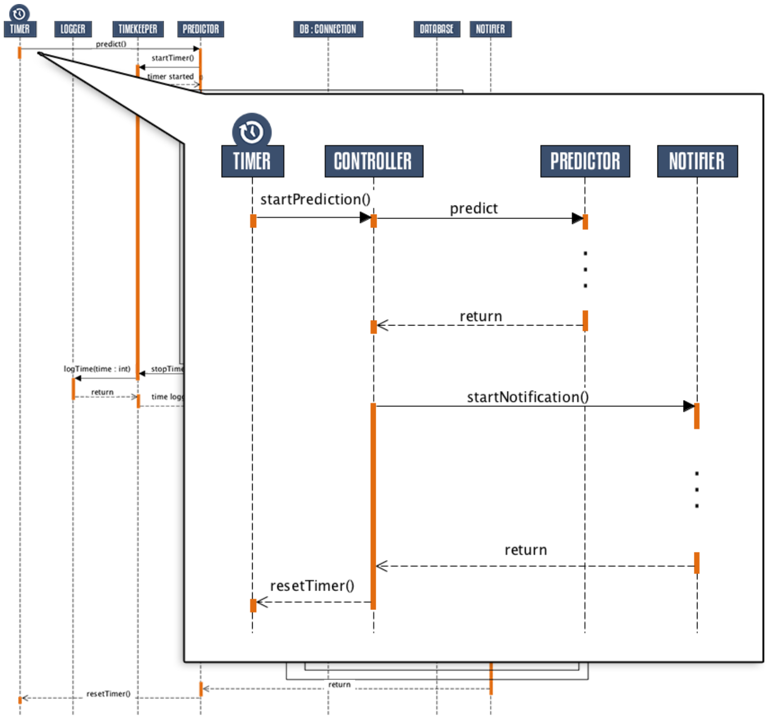


Figure-23: Alternate design sequence diagram for UC-13: PredictAndNotify

In an alternate design, as shown in Figure-23, the controller can be used as a buffer between the initiating actor and the predictor. Although this design appears more eloquent, where the controller serves as the sole communicator (thus not requiring that the predictor call the notifier as before) it would also transform the controller into a boundary concept and therefore leave the entire system to be prone to security faults. Since the controller serves as a way of coordination between all concepts, it should be paramount that no initiating actors have direct control over it. Therefore, this design will not be taken into consideration.

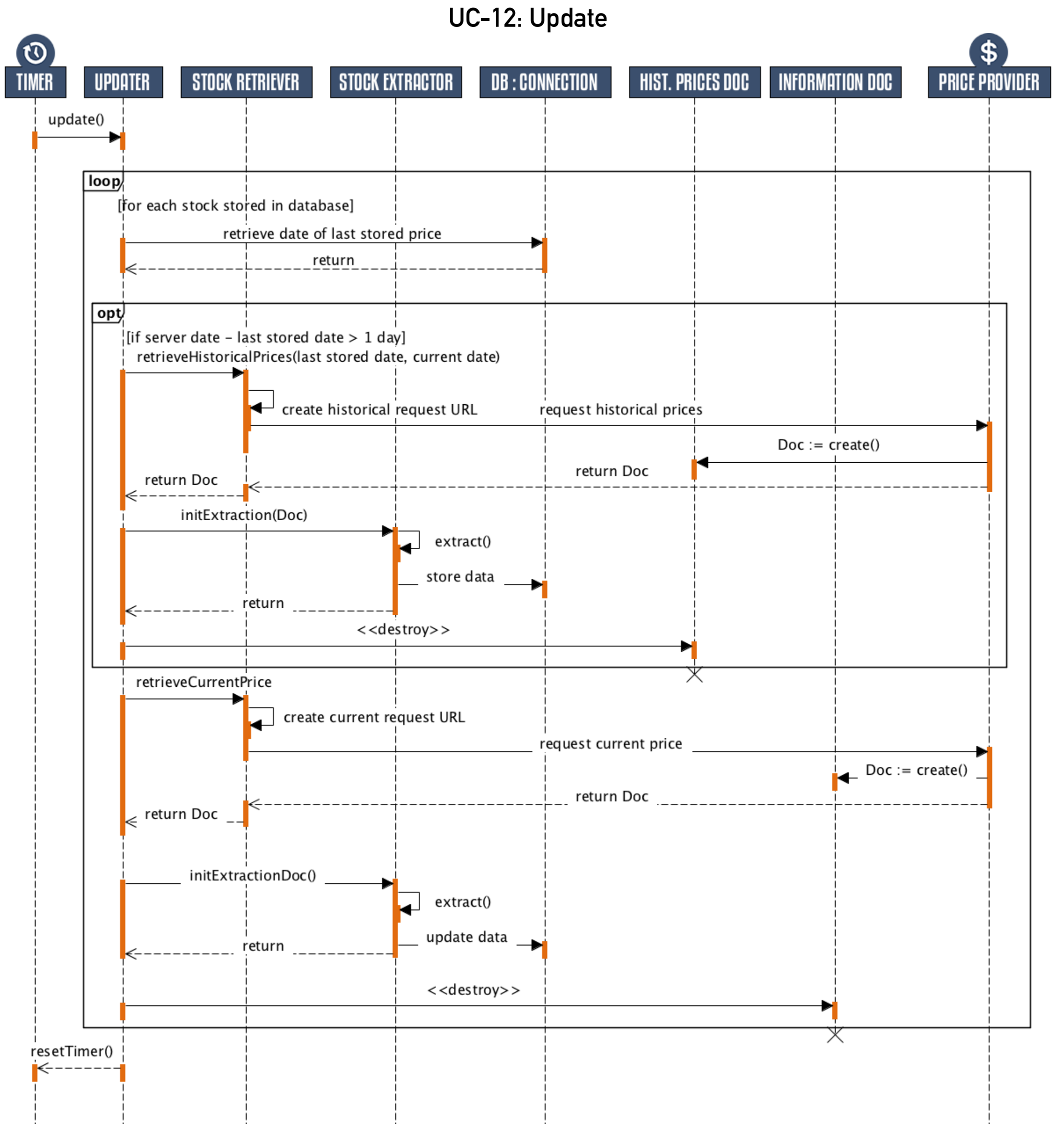


Figure-24: Design sequence diagram for UC-12: Update

Figure-24 shows the design sequence diagram for Use Case 12: Update. A main loop is first entered for each stock being stored within the database. The Updater compares the date of the last stored price for a stock with the server data and if the two differ, historical prices for the missing days are retrieved by the Stock Retriever and then extracted and stored by the Stock Extractor. The current price for each stock, in a similar fashion, is then retrieved by the Stock Retriever and then extracted and stored by the Stock Extractor.

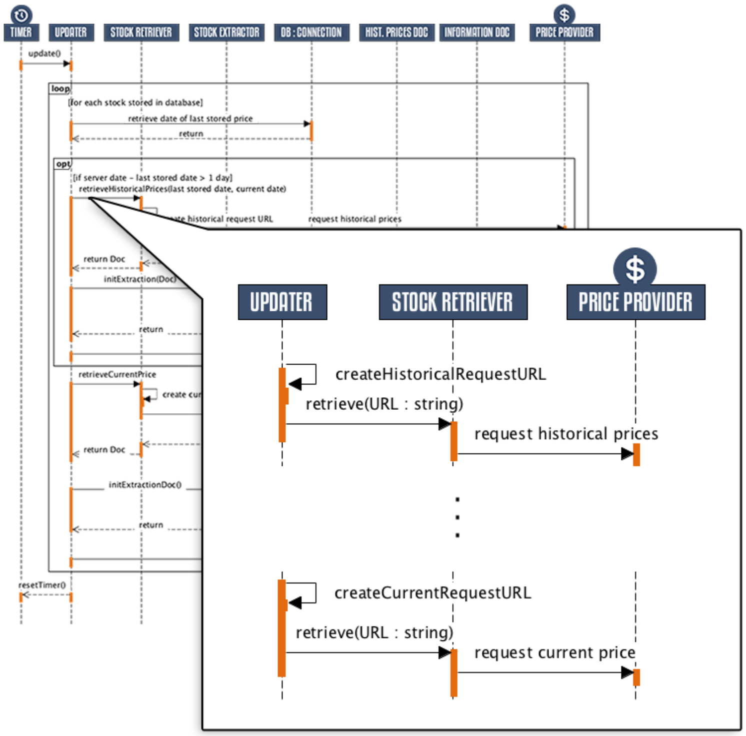


Figure-25: Alternate design sequence diagram for UC-12: Update

An alternate design is shown in Figure-25 where the request URL creation is handled by the Updater and then passed onto the Stock Retriever. This method follows the expert doer principle since the Updater first receives the information necessary to create a request URL. The high cohesion principle, on the other hand, would suggest that URL creation should remain a responsibility of the Stock Retriever. If you recall, we have already chosen that the responsibility of creating URLs remain a process within the Stock Retriever in the design of UC-2: AddStock, and thus for consistency will also choose the same here.

