**CS 440**

**Fall 2014**

**Testing Report of Stock Buddy**

**Group 9**

**- BRESIA PRUDENTE**

**- DIBYAYAN DAS**

**- KARL CONSING**

**- TANIMA CHATTERJEE**

**Overview:**

This a documentation of the tests carried out on the software project created by Group 9(CS 440), Stock Buddy. The code began with code testing, followed by unit testing and then equivalence testing.

Code testing involved testing individual classes, header files and their codes.

Unit testing was carried out on all individual use cases.

Equivalence testing tested the system’s output on the equivalence class inputs.

Before we began testing, a test plan was carried out which involved preparing test specifications followed by a test report on corresponding test cases.

**CODE TESTING**

|  |  |  |  |
| --- | --- | --- | --- |
| **File Name** | **Description of File** | **Comments** | **Major Rework** |
| mainwindow.cpp | Contains login screen that prompts for username and password. | Function names need to be given significant names. | Declared a global instance for the username to be used consistently throughout the program.  Removed connection debugging code. |
| mainwindow.h | Contains classes and private slots for mainwindow.cpp, as well as a database connection to be accessed in all source files. | Private slots need to reflect any changes to significant function names.  Get rid of code that is being used for debugging. | Fixed a connection issue that was preventing all source files from being able to grab and parse queries. |
| summary.cpp | Displays to the user all of the purchases and sales made. | Give the functions significant names. | Got rid of unused commented code.  Removed connection debugging code. |
| summary.h | Contains classes and private slots for summary.cpp. | Private slots need to reflect any changes to significant function names. | Got rid of unused private slot functions.  Reindented entire code. |
| prompt.cpp | Displays a prompt to the user containing a choice that takes them to that page. | Function names need to have more significant names. | Added comments to briefly describe each function’s job. |
| prompt.h | Contains classes and private slots for prompt.cpp. | Private slots need to reflect any changes to significant function names. | Reindented entire code. |
| register.cpp | Allows user to register their information to be stored on the database. | Functions need to have significant names. | Removed connection debugging code.  Removed unused commented code. |
| register.h | Contains classes and private slots for register.cpp. | Private slots need to reflect any changes to significant function names. | Reindented entire code.  Removed unused private slot. |
| update.cpp | User can update existing information on the database. | Need to double check the query prompt messages. | Removed unused commented code and edited query error message. |
| update.h | Contains classes and private slots for update.cpp. | N/A | Reindented entire code. |
| welcome.cpp | Displays the stock history and the user’s investment history. | Update some function names with more significant ones. | Removed unused commented code and reindented entire code. |
| welcome.h | Contains classes and private slots for welcome.cpp. | If changes were made to function names in .cpp file, have it reflect in private slots also. | N/A |
| buy.cpp | Allows user to buy stocks. | Give significant names to the button functions. | Resolved issue with bought stocks reflected on the summary page by editing update query to use qstring instead of float.  Reindented entire code and also removed unused commented code. |
| buy.h | Contains classes and private slots for buy.cpp. | Private slots need to reflect any changes to significant function names. | N/A |
| receipt.cpp | Displays the user’s stock purchases and sales for the day. | Button functions should have more significant names. | Removed unused commented code as well as connection debugging code.  Also added some comments to make some parts of the code easier to understand. |
| receipt.h | Contains classes and private slots for receipt.cpp. | Private slots need to reflect any changes to significant function names. | N/A |
| sell.cpp | Allows user to sell stocks in their inventory. | Button functions should have more significant names. | Resolved issue with sold stocks reflected on the summary page by editing update query to use qstring instead of float.  Reindented entire code, removed unused commented code, and also got rid of debugging code. |
| sell.h | Contains classes and private slots for sell.cpp. | Private slots need to reflect any changes to significant function names. | N/A |

**UNIT TESTING**

**Test Specification:**

**Use Case 1:**

**Use Case Name:** Log In

**Actor:** Stock Buddy User

**Entry Condition:**

1. User enters username and password & clicks on ok OR

2. User clicks the register button OR

3. User clicks the update button

**Flow of Events:**

1. User enters username and password & clicks on ok OR

2. User clicks the register button OR

3. User clicks the update button

4. User is navigated to welcome page or prompted for incorrect username or

Password OR

5. Navigated to register page OR

6. Navigated to update page.

**Exit Condition:**

1. User is on the welcome page or login page.

2. User is on the registration page.

3. User is on the update details page.

**Test Case 1:**

**Test Case Identifier**: Login

**Test location:** C:\Users\sony\Desktop\StockBuddy

**Feature to be tested:**

1. Proper fetching of query () from database.

2. Proper navigation between main window & update/register/welcome

page.

**Feature Pass/Fail criteria:**

**Pass:** 1. If correct username/password => Welcome page

else Error Prompt.

