Test Plan Specification



**Table of Contents**

[**1.0 Introduction 3**](#_heading=h.gjdgxs)

[1.1 Goals and Objectives 3](#_heading=h.30j0zll)

[1.2 Constraints 3](#_heading=h.1fob9te)

[**2.0 Testing Methods 3**](#_heading=h.3znysh7)

[2.1 Unit Testing 4](#_heading=h.2et92p0)

[2.2 Integration Testing 4](#_heading=h.tyjcwt)

[2.3 System Testing 4](#_heading=h.3dy6vkm)

[**3.0 Testing Strategy 6**](#_heading=h.17dp8vu)

[3.1 Test Items 6](#_heading=h.3rdcrjn)

[3.2 Test Descriptions 6](#_heading=h.26in1rg)

[**4.0 Software Testing 11**](#_heading=h.lnxbz9)

[**5.0 Response To Failure 12**](#_heading=h.35nkun2)

[**6.0 Revision Log 13**](#_heading=h.1ksv4uv)

# **1.0 Introduction**

Code Craft has been given a task to create an application for computerized stock trading, as well as create our algorithm that will be used in the application. The application will give the user points on which stocks to invest and sell. This document will outline how Code Craft will test our application.

## 1.1 Goals and Objectives

The purpose of the Test Plan document is to list out the components of the application that need to be tested, give descriptions of the components, and list how the test results could affect our product. The overall objective is for this document to give clear and detailed descriptions of the testing methods, to give the intended user an understanding of how the application was developed and tested before it was given to them. Another objective is to reflect any changes that were made during development.

## 1.2 Constraints

Some constraints with testing that could arise are:

* The application needs to work for all trading platforms, even though Alpaca is the main platform that will be used, compatibility for all platforms should be tested as well.
* **Time**: Most members of the team are not familiar with creating an API in Python as well as stock trading. As well as creating an algorithm to go with the application.
* **Experience**: Due to the unfamiliarity in Python, APIs, and stock trading, this will result with most of our time in and out of meetings researching these topics to better understand the project.

# **2.0 Testing Methods**

## 2.1 Unit Testing

Unit testing involves isolating the components of the software and testing each piece so that it works correctly on its own. Unit tests are run by themselves before the components of the software are put back together and tested as one product. Seeing as the product will be made up of several different programs, unit testing will be important in making sure every single component of the product is working independently. We will look to use Visual Studio Code for unit testing since our product will be written in mainly Python. Testing will allow us to see if changes and functions are being presented to the application accurately.

## 2.2 Integration Testing

Integration testing involves combining two or more components that have already been tested (via unit testing) and testing the interface between them. The goal of integration testing is to continuously combine the different parts of the product and test them until you end up with one model. Integration testing points out problems that occur when different components of the software are put together. A test plan that calls for the testing of every component ensures how viable the unit is before combining it with other units. Errors discovered when combining units are likely related to the interface between units. This method reduces the number of possibilities to a far simpler level of analysis.

## 2.3 System Testing

System testing is a level of software testing where complete and integrated software is tested. The purpose of this test is to verify that the product that is to be delivered to the client meets the specifications from the functional and design specifications. It needs to explore both non-functional requirements and functional requirements.

# 

# 

# 

# **3.0 Testing Strategy**

## 3.1 Test Items

* Algorithm
* API
* Button Configuration

## 3.2 Test Descriptions

The following section describes the various components of the product that must be tested to ensure the product is working as intended. These tests include descriptions and instructions for how to execute them as well as the expected outcome when run successfully. In each test, we will define the component that is being tested, the name of the test, a brief test description, and instructions on how to run the test.

Test Name: **TrendTrade1**

Test Description: **We want the bot to view the percentage of the stocks and see if they will decrease or increase by a certain percent so it can sell or buy whichever number hits first.**

Test Instructions:

**def sell\_stocks\_if\_price\_change():**

**orders = get\_orders()**

**current\_time = time.localtime()**

**for order in orders:**

**symbol = order['symbol']**

**try:**

**current\_price = get\_current\_price(symbol)**

**purchase\_price = get\_purchase\_price(symbol)**

**percentage\_change = ((current\_price - purchase\_price) / purchase\_price) \* 100**

**if abs(percentage\_change) >= 0.3 or is\_time\_to\_sell(current\_time):**

**create\_order(symbol=symbol, qty=order['qty'], side='sell', type='market', time\_in\_force='day')**

**except Exception as e:**

**log\_error(e)**

**deactivate\_bot()**

**def is\_time\_to\_sell(current\_time):**

**# Check if it's 7:12PM**

**return current\_time.tm\_hour == 19 and current\_time.tm\_min == 12**

**def log\_error(error):**

**# Log the error to a file or print it for debugging**

**print(f"An error occurred: {error}")**

**# Other helper functions (get\_orders, get\_current\_price, get\_purchase\_price, create\_order, deactivate\_bot) should be defined elsewhere.**

Expected Outcome: **Trade stocks when percentage increases by 0.3% and decreases by 0.3%**

Date: **04/17/2024**

Results