UC-8: Track

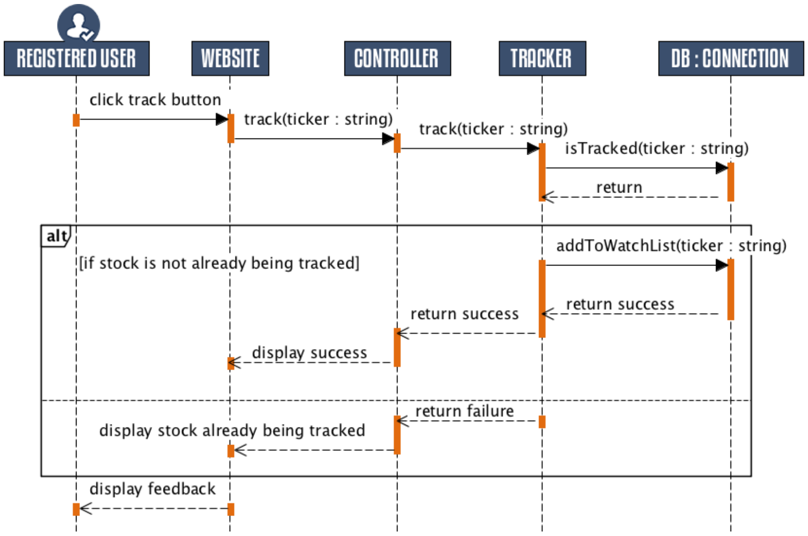


Figure-26: Design Sequence Diagram for UC-8: Track

Figure-26 shows the design sequence diagram for Use Case 8: Track. Once a user clicks the "track" button, the website notifies the controller of the event. The controller passes the associated stock to the Tracker which then will verify if the stock already exists within the user's "watch list". If not, the stock is added and the user is displayed the appropriate feedback through the Website.

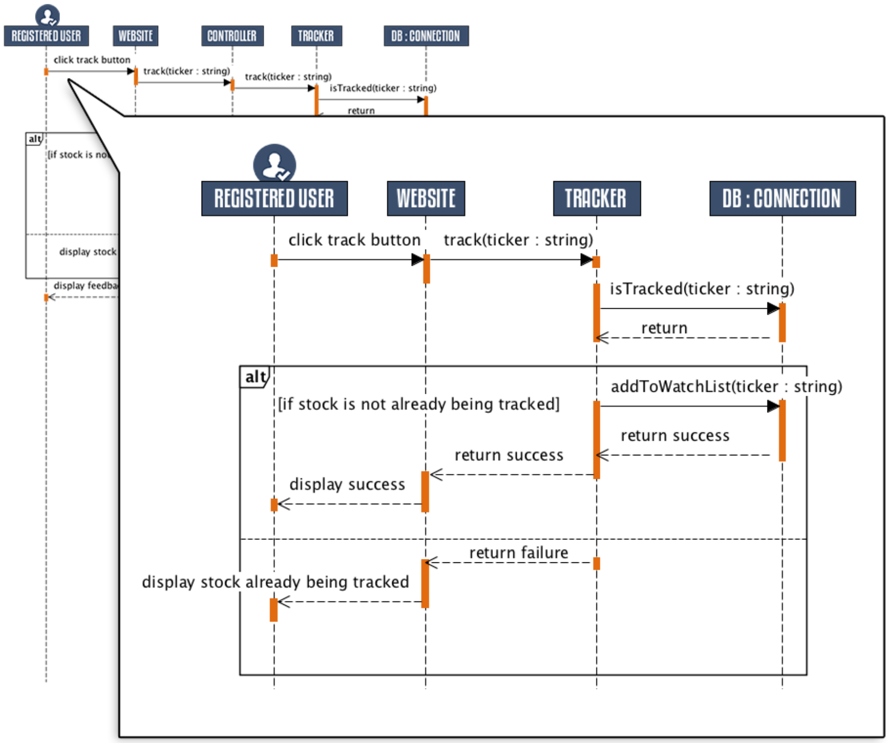


Figure-27: Alternate design sequence diagram for UC-8: Track

It is questionable whether the controller is even needed in this sequence. Figure-27 shows just that case. The controller is removed, and the website communicates directly with the tracker. This method is favored by the low coupling principle and the diagram seems to be greatly simplified. However, as stated before, the controller should be responsible for all communication between concepts (except for special cases) and thus for consistency as well as high cohesion we will refrain from using this design.

2. Class Diagram and Interface Specification

2.1. Class Diagram

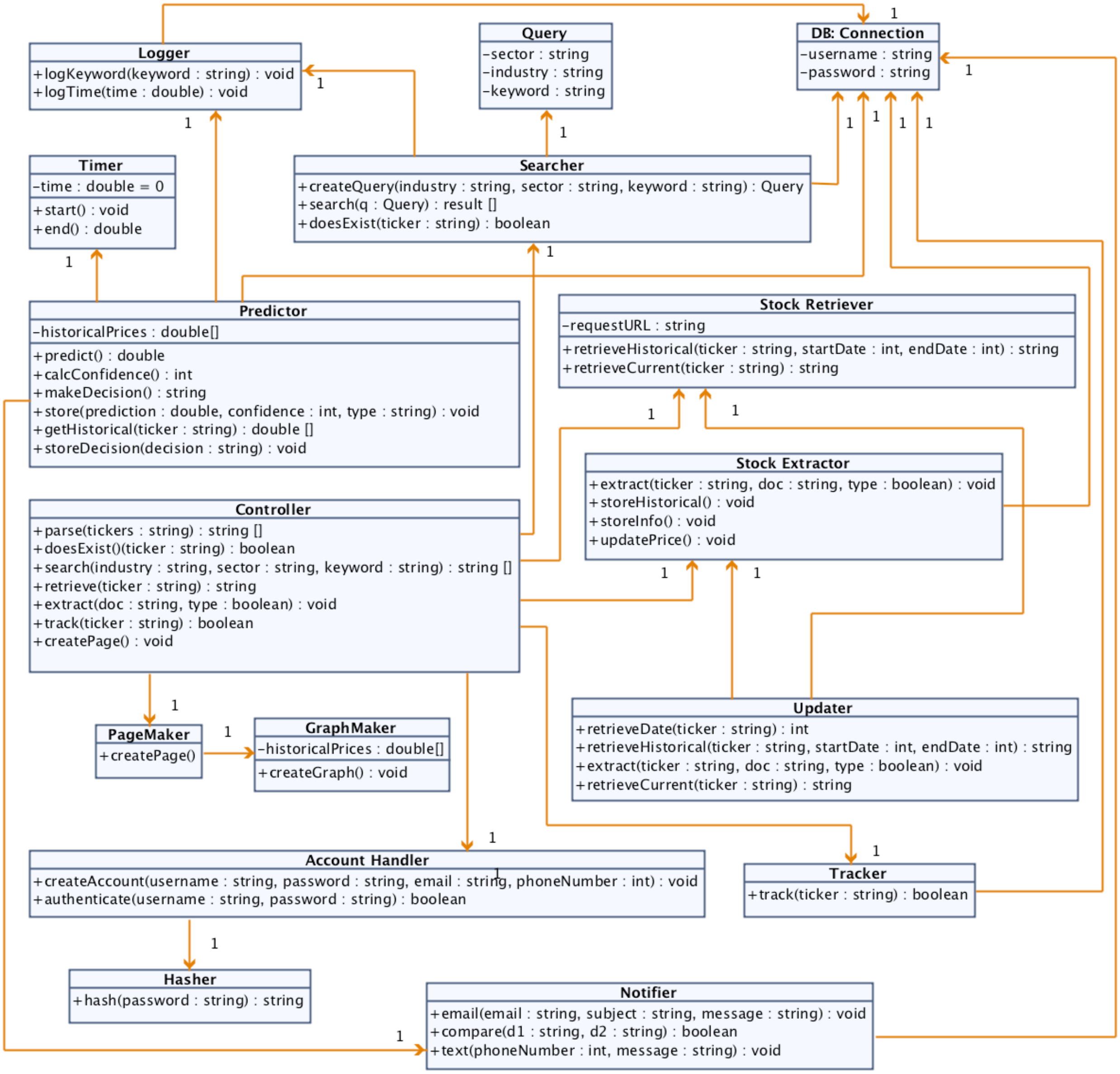

Logger
+logKeyword(keyword : string) : void
+logTime(time : double) : void
Query
-sector : string
-industry : string
-keyword : string
DB: Connection
-username : string
-password : string
Timer
-time : double = 0
+start() : void
+end() : double
Searcher
+createQuery(industry : string, sector : string, keyword : string) : Query
+search(q : Query) : result []
+doesExist(ticker : string) : boolean
Predictor
-historicalPrices : double[]
+predict() : double
+calcConfidence() : int
+makeDecision() : string
+store(prediction : double, confidence : int, type : string) : void
+getHistorical(ticker : string) : double []
+storeDecision(decision : string) : void
Stock Retriever
-requestURL : string
+retrieveHistorical(ticker : string, startDate : int, endDate : int) : string
+retrieveCurrent(ticker : string) : string
Stock Extractor
+extract(ticker : string, doc : string, type : boolean) : void
+storeHistorical() : void
+storeInfo() : void
+updatePrice() : void
Controller
+parse(tickers : string) : string []
+doesExist()(ticker : string) : boolean
+search(industry : string, sector : string, keyword : string) : string []
+retrieve(ticker : string) : string
+extract(doc : string, type : boolean) : void
+track(ticker : string) : boolean
+createPage() : void
PageMaker
+createPage()
GraphMaker
-historicalPrices : double[]
+createGraph() : void
Updater
+retrieveDate(ticker : string) : int
+retrieveHistorical(ticker : string, startDate : int, endDate : int) : string
+extract(ticker : string, doc : string, type : boolean) : void
+retrieveCurrent(ticker : string) : string
Account Handler
+createAccount(username : string, password : string, email : string, phoneNumber : int) : void
+authenticate(username : string, password : string) : boolean
Tracker
+track(ticker : string) : boolean
Hasher
+hash(password : string) : string
Notifier
+email(email : string, subject : string, message : string) : void
+compare(d1 : string, d2 : string) : boolean
+text(phoneNumber : int, message : string) : void
1

Figure-28: Class diagram for the web based stock forecasting system.