2. Register page in 2 seconds if register button is pressed.

3. Update page in 2 seconds if update button is pressed.

**Means of control**: on\_PushButton1\_Clicked() method is called in MainWindow.cpp

through Form main window.

**Data:** 1. Login database

2. Table used in login.db: User\_info

Use Case 2:

**Use Case Name**: Register

**Actor:** Stock Buddy User

**Entry Condition**: User presses the Register button on login page.

**Flow of Events:** 1. User enters First Name.

2. User enters Last Name.

3. User enters Username.

4. User enters Email Id.

5. User enters Password.

6. User is prompted for successful Registration or an error if the Username

already exists or a field is left blank.

7. Navigated back to welcome page.

**Exit Condition:** 1. User is on the login page.

Test Case 2:

**Test Case Identifier:** Register

**Test location**: C:\Users\sony\Desktop\StockBuddy

**Feature to be tested**: 1. Proper fetching of query () from database.

2. Proper navigation between Register & Login

page.

**Feature Pass/Fail criteria:**

**Pass**: 1. If no duplicate Username is entered then prompt for successful

registration and navigate back to login page.

2. If duplicate Username is entered then prompt for duplicate

username.

3. If any field is left vacant, prompt for error.

4. When the user logs in again with the registered username and

password, he is navigated to welcome page.

**Means of control**: 1. on\_PushButton1\_Clicked() method is called in Register.cpp through

Form Register.

**Data:** 1. Login database

2. Table used in login.db: User\_info

**Test procedure**: 1. Fill in details and press Ok.

2. Fill in duplicate username and press Ok.

3. Leave a field blank and press Ok.

**Use Case 3:**

**Use Case Name**: Update

**Actor**: Stock Buddy User

**Entry Condition**: 1. User presses the Update button on login page.

**Flow of Events**: 1. User enters Username.

2. User updates Last Name.

3. User updates First Name.

4. User updates Email Id.

5. User updates Password.

6. User is prompted for successful Update or an error if a field is left blank.

7. Navigated back to welcome page.

**Exit Condition**: 1. User is on the login page.

**Test Case 3:**

**Test Case Identifier**: Update

**Test location**: C:\Users\sony\Desktop\StockBuddy

**Feature to be tested**: 1. Proper fetching of query () from database.

2. Proper navigation between Update & Login

page.

**Feature Pass/Fail criteria**:

**Pass**: 1. If no field is left blank then prompt for successful update and

navigate back to login page.

2. If any field is left vacant, prompt for error.

3. When the user logs in again with the username and new password (if

any), he is navigated to welcome page.

**Means of control**: on\_PushButton1\_Clicked() method is called in Update.cpp through

Form Update.

**Data**: 1. Login database

2. Table used in login.db: User\_info

**Test procedure**: 1. Fill in details and press Ok.

2. Leave a field blank and press Ok.

**Use Case 4:**

**Use Case Name**: Welcome

**Actor**: Stock Buddy User

**Entry Condition**: 1. User presses the OK button on login page

**Flow of Events**:

1. User Presses the refresh button and selects the company name from the

combo box in tab investment history or moves to the tab Current Stock

Rate.

2. User presses the refresh button and selects the company name from the

combo box current stock rate tab.

3. User presses the Next Button.

4. In Investment History tab, the user selects the company in which he has

invested through the dropdown combo box and views his corresponding

investment history.

5. In Current Stock Rate tab, the user selects the company he is interested in,

and views his corresponding Stock Rates.

6. User presses the Next button.

**Exit Condition**: User is on the prompt page.

**Test Case 4**:

**Test Case Identifier**: Welcome

**Test location**: C:\Users\sony\Desktop\StockBuddy

**Feature to be** **tested**: 1. Proper fetching of query from database.

2. Proper navigation between welcome and prompt page.

**Feature Pass/Fail criteria**:

**Pass**: 1.The stock details of the company selected from the combo box should

appear in the table view within 2 seconds of selection in Investment

History Tab.

2. The stock details of the company selected from the combo box

should appear in the table view within 2 seconds of selection in

Current Stock Rate Tab.

3. When the user presses the Next button, he should be navigated to the

prompt page.

**Means of control**: 1. on\_get\_currentstockrate\_Clicked() method is called in Update.cpp

through Form Update.

2. on\_PushButton2\_Clicked() method is called in Update.cpp through

Form Update.

3. on\_PushButton3\_Clicked() method is called in Update.cpp through

Form Update.

4. on\_comboBox\_currentIndexChanged(const QString &arg1) method is

called in Update.cpp through Form Update.