Figure-28 shows the derived class diagram including all relevant attributes and operations for our system.

2.2. Data Types and Operation Signatures

Controller

Coordinates the communication and transfer of data between classes.

+parse(tickers : string) : string []

Parses the input string of comma separated ticker symbols by converting it into an array of ticker symbols.

+doesExist()(ticker : string) : boolean

Calls the searcher to verify if a ticker exists within the database.

+search(industry : string, sector : string, keyword : string) : string []

Calls the searcher to search for a stocks matching the given industry, sector, and keyword.

+retrieve()(ticker : string) : string

Calls the Stock Retriever to retrieve necessary stock data and information and returns the respective CSV files.

+extract(doc : string, type : boolean) : void

Calls the Stock Extractor to extract to begin extracting the necessary data from doc (CSV file). "Type" alerts the method of whether the file contains historical prices or current stock information.

+track(ticker : string) : boolean

Calls the tracker to add the given stock to a user's "watch list". Returns true, if successful and false otherwise.

+createPage() : void

Calls the Page Maker to create the results page after a search is initiated.

Predictor

Makes predictions based on a variety of models and calculates confidence values for each prediction. Also calculates an overall predicted price, confidence value, and trade decision based on each individual model.

+predict() : double

Calculates and returns predicted stock price.

+calcConfidence() : int

Calculates and returns confidence value associated with predictions.

+makeDecision() : string

Returns a trading decision based on prediction results from all models.

+store(prediction : double, confidence : int, type : string) : void

Stores predicted price and confidence value into the database. “Type” alerts the method of which prediction model was used and thus which table the information should be saved under.

+getHistorical(ticker : string) : double []

Returns an array of historical prices for a given stock.

+storeDecision(decision : string) : void

Stores the trading decision associated with the overall prediction into the database.

Timer

Keeps track of the elapsed time for a given prediction session.

-time : double = 0

The running time since the timer was started. Initial value is set to 0.

+start() : void

Starts the timer.

+end() : double

Ends the timer and returns the elapsed time.

Logger

Logs keywords from user searches and times calculated by the timer.

+logKeyword(keyword : string) : void

Logs the keyword in a given user search into the database. If the keyword already exists, the counter associated with the keyword is incremented and then updated within the database.

+logTime(time : double) : void

Logs the elapsed time for a given prediction session into the database.

Query

Used to query the database.

-sector : string

User’s selected sector.

-industry : string

User’s selected industry.

-keyword : string

User’s entered search keyword.

Searcher

Queries the database based on a users unique search.

+createQuery(industry : string, sector : string, keyword : string) : Query

Creates a query object with the given sector, industry, and keyword.

+search(q : Query) : result []

Queries the database with the given query and returns the results.

+doesExist(ticker : string) : boolean

Returns true if the given stock exists within the database and false otherwise.

DB: Connection

Establishes a connection to the database.

-username : string

Username required to access the database.

-password : string

Password required to access the database.

Stock Retriever

Retrieves specific stock information from the Price Provider.

-requestURL : string

The initial part of the URL necessary to request a CSV file of stock information from the Price Provider.

+retrieveHistorical(ticker : string, startDate : int, endDate : int) : string

Retrieves and returns a CSV file of historical prices for the given stock within the given time interval.

+retrieveCurrent(ticker : string) : string

Retrieves and returns a CSV file of current stock information for the given stock, including industry, sector, current price, etc.

Stock Extractor

Extracts and stores relevant data from the CSV files returned from the Stock Retriever

+extract(ticker : string, doc : string, type : boolean) : void

Extracts and stores the necessary information from the input CSV file. “Type” alerts the method of whether the file contains historical prices or current stock information.

+storeHistorical() : void

Stores the extracted historical prices into the database.

+storeInfo() : void

Stores the extracted current stock information into the database.

+updatePrice() : void

Updates the current price of a stock within the database.

Updater

Makes sure that prices stored within the database are kept up to date.

+retrieveDate(ticker : string) : int

Retrieves and returns the date of the last stored price for a given stock.

+retrieveHistorical(ticker : string, startDate : int, endDate : int) : string

Calls the Stock Retriever to retrieve and return a CSV file of historical prices for the given stock within the given time interval.

+retrieveCurrent(ticker : string) : string

Calls the Stock Retriever to retrieve and return a CSV file of current stock information for the given stock, including industry, sector, current price, etc.

+extract(ticker : string, doc : string, type : boolean) : void

Calls the extractor to extract the relevant information from the given input file. “Type” specifies whether the file contains historical prices or current stock information.

PageMaker

Creates and organizes a page of results after a user initiates a search.

+createPage()

Creates the results page after a search is initiated.

GraphMaker

Retrieves a graph of historical prices from the Grapher.

-historicalPrices : double[]

An array of historical prices for a given stock.

+createGraph() : void

Graphs the information within the array of historical prices.

AccountHandler

Handles user accounts, specifically the creation of new accounts and the authentication of existing accounts.

+createAccount(username : string, password : string, email : string, phoneNumber : int) : void

Creates an account with the given username, password, email, and phone number.

+authenticate(username : string, password : string) : boolean

Checks if a user is a registered user stored within the system. Returns true if so, and false otherwise.

Hasher

Hashes passwords when creating new accounts to ensure security.

+hash(password : string) : string

Hashes a given password.

Tracker

Handles the addition of new stocks into a user's "watch list".

+track(ticker : string) : boolean

Verifies whether a stock is stored within a user's "watch list" and if not, adds the given stock.

Notifier

Handles the notification of users when a trade decision is predicted

+email(email : string, subject : string, message : string) : void

Sends an email to the given email address with the given subject and message.

+text(phoneNumber : int, message : string) : void

Sends a text message to the given phone number with the given subject.

+compare(d1 : string, d2 : string) : boolean

Compares two trade decisions and returns true if they are the same and false otherwise.

2.3.Traceability Matrix

	SOFTWARE CLASSES															
DOMAIN CONCEPTS	Controller	Query	Predictor	Searcher	Timer	Logger	Stock Retriever	Stock Extractor	Updater	Grapher Maker	Account Handler	Hasher	Tracker	DB: Connection	Page Maker	Notifier
Controller	x															
Query		x														
Credentials											x					
Website																
PageMaker															x	
Account																
DB: Connection														x		
Searcher				x												
Logger						x										
Account Handler											x					
Securer												x				
Tracker													x			
Graph: Connection										x						
Time Keeper					x											
Stock Retriever							x									
Stock Extractor								x								
Current Information Doc																
Historical Prices Doc																
Notifier																x
Predictor			x													
Stock Info																

Figure-29: Domain model to software classes traceability matrix

Figure-29 shows how the software classes map to the domain model. As the matrix shows, a few concepts could not be mapped to. The website is simply the html page a user views on his/her browser. Account and StockInfo are concepts that are storing specific data which will already be stored within the relational database. The concepts of Current Information Doc and Historical Prices Doc will be provided to the system by the Price Provider as CSV files. All of these concepts do not require to be mapped to a specific class since they are all of the “knowing” type and thus perform no important tasks.

3. System Architecture and System Design

3.1. Architectural Styles

Our software uses several architectural styles. They follow:
1) Client/Server
2) Event-driven
3) Rule-based system
4) Database-centric

Client/Server Architecture

Client/Server is our main architectural style; it separates our system requirements into two easily programmable systems. First, the client, which acts as the User Interface, requests data from the server, and waits for the server’s response. Secondly, the server, which authorizes users and processes stock data into information the user can use. It then sends this processed information to the client to display to the user.

Event-driven Architecture

Our system will only need to execute its functions after some major state change. It has no real time components like a video game. Instead, we’ll have two event emitters, the user and the timer. Both will drive the application to execute relevant operations though the execution of different events. These events include login, adding new stocks, deleting stocks, and requesting an updated lists of stocks for the user; and a time-based update for the timer.

Rule-Based System

Our application will be rule-based. In other words, the system will use a set of rules that we determine it to analyze the stock information it gathers. These rules comprise a semantic reasoned which makes decisions for the application and the user. It uses a match cycle act cycle to deduct which stocks will be best to buy and which stocks would be best to sell. Then, it outputs these results to the user-interface.