**Data**: 1. Login database

2. Table used in login.db: InvHistory, StockMarketData.

**Test procedure**:

In Current Stock Rate tab:

1. Select a company from the dropdown combo box.

2. Press get current stock rate button.

In Investment History Rate tab:

1. Select a company from the dropdown combo box.

2. Press get investment history button.

**Use Case 5**:

**Use Case Name**: Prompt

**Actor**: Stock Buddy User

**Entry Condition**: User presses the buy/sell button.

**Flow of Events**: User is navigated to the buy/sell page.

**Exit Condition**: User is on the buy/sell page.

**Test Case 5**:

**Test Case Identifier**: Prompt

**Test location**: C:\Users\sony\Desktop\StockBuddy

**Feature to be tested**: 1. Proper navigation between prompt and buy/sell page.

**Feature Pass/Fail criteria**:

Pass: 1. When the user presses the buy/sell button, he should be navigated to

the buy/sell page.

**Means of control**: 1. on\_PushButton1\_Clicked() method is called in Update.cpp through

Form Update.

2. on\_PushButton2\_Clicked() method is called in Update.cpp through

Form Update.

**Data**: None.

**Test procedure**:

1. Press the Buy button.
2. Press the Sell button.

**Use Case 6**:

**Use Case Name**: Buy

**Actor**: Stock Buddy User

**Entry Condition**: 1. User presses the buy button on the prompt page.

**Flow of Events**: 1. User presses the Refresh button.

2. User double clicks on the company name he wants to buy stocks from.

3. User enters the number of stocks he wishes to buy.

4. User presses the Get Total button.

5. User presses the next button.

**Exit Condition**: 1. User is on the Summary page.

**Test Case 6**:

**Test Case Identifier**: Buy

**Test location**: C:\Users\sony\Desktop\StockBuddy

**Feature to be tested**: 1. Proper fetching of query from database.

2. Proper calculation of total cost of the stocks.

3. Proper navigation between Buy & Summary page.

**Feature Pass/Fail criteria**:

**Pass**: 1. The stock details of the company selected from the combo box

should appear in the table view within 2 seconds of selection.

1. The name of the company the user selects should appear in the line

edit within a second.

1. When the user presses the Get Total button, the correct cost should

be calculated.

**Means of control**: 1. on\_pushButton\_clicked() method is called in Buy.cpp through

Form Buy.

2. on\_pushButton\_2\_clicked() method is called in Buy.cpp through

Form Buy.

1. on\_tableView\_activated()method is called in Buy.cpp through

Form Buy.

**Data**: 1. Login database

2. Table used in login.db: StockMarketData

**Test procedure**: 1. Select a company from the table.

2. Enter the number of stocks to be bought.

3. Press get total.

4. Press Next button.

**Use Case 7**:

**Use Case Name**: Sell

**Actor**: Stock Buddy User

**Entry Condition**: 1. User presses the sell button on the prompt page.

**Flow of Events**: 1. User presses the Refresh button.

2. User double clicks on the company name he wants to sell stocks from.

3. User enters the number of stocks he wishes to sell.

4. User presses the Get Total button.

5. User presses the next button.

**Exit Condition**: 1. User is on the Summary page.

**Test Case 7**:

**Test Case Identifier**: Sell

**Test location**: C:\Users\sony\Desktop\StockBuddy

**Feature to be tested**: 1. Proper fetching of query from database.

2. Proper calculation of total cost of the stocks.

3. Proper navigation between Sell & Summary page.

**Feature Pass/Fail criteria**:

**Pass**: 1. The stock details of the company selected from the combo box

should appear in the table view within 2 seconds of selection.

2. The name of the company the user selects should appear in the line

edit within a second.

3. When the user presses the Get Total button, the correct cost should

be calculated.

**Means of control**: 1. on\_pushButton\_clicked() method is called in Sell.cpp through

Form Sell.

2. on\_pushButton\_2\_clicked() method is called in Sell.cpp through

Form Sell.

3. on\_tableView\_activated() method is called in Sell.cpp through

Form Sell.

**Data:** 1. Login database

2. Table used in login.db: InvHistory.

**Test procedure**: 1. Select a company from the table.

2. Enter the number of stocks to be sold.

3. Press get total.

4. Press Next button.

**Use Case 8:**

**Use Case Name**: Summary

**Actor**: Stock Buddy User

**Entry Condition**: 1. User presses the next button on the buy/sell page.

**Flow of Events**: 1. User presses the Refresh button.

2. User presses the View Receipt button.

3. User presses the Exit button.

**Exit Condition**: 1. User Exits Stock Buddy.

**Test Case 8**:

**Test Case Identifier**: Summary

**Test location**: C:\Users\sony\Desktop\StockBuddy

**Feature to be tested**: 1. Proper fetching of query from database.

2. Proper update of the stocks sold or bought in the database.

**Feature Pass/Fail criteria**:

**Pass**: 1. The page reflects the changes in the user’s investment history due to

the recent transaction.