Database-centric Architecture

Our system relies heavily on its database, both the store relevant stock data and to analyze the data we give it. The database-centric architecture offers:

1) A standard relational database management system. This means the data will be stored away from the client side application.
2) Dynamic table-driven logic. We need to update the tables every time stock prices change.
3) Stored procedures running on database servers to analyze our data.
4) A shared database for communication between parallel processes

In short, it's a good way of managing a large amount of data

3.2. Identifying Subsystems

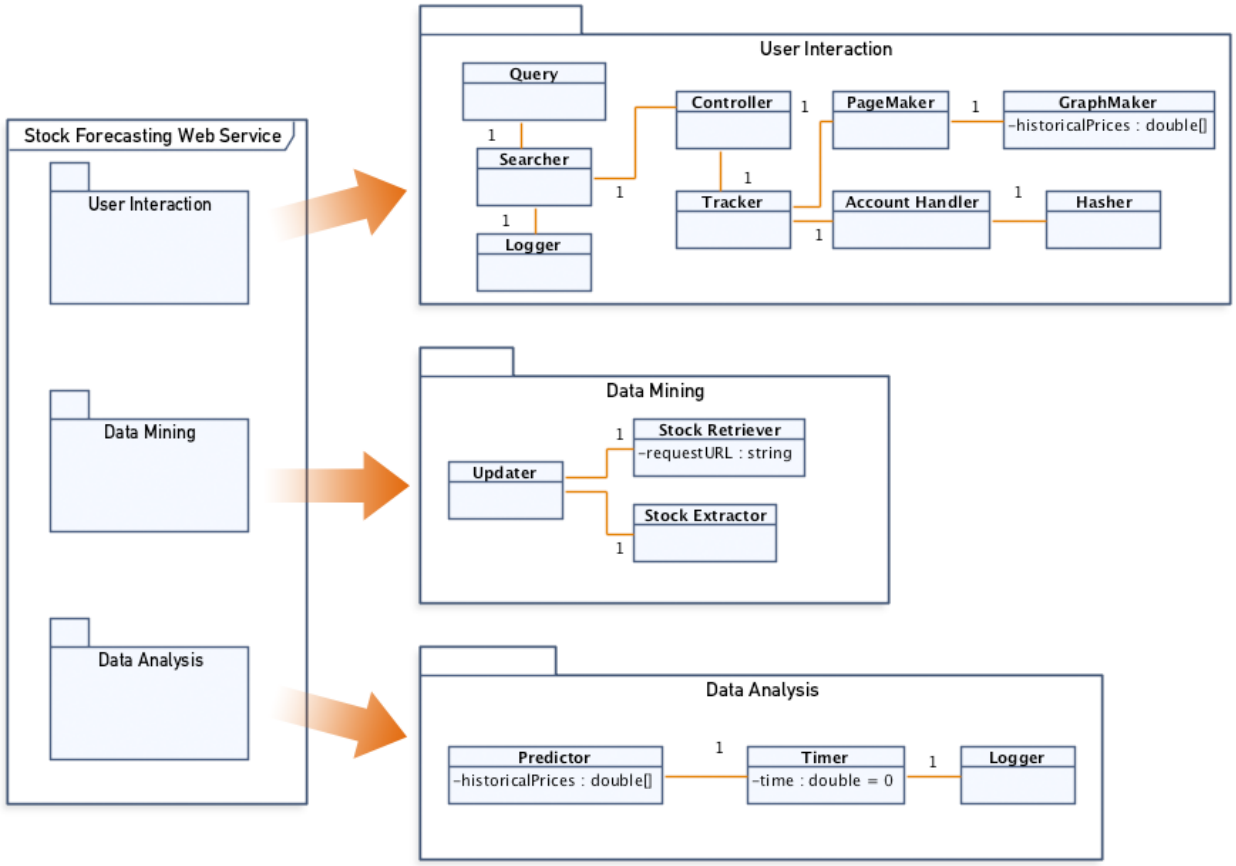


Figure-30: Package diagram showcasing the three subsystems within the main system.

Digging deeper into the system we observe three subsystems emerge as shown by Figure-30 as well as the classes relating to each subsystem. The user interaction subsystem deals with the web interface and basic input/outputs and encompasses any class that assists in such matters. The data mining subsystem deals with gathering relevant stock information at a fairly frequent intervals and storing that data locally, incorporating the classes of Updater, Stock Retriever, and Stock Extractor. The data analysis subsystem holds the responsibility of implementing prediction models on the previously mined data and storing those predictions for later use.

3.3. Mapping Subsystems to Hardware

Our software employs the use of a client/server system. The client–server model of computing is a dispersed application structure that divides tasks between the provider of a resource or service, called the server, and an entity making a request, called the client. This establishment can be made with a network connection between host (the server) and client or it is possible for this relationship to exist on the same system, sharing hardware.

The web-based stock forecasting design we presume to execute will need to be accessible anywhere that web access exists. This means the client in our client/server relationship will be a web client. A web browser is an example of a web client, and can remotely access requested documentation from the server via HTTP. This web client/server model will need to be run across multiple computers, or subsystems.

A web browser will be used to request the various data from our server, as well as retrieve stock information and updates that can be sent and retrieved via communication with that server. The GUI, which will run on the web browser, will be executed client side while the process itself is handled by the server's web service. The web service will ensure for proper transmission of user data between the client and the server.

3.4. Persistant Data Storage

The system-to-be requires data to be saved in order to outlive a single execution of the system. This data includes historical stock prices for all relevant stocks, related information for all relevant stocks, user account information, user search queries, and the timers dictating when the system should update current stock prices as well as begin calculating predictions. In order to organize this data, one must first elaborate on the different methods of storage.

A flat file database typically consists of multiple text files storing one record per line. Text files are simple and portable and can, in most cases, be used without requiring special software. They make it, however, difficult to search for specific information or to create reports that include only certain fields from each record. Thus, when creating new records, numerous redundancies occur as a portion of information in one file must be rewritten to all others.

A relational database, on the other hand, consists of multiple tables linked by "keys" — certain pieces of information shared by two or more tables. This model takes advantage of the uniformity to build completely new tables out of required information from existing tables. In other words, it uses the relationship of similar data to increase the speed and versatility of the database.

The implementation of the timers previously stated will be through a Cronjob and thus a flat file is ultimately the best case for storing them since they require a simple line of code each such as:

```
0 */2 * * *  /home/username/test.php
```

Which makes the user script test.php run every two hours, at midnight, 2am, 4am, 6am, 8am, and so on.

For all other information the system-to-be relies heavily on large amounts of data storage, organization, analysis, and security. Thus, a relational database will be used for its efficiency and power. The tables are organized as follows:

STOCKS

Field	Type	Null	Key	Default	Extra
StockID	int(11)	NO	PRI	NULL	auto_increment
Company	varchar(20)	NO		NULL	
Ticker	varchar(5)	NO		NULL	
Industry	varchar(20)	NO		NULL	
Sector	varchar(20)	NO		NULL	

Table-1: Organization of the table *Stocks* within the database.

The *Stocks* table, as shown by Table-1, holds all stock related information including the company name, ticker symbol, industry, and sector. An auto incremented ID is used to insure that each entry is unique.

HISTORICAL PRICES

Field	Type	Null	Key	Default	Extra
StockID	int(11)	NO	PRI	NULL	
Date	date	NO		NULL	
Open	decimal(10,0)	NO		NULL	
High	decimal(10,0)	NO		NULL	
Low	decimal(10,0)	NO		NULL	
Close	decimal(10,0)	NO		NULL	
Volume	int(11)	NO		NULL	

Table-2: Organization of the table *Historical Price* within the database.

The *Historical Prices* table, as shown by Table-2, holds all historical data for all stocks. This historical data includes the data, opening price, daily high price, daily low price, closing price, and volume. Stocks are differentiated from each other using the stockID established in the *Stocks* table.

USERS

Field	Type	Null	Key	Default	Extra
UserID	int(11)	NO	PRI	NULL	auto_increment
Email	varchar(20)	NO		NULL	
Password	varchar(20)	NO		NULL	
SecurityKey	varchar(30)	NO		NULL	
Phone	int(10)	NO		NULL	

Table-3: Organization of the table *Users* within the database.

The *Users* table, as shown by Table-3, holds all user account related information including userID, email address, password (hashed), security key provided by the hasher, and phone number. The userID field is an auto incrementing integer, ensuring that every stored user is unique.

OVERALL PREDICTION

Field	Type	Null	Key	Default	Extra
StockID	int(11)	NO		NULL	
Date	date	NO		NULL	
PredictedPrice	int(11)	NO		NULL	
ConfidenceValue	int(11)	NO		NULL	
PredictedDecision	varchar(10)	NO		NULL	

Table-4: Organization of the *Overall Prediction* table within the database.

The Overall Prediction table, as shown by Table-4, holds the overall prediction results and related information. This includes the prediction date, predicted price, calculated overall confidence value, and predicted trade decision. The stockID from the *Stocks* table is used to differentiate between different stocks.

PREDICTION1

Field	Type	Null	Key	Default	Extra
StockID	int(11)	NO		NULL	
Date	date	NO		NULL	
PredictedPrice	int(11)	NO		NULL	
ConfidenceValue	int(11)	NO		NULL	

⋮

PREDICTION8

Field	Type	Null	Key	Default	Extra
StockID	int(11)	NO		NULL	
Date	date	NO		NULL	
PredictedPrice	int(11)	NO		NULL	
ConfidenceValue	int(11)	NO		NULL	

Table-5: Organization of the intermediate prediction tables within the database.

The system will utilize a total of eight intermediate prediction models and a table will exist to store predictions results and related data for each, ash shown by Table-5. This includes the current data, the predicted price, and the calculated confidence value. The stockID from the *Stocks* table is used to differentiate between stocks.

TRACKEDSTOCKS

Field	Type	Null	Key	Default	Extra
UserID	int(11)	NO		NULL	
StockID	int(11)	NO		NULL	

Table-6: Organization of the *Tracked Stocks* table within the database.

A table will be used to keep track of user's "watch lists". The *Tracked Stocks* table, as shown by Table-6, will associate a userID from the *Users* table to a stockID in the *Stocks* table.

KEYWORDS

Field	Type	Null	Key	Default	Extra
KeywordID	int(11)	NO	PRI	NULL	auto_increment
keyword	varchar(20)	NO		NULL	
counter	int(11)	NO		NULL	

Table-7: Organization of the *Keywords* table within the database.

The Keyword table, as shown by Table-7, will store all keywords used for user search queries. Each entry gets a unique identifier, the keyword, and counter specifying the number of times the keyword was queried.

TIMES

Field	Type	Null	Key	Default	Extra

Date	date	NO	PRI	NULL	
time	double(10)	NO		NULL	

Table-7: Organization of the *Times* table within the database.

The Times table, as shown by Table-7, holds the amount of time a given prediction session takes. Each time is associated with the date the prediction session had taken place. This will serve useful for future analytics.

Figure-31 below shows the relationship between all the aforementioned tables. Tables *Keywords* and *Times* share no relation and thus were omitted from the diagram.

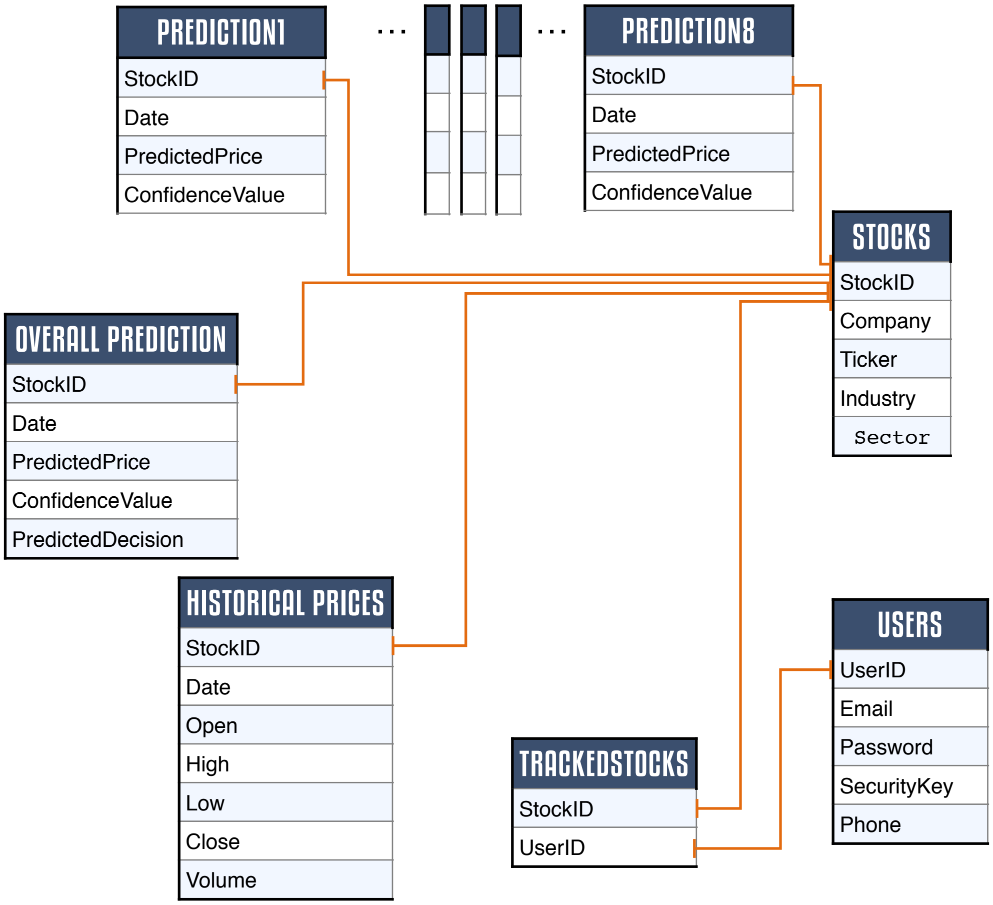


Figure-31: Relationship between tables within the database

3.5. Network Protocol

Simply, our software will communicate with a single main database. This database will use PHP scripts to both send data to our user's localized systems and call data from Yahoo stocks for analysis. The data itself will be managed by SQL software, and the PHP will output HTML to user's systems.

The components will be connected in the following way:
1) PHP requests for raw stock data from Yahoo
2) The data is stored by using SQL
3) Using SQL, we will apply our stock analysis algorithms
4) PHP waits for prompt from user's systems.
5) When prompted, PHP converts analyzed data into HTML
6) User's system converts HTML to working UI

We decided to use PHP because it is standard in creating dynamic web pages. Furthermore, it works well with relational database management systems. We chose SQL because it is the standard RDMS used to manage and manipulate large amounts of data. Lastly, we use HTML because, with the release of HTML 5, HTML has become one of the most powerful and simple markup languages for developing web pages.

3.6. Global Control Flow

Execution Orderliness:
The system can generally be defined as event-driven; it will wait for a user to make an action before processing data. The user's interaction will characterize their visit and the control structure will wait for the user's request, remaining idle until it receives such information. This allows for a user to sequence their actions upon a visit in different patterns without confusing the system. Some actions may require multithreading in order for updating to be accomplished thoroughly.

Time dependency:
The software will make use of timers to keep current, up-to-date, information regarding stocks in our database at all times. This is a real-time system that will update the database at exact defined times throughout a given day.

Concurrency:
Our system has been implemented to function on the Web; this means that multithreading must be supported. We expect that at some instance there will be concurrent users accessing either the website or the database, so that will be accounted for using multiple threads.

3.7. Hardware Requirements

There are really three instances that need to be addressed when looking at hardware specification; the hardware to run the server, the hardware to access the website, and what type of hardware is needed to use any supplemental mobile technology.

The server

The server requires a processor that has at least one 1.4 GHz single core (64-bit) or a 1.3 GHz dual-core, 2 Gigabytes of RAM, and has at least 10 gigabytes of hard drive available. The server computer must have networking capability allowing access to a router either via network cable or wirelessly. The router should be an UPnP-certified device; however this is not a requirement.

To access the website (via Computer or Mobile)
The website is accessible through any web-enabled device. Although having a display resolution of 1024x768 or greater is highly recommended.

Mobile App
In the event that a mobile app is developed, a smartphone will be required (iOS or Android).

These requirements are based off the use of a smaller database that if expanded may require more resources. Also, note that these not necessarily the minimal requirements, but are recommended for optimal performance.

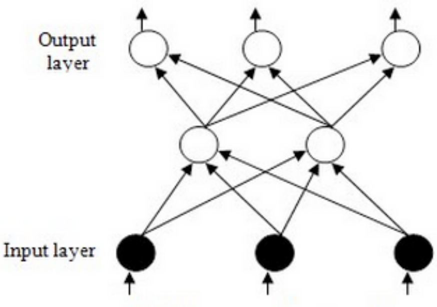
4. Algorithms and Data Structures

4.1. Algorithms

Artificial Neural Network (ANN)

An Artificial Neural Network is a system built upon the observation of how neural networks behave naturally. The practice of how artificial neurons relate to neural network training, or learning, is what the task at hand demands. This essentially means how we can adapt a mathematical algorithm to simulate the human brain.

These networks are excellent for designing the behavior of more complicated structures because of there ability to learn. A major advantage in using this method for what we seek to accomplish is that they can be used to model a system of events, stock trends, that are totally unknown and how it will react to distortion and interference or in our case the imperfections of the stock market.

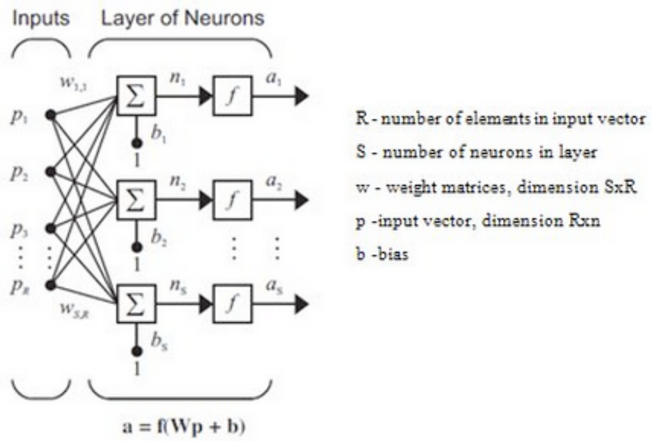


Multilayer neural network

Neuron networks can have up to three layers; the use of more layers will not improve the quality but will extend the learning process.

MATLAB features a special add-on application as one of their toolbox features known as the neural network toolbox. This means through the use of this software we can adapt the information we know about stocks to their system that is already set up to handle neural networks. Also, the code for all the activation and training algorithms has already been developed. It has all the information regarding structure that is required; structure of layers, and connectivity between them.

In order to do this MATLAB must first be “trained” with data. The data will be imported via a file that is created from historical data from a database, such as our database or a database such as Yahoo Finance. This data will have to be defined into a fitting problem for the toolbox. A set of input vectors will be arranged into columns in a matrix, while target vectors will be arranged in a second matrix.



Once the data is ready, the neutral network can be created. Through the use of various properties such as network type, layer number, train, adapt, performance, properties for, and the actual training itself a created network will be visualized graphically and made available to the user.