2. The Receipt appears within 2 seconds of pressing the View Receipt

button.

3. The receipt contains details of the current state and balance in the user’s Stock Buddy account.

**Means of control**: 1. on\_viewReceipt\_clicked() method is called in Summary.cpp

through Form Summary.

2. on\_pushButton\_2\_clicked() method is called in Summary.cpp

through Form Summary.

3. on\_pushButton\_2\_clicked() method is called in Summary.cpp

through Form Summary.

**Data**: 1. Login database

2. Table used in login.db: InvHistory.

**Test procedure**: 1. Press the Refresh Button.

2. Press the View Receipt Button.

3. Press Exit button.

Test Report Summary:

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Oracle | | Comment |
| Test Case1 | Expected | Actual | Pass |
| 1. Navigated to Welcome /Register/Update Page. 2. Prompt on incorrect Username/Password | 1. As Expected 2. As Expected |
| Test Case 2 | 1. Prompt on successful registration 2. Prompt on duplicate username 3. Prompt on field left blank. | 1. As Expected 2. Accepts duplicate username 3. As expected | a. Pass  b. Fail  c. Pass |
| Test Case 3 | a. Prompt on successful update  b. Prompt on field left blank | a. As Expected  b. As Expected | a. Pass  b. Pass |
| Test Case 4 | a. Stock details of the company selected via combo box in table view.  b. Correct investment history of the user as expected  c. Navigated to prompt page on pressing OK | a. As Expected  b. As Expected  c. As expected | a. Pass  b. Pass  c. Pass |
| Test Case 5 | User is navigated to sell or buy page. | As expected | Pass |
| Test Case 6 | a. The name of the company the user double clicks on appears in line edit.  b. Accurate total is calculated.  c. User is navigated to the summary page. | a. As Expected  b. As Expected  c. As Expected | a. Pass  b. Pass  c. Pass |
| Test Case 7 | a. The name of the company the user double clicks on appears in line edit.  b. Accurate total is calculated.  c. User is navigated to the summary page. | a. As Expected  b. As Expected  c. As expected | a. Pass  b. Pass  c. Pass |
| Test Case 8 | a. The changes in the user’s investment history due to the recent transaction is reflected.  b. The user is shown the receipt when the receipt button is pressed.  c. The application exits on pressing the exit button. | a. As Expected.  b. As Expected  c. As Expected | a. Pass  b. Pass  c. Pass |

EQUIVALENCE TESTING

1. MainWindow.cpp

|  |  |  |
| --- | --- | --- |
| **Equivalence class** | **Value for username** | **Value for password** |
| **Correct username/password** | tchatt2 | abc2 |
| **Incorrect username/password** | tchatt2 | xyz123 |

1. Register.cpp

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Equivalence class** | **First Name** | **Last Name** | **Username** | **Password** | **email** |
| **Pre-existing First Name** | Tanima | Banerjee | tchatt3 | abc3 | tchatt3@uic.edu |
| **Pre-existing Last Name** | Tia | Chatterjee | tia3 | Xyz1 | Tia3@uic.edu |
| **Pre-existing Username** | Rupa | Bose | tchatt2 | Bh2hj5 | Rbose2@uic.edu |
| **Pre-existing Password** | Esha | Das | Edas | abc2 | Edas4@uic.edu |
| **Pre-existing email** | Mahak | Garg | Mgarg12 | Bn5thf | tchatt2@uic.edu |
| **Blank First Name** |  | Das | Edas | xyz | Edas4@uic.edu |
| **Blank Last Name** | Mahak |  | Mgarg12 | Gh09p | Mgarg3@uic.edu |
| **Blank username** | Rupa | Bose |  | Jk9yt5 | Rbose2@uic.edu |
| **Blank password** | Tia | Chatterjee | tia3 |  | Tia3@uic.edu |
| **Blank email** | Tia | Chatterjee | tia3 | Bla123 |  |

1. Sell.cpp

|  |  |  |
| --- | --- | --- |
| **Equivalence class** | **No of Stocks** | **Company Name** |
| **Correct Company Name** | 100 | AAPL |
| **Incorrect Company Name** | 100 | UPS |
| **Correct No. of Stocks** | 100 | AAPL |
| **Incorrect No. of Stocks** | 100000000 | AAPL |
| Username: tchatt2 | | |

1. Buy.cpp

|  |  |  |
| --- | --- | --- |
| **Equivalence class** | **No of Stocks** | **Company Name** |
| **Correct Company Name** | 100 | AAPL |
| **Incorrect Company Name** | 100 | UPS |
| **Correct No. of Stocks** | 100 | AAPL |
| **Incorrect No. of Stocks** | 100000000 | AAPL |
|  | | |