At this point our data will be pre-trained. Meaning our software team will have to manually upload historical data into MATLAB and analyze it to create graphical data. Our software proposes to give the user access to view these models versus actual graphical data, and for insight into the future. This is subject to change to a more dynamic and automated method that we seek to implement.

Autoregressive integrated moving average (ARIMA)

ARIMA models are said to be one of the most generic models for forecasting a time series of events that can be predict future trends. This technique is a regressive analysis that is an established version of the random-walk and random-trend models used in stock prediction. The random walk model is a mathematical and financial theory approach to moving randomly forward, while random trend looks at the random walk model over a period of time to compare differences. This model will build upon that by adding lags of the differenced series and lags of forecasting errors.

The mathematical model below is the theoretical approach to the "ARIMA(p,d,q)" model;

The **p**, **q**, and **d** are defined as follows:

p is the number referring to order of autoregressive terms
d is the number of differences
q is the moving average

Given a time series of data X_t and index t an ARIMA model is:

$$\left(1-\sum_{i=1}^{p'}\alpha_i L^i\right) X_t = \left(1+\sum_{i=1}^{q}\theta_i L^i\right)\varepsilon_t$$

L is the lag, α_i is a parameter of auto regression, θ_i is a parameter of moving average and ε_t is the error.

Now, given that the term $\left(1-\sum_{i=1}^{p'}\alpha_i L^i\right)$ has a root of d, it can be rewritten as:

$$\left(1-\sum_{i=1}^{p'}\alpha_i L^i\right) = \left(1-\sum_{i=1}^{p'-d}\phi_i L^i\right)(1-L)^d$$

To get this model into ARIMA(p,d,q) form, p=p'−d:

$$\left(1-\sum_{i=1}^{p}\phi_i L^i\right)(1-L)^d X_t = \left(1+\sum_{i=1}^{q}\theta_i L^i\right)\varepsilon_t$$

Simple Moving Average (SMA)

A Simple Moving Average can be calculated by adding closing prices over a time period and then dividing that total by the time periods. This means that slow reactions are formed from long-term averages while faster reactions can be seen in a shorter term. These are averages over a given time-frame, traders can watch how short-term and long-term averages relate to one another and seen beginnings to an uptrend.

A generic mathematical model can be seen below.

The *n day simple moving average* for day d is computed by:

$$A_d = \frac{\sum_{i=1}^{n} M_{(d-i)+1}}{n} \quad n \leq d$$

For ten measurements, M_1 to M_{10}, we can calculate a four day moving average using the moving averages from these consecutive days:

$$\begin{aligned} A_4 &= (M_4 + M_3 + M_2 + M_1)/4 \\ A_5 &= (M_5 + M_4 + M_3 + M_2)/4 \\ &\vdots \\ A_{10} &= (M_{10} + M_9 + M_8 + M_7)/4 \end{aligned}$$

Four days of data are required before a four day moving average can be calculated, hence why the first term is A4.

Exponential Moving Average (EMA)

The Exponential Moving Average is very similar to the aforementioned SMA, but in this case more weight is given to the latest data. This model will change even more rapidly than the SMA to recent changes in price. This approach is generally more commonly used than the SMA.

The mathematical approach would be:

$$EMA = (P * \alpha) + [Previous\ EMA * (1 - \alpha)]$$

Where P = Current Price, α = Smoothing factor = $2/(1+N)$, and N = Number of time periods.

Rate of Change (ROC)

Rate of Change is a technical indicator that is a measurement of percent changes between recent prices and the price over a period of the past. It is an indication of the momentum of trends. Divergence in the ROC is generally a great indication that a drastic decline (or incline) may be in the near future by comparison of it with the price of its asset.

Mathematically, ROC can be simply looked at as:

$$ROC = \frac{(Current\ Closing\ Price - Closing\ Price"\ N")}{Closing\ Price\ "N"}$$

Where N = number of elapsed periods.

Relative Strength Index (RSI)

Relative Strength Index is another technical momentum indicator where the magnitude of recent changes in gain or loss determines conditions of assets that are bought and sold over their intended limit. This is generally done on a scale of 100 where overbought is 70 and above(overvalued asset) and when the RSI goes to 30 or below it is an indication that it is being sold too frequently and is under its intended value.

A simple mathematical approach:

$$RSI = 100 - \frac{100}{(1 + RS)}$$

Where RS is = (average gain)/(average loss)

Average True Range (ATR)

Average True Range is a moving average of true ranges. A true range indicator is an indication of the greatest factor when considering whether a current high is less than a current low, the absolute value of the current low are less than the previous close, and the absolute value of the current high is less than the previous close. Generally a stock with a high level of volatility will have a higher ATR while a low volatility stock has a lower ATR.

ATR calculation is as follows:

The range is simply defined as high minus low.

$$\text{true range} = \max[(\text{high} - \text{low}), \text{abs}(\text{high} - \text{close}_{\text{prev}}), \text{abs}(\text{low} - \text{close}_{\text{prev}})]$$

The **true range** is the largest of the following:
- Recent period's high minus recent period's low
- Absolute value of recent high minus the previous close
- Absolute value of recent low minus the previous close

The ATR at the moment of time t is calculated using the following formula:

$$ATR_t = \frac{ATR_{t-1} \times (n-1) + TR_t}{n}$$

Where the first ATR value is calculated using the arithmetic mean formula:

$$ATR = \frac{1}{n}\sum_{i=1}^{n} TR_i$$

Average Directional Index (ADX)

Average Directional Index is an indicator of trend strengths based off of an objective value. ADX will show the strength of trend regardless of whether it is up or down. This is a bit different from the other methods in that it does not indicate direction or momentum. This indicator is usually observed verses the DMI, (Directional Movement Indicators) since ADX is a derivation of the relationship it has among these DMI lines in graphical terms.

The ADX is a combination of two other indicators, the positive directional indicator (+DI) and negative directional indicator (-DI). The ADX combines them and smooths the result with an exponential moving average.

Calculation of +DI and –DI needs price data consisting of high, low, and closing prices for every time period. First calculate the directional movement (+DM and −DM):

UpMove = today's high − yesterday's high
DownMove = yesterday's low − today's low
if UpMove > DownMove and UpMove > 0, then +DM = UpMove, else +DM = 0
if DownMove > UpMove and DownMove > 0, then −DM = DownMove, else −DM = 0

After selecting the number of periods, +DI and −DI are:

$$+\mathrm{DI} = 100 * \frac{EMA\ of + DM}{ATR}$$

$$-\mathrm{DI} = 100 * \frac{EMA\ of - DM}{ATR}$$

The EMA is calculated over the number of time periods, and the ATR is an exponential average of the true ranges. Thus:

$$ADX = 100 * \frac{EMA\ of\ |(+\mathrm{DI}) - (-\mathrm{DI})|}{[(+\mathrm{DI}) + (-\mathrm{DI})]}$$

Accumulative Swing Index (ASI)

Accumulative Swing Index is a variation of Welles Wilder's swing index. It is a way of comparison of bars that contain high, low, opening, and closing prices in a given time period. These bars can be defined as a value from 0 to 100 is an up bar and 0 to -100 is a down bar. When this index is an up bar it conveys that the long-term trend will be higher, and when it is a down bar it suggests that this trend may be lower.

The mathematical approach:

$$Swing\ Index = 50\ \left(\frac{CC - PC + 0.5(CC - CO) + 0.25(PC - PO)}{R}\right)\frac{K}{L}$$

The variables are defined as;

PO = Prior day’s Open

CO = Current day's Open
PH = Prior day's High
CH = Current day's Open
PL = Prior day's Low
CL Current day's Low
PC = Prior day's Close
CC = Current day's Close
K = (the larger of the two) (1) $CH - PC$ or (2) $CL - PC$
L = Limit move
Choosing R is a multi-step process.

R = the largest of the three choices
(1) $CH - PC$
(2) $CL - PC$
(3) $CH - CL$

If (1), $R = (CH - PC) - 0.5(CL - PC) + 0.25(PC - PO)$
If (2), $R = (CL - PC) - 0.5(CH - PC) + 0.25(PC - PO)$
If (3), $R = (CH - CL) + 0.25(PC - PO)$

4.2. Data Structures

Our system, that is to be developed, will ultimately use a database to store all of its data. Making use of SQL, our database will contain tables holding the information being stored and/or retrieved. The timers, implemented through a CronJob, will be stored in a flat-file. We thought this approach was the most simple and accurate way of storing data thoroughly. The benefits of using both the flat file as well as the relational database have been extensively discussed in section 3.4 and can be read through again for further clarification. A brief summary is given below:

Text files are simple and portable and can, in most cases, be used without requiring special software. Since, we only need to store two timers, a relational database will offer no benefits in speed or organization. The implementation of the timers previously stated will be through a Cronjob and thus a flat file is ultimately the best case for storing them since they require a simple line of code each such as:

```
0 */2 * * *  /home/username/test.php
```

Which makes the user script test.php run every two hours, at midnight, 2am, 4am, 6am, 8am, and so on.

A relational database consists of multiple tables linked by "keys" — certain pieces of information shared by two or more tables. This model takes advantage of the uniformity to build completely new tables out of required information from existing tables. In other words, it uses the relationship of similar data to increase the speed and versatility of the database. For most of its information the system-to-be relies heavily on large amounts of data storage, organization, analysis, and security. Thus, a relational database will be used for its efficiency and power when dealing with these large amounts of data.

5. User Interface Design and Implementation

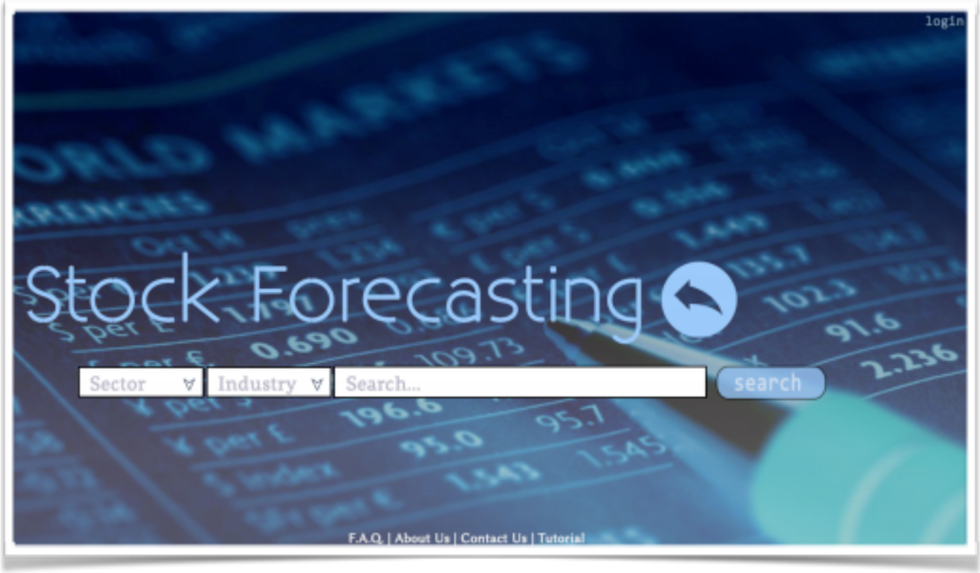


Figure 32: Home Page

Upon entering the site, the user is presented with the following interface. A search function is included on the home page for ease-of-use as the user can now quick search any stock without the hassle of logging in. The login button has been moved to the top right corner so as to separate it from the search button — this saves the user from having to differentiate between buttons.

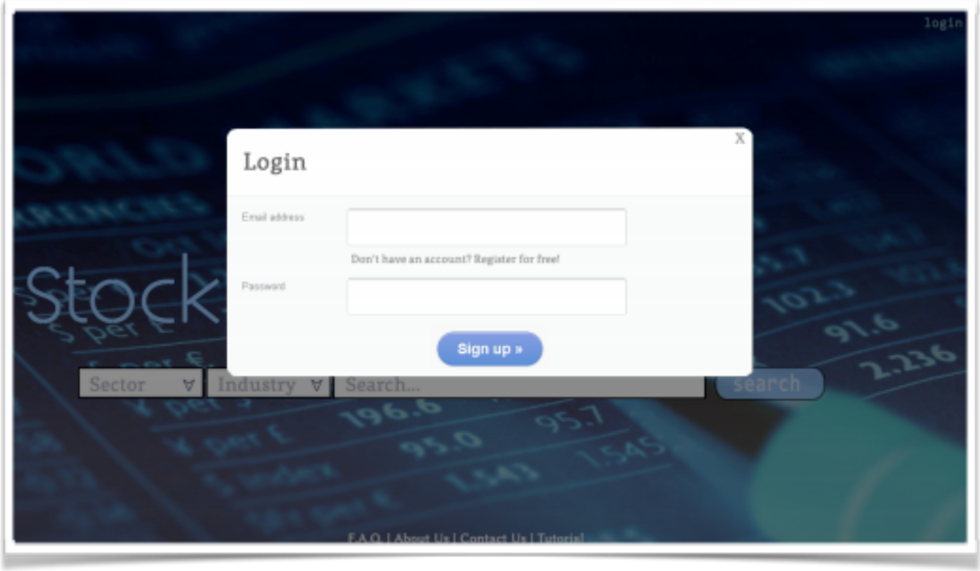
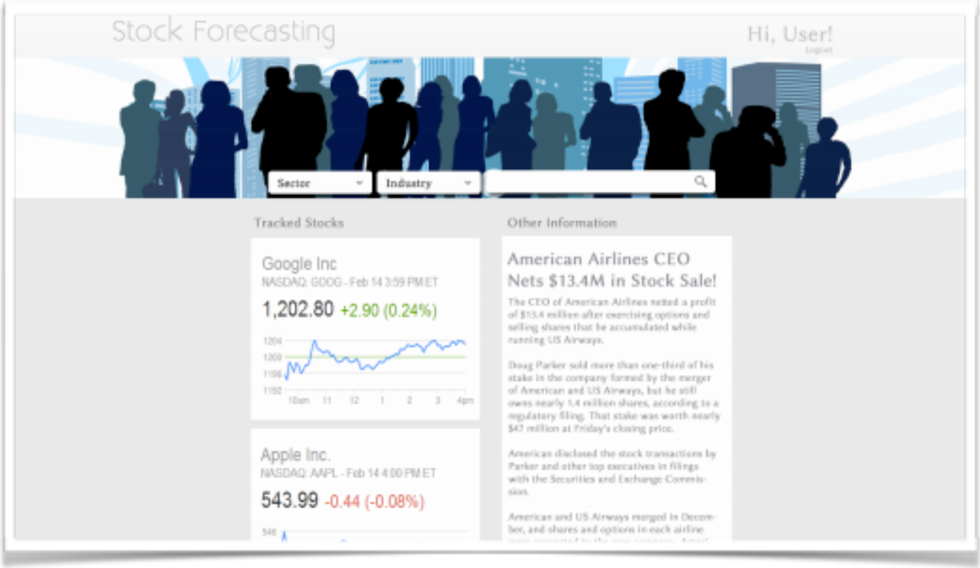

login
X
Login
Email address
Don't have an account? Register for free!
Password
Sign up »
Stock
Sector
Industry
Search...
search
F.A.Q. | About Us | Contact Us | Tutorial

Figure-33: Login

Clicking the Login button calls the above interface. It is a standard login query which requires the user input two pieces of data: a username and a password. An account registration link is also included on this interface so the user does not have to close the login interface to register an account.



Google Inc
NASDAQ: GOOG - Feb 14 3:59 PM ET
1,202.80 +2.90 (0.24%)

Apple Inc.
NASDAQ: AAPL - Feb 14 4:00 PM ET
543.99 -0.44 (-0.08%)

American Airlines CEO Nets $13.4M in Stock Sale!

The CEO of American Airlines netted a profit of $13.4 million after exercising options and selling shares that he accumulated while running US Airways.

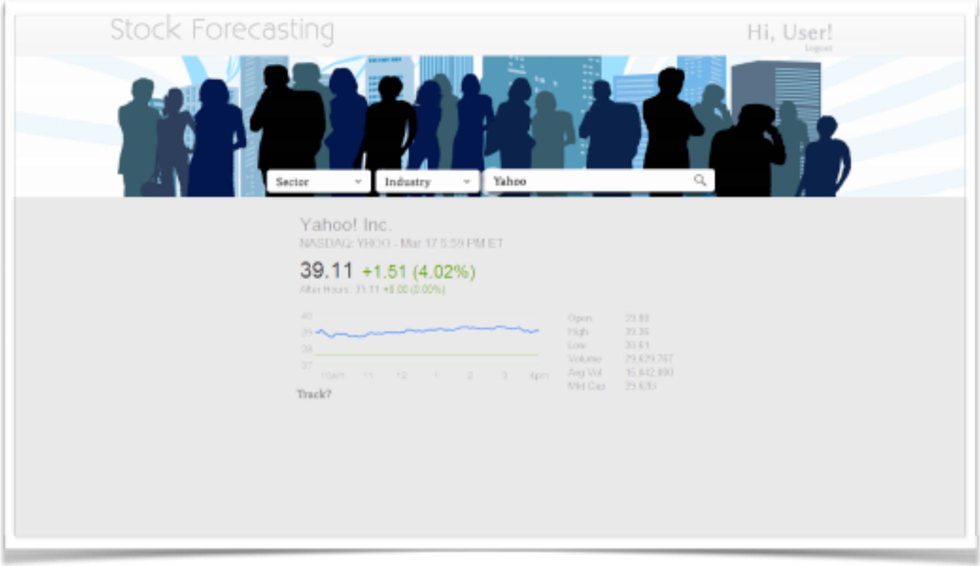
Doug Parker sold more than one-third of his stake in the company formed by the merger of American and US Airways, but he still owns nearly 1.4 million shares, according to a regulatory filing. That stake was worth nearly $47 million at Friday's closing price.

American disclosed the stock transactions by Parker and other top executives in filings with the Securities and Exchange Commission.

American and US Airways merged in December, and shares and options in each airline

Figure-34: Dashboard

Upon logging in, the user is sent to the dashboard where users' stocks are saved and presented in the above card form. The dashboard's search module has been changed. The dashboard's original search module was unintuitive — it contained an arrow which "pulled" the search text field down. The arrow contained no clear indication it was connected to the search query, so we replaced it with three search fields: Sector, Industry, and General Search. All three are clearly labeled, easily accessible by the user, and intuitive. This will cut user effort by at least one click per search.



Yahoo

Yahoo! Inc.
NASDAQ: YHOO - Mar 17 5:59 PM ET
39.11 +1.51 (4.02%)
After Hours: 39.11 +0.00 (0.00%)

Figure-35: Search result.

Upon searching a stock and hitting enter, the user will be presented with either a match or a list of relevant stocks. All stocks will be shown with company name, price, change in price, and price charts clearly displayed because they are the most relevant data for the user. Furthermore, the user will be able to add stocks to their list of stock cards by clicking on the stock's company name or its details and traverse back to the dashboard by clicking anywhere else, keeping user effort low.

6. Design of Tests

A good controller is efficient for the stability and functionality of the proposed model and application. One can compare it to the engine of a car. Therefore, an important aspect of the controller is to keep track of certain stocks. This test can also check many other smaller issues along its path like an odometer that keeps track of the speed of a car. If an incorrect output is received then we can trace it to the cause(s) (engine, tire-rotation, pressure on gas-pedal,other calibrations) of the fault. Similarly, if the tracking of a stock is incorrect, perhaps it was a search of the stocks in the database or the current external website(YahooFinance). This can be checked through creating more boolean values. This test case is focused on the search function and uses the track function to display its findings (Function requirement 4). It will use state testing, to determine what the history of the grabbed stocks was so that it did grab the right ones. This is because in history there might be similar data such as industry, sector, and keyword however there will also be a timestamp.

```
public class ControllerTest{

        void checktracking(){

        //1. Set up
        Controller one = new Controller(/*set parameters*/);
        bool value = false;
        //make sure stocks are in the repository

        //2. act
        string[] tickers = one.search(/*parameters*/);

        /*it should be greater than one at least*/
        if(length of tickers >= 1){
                string retrievedone = tickers[1];
                if(
                /*Compared the retrieved stock with its history(using the stock name)
and
                preform certain analysis to make sure search generated the current of
true
               stock(s) (Can be done through checking timestamps.*/ ){
                value = track[retrievedone];}
                }
                else
```

```
             value = false;

      //3.verify
      if(value == 0)
             cout<<"Could not track";
      else
             cout<<"Tracking was fine";

}
}
```

For the predictor, not only must the confidence level be accurate but also the decisions that are made. Therefore, based on the comparison of what each prediction models and confidence level provided it makes only logical sense that the two should be directly related. Also the decisions made should be related to what the predicted price of a stock is based on the current price. Therefore certain events and scenarios need to be checked. This is probably the most important test case. This is because it is in the closest proximity of the customer. This covers important functional requirements 4 & 5.

```
public class predictorTest{

      void validconfidencemaking{

      //1.SetUp
      bool isFine;
Predictor oracle = new Predictor();

      //2.Act
      if(no stocks are in database)
             if(oracle.calcConfidence !=0)
                    isFine = false;
      else{
      //required that some stocks are already in database
      predict();
      if(oracle.calcConfidence ==0)
             //check various models to see if that does make sense if so set isFine
to true
      else if(oracle.calcConfidence>0)
             //check various models to see if that does make sense  if so set isFine
to true

      else if(oracle.calcConfidence<0)
             //check various models to see if that does make sense  if so set isFine
to true

      else
             isFine = false;
}
//3. Verify
if(isFine==false)
      cout<<"Something is wrong";
else
      cout<<"Everything is fine";
```

```
}

bool validDecisionMaking{
//1.Setup
Predictor oracle = new Predictor();
double value = oracle.predict();
bool isFine;

//2.Act
if(oracle.predict > current price && oracle.makePredicton is buy)
       isFine = true;
else if(oracle.predict < current price && oracle.makePredicton is sell){
       isFine = true;
else if(oracle.predict > current price && oracle.makePredicton is do nothing){
       isFine = true;
else
       isFine = false;

//Verify
return isFine;
}
}
```

Boundary Testing can be performed for this too. Pass the confidence value thresholds as inputs to another test case and check what happens to the state and confidence as each prediction is sequentially entered.

Another important functional requirement - Req2 is tested by the case below. The regulation here is to make sure everything is up to do since we do not wish to derive any incorrect or out of date decisions for the customer. This test case is the first gatekeeper in that regulatory process.

```
class UpdaterChecker{

bool isUpdaterworking{

       1. SetUp
       Updater process = new Updater();
       //collect date of last known update from direct contact with Yahoo Finance
       int truedate = /*from external.
       bool isUptodate;

       2. Act
       if(truedate == process.retrievedate(/*specifiedticker*/)
              isUptodate == true;
       else
              isUptodate == false;
       3.Verify
       return isUptodate;
}
}
```

Integration Testing Strategy

The best one to utilize for this application is vertical integration. The most important reason is because it delivers a working testing platform quickly and time is of the essence. Time is precious here since time is money. Every second wasted could potentially cause customers to miss lucrative opportunities. Therefore when customers or programmers log tickets it should be answered quickly.

Testing Algorithms

To test algorithms that predict future stock prices we will simply wait and compare the predicted price to the actual price. We can further test these methods on past prices as well. The confidence value's will be calculated using a percent error between the predicted price and actual price and will be used to gauge an algorithms accuracy and effectiveness.

Testing Non-functional Requirements

Non-functional requirements that requires keeping track of time, can be tested easily by issuing a dummy variable and sleeping the program for that specified time, and checking the state of the program afterwards. Furthermore, third party browser extensions will be used to measure website loading times. Other tests such as gauging the ability of a user to intuitively utilize the system will be conducted through observing random users' inputs when told to perform a certain task. For example, a certain user might be told to perform a search and the number of correct/incorrect mouse clicks will be recorded.

7. Project Management and Plan of Work

7.1. Merging the Contributions from Individual Team Members

Once work was divided up between team members the majority of issues encountered were communication related. Although the modern era of technology allows for an almost abundant means of communication some of which including phone, texts, email, instant messaging, etc, they simply cannot replace face to face meetings. The use of technology, in this rare case, has begun to hinder our development when it came to assigning and meeting deadlines, reviewing team members work, and brainstorming new as well as improving old software ideas. For example, a team member might submit his/her assigned work only to later find out that most, if not all, of it is incomplete and unsatisfactory when judged by the rest of the team. To tackle issues like these we have created stricter deadlines and allowed for more face to face meetings throughout the week. This will give us more time to peer review work as well as better communicate as a group allowing for a surge of different viewpoints into each individual's task.

Further dilemmas faced were a lack of examples within the textbook for certain diagrams necessary in the report. We simply did not know how the diagram should be explained, mainly which points to emphasize and which to ignore. We have tackled this issue by using other recourses such as the internet to retrieve a myriad of further examples.

7.2. Project Coordination and Progress Report

So far the relational database has been established and use cases Search and Suggest implemented. Research on further use cases such as Predict and Notify, Update, and AddStock has been extensively done and are in the process of being implemented. A preliminary user interface has been created and we will be using the following weeks leading up to demo one to finalize our most important use cases as well as full UI. If timer permits, we will move on to implementing less priority use cases and if not, these will be saved for the second demo.

7.3. Plan of Work

We plan to have the use case of Search, Suggest, Predict and Notify, Update, and AddStock implemented within the next two weeks in time for demo one. We set a designated date of March 23 for a deadline for our individual parts of the demo so we can use the following week for integration. Once the demo is completed the following week will be used to implement lesser priority use cases Track, Login, RemoveTrackedStock, Register, and Logout. The following week will be used to implement the remaining use cases of EditUpdateTimer, EditPredictionTimer, Learn, and Support. Throughout this time period we will also be incrementally optimizing both the appearance and functionality of the user interface. Any remaining time will be used to optimize the system as a whole.

7.4. Breakdown of Responsibilities

Currently Mohammed and Vince are in charge of developing, coding, and testing the classes of Updater, Stock Retriever, and Stock Updater, or the data mining subsystem.

Robert, Robin, and Manoj are in charge of developing, coding, and testing the classes of Predictor, Timer, and Controller, mainly the data analysis subsystem.

Syedur and Peter are in charge of developing, coding, and testing the classes of Searcher, Logger, PageMaker, and Grapher.

We will determine the difficulty of each task assigned as we dig deeper into development and assign further classes and modules to those members that have easier tasks.

Integration and integration testing will be coordinated by all members since each sub group will be familiar with their own class. We feel this will be the most efficient method of integration.

8. References

1.1. Useful Information

1. Mathematical Models:
   http://www.vatsals.com/Essays/MachineLearningTechniquesforStockPrediction.pdf
   http://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average

2. Introduction to ARIMA
   http://people.duke.edu/~rnau/411arim.htm

3. Autoregressive integrated moving average
   http://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average

4. Average true range
   http://en.wikipedia.org/wiki/Average_true_range

5. Stock Market Prediction – Neural Networks toolbox - MATLAB
   http://www.breakyourhead.com/2013/03/stock-prediction-artificial-neural.html

6. Trend Forecasting in Capital Markets with Neural Networks in MatLab
   http://www.cvis.cz/eng/hlavni.php?stranka=novinky/clanek.php&id=69

7. Python: Accumulative Swing Index Mathematics and stock Indicators
   http://sentdex.com/sentiment-analysisbig-data-and-python-tutorials-algorithmic-trading/python-accumulative-swing-index-asi-mathematics-stock-indicators/

8. Average directional movement index
   http://en.wikipedia.org/wiki/Average_directional_movement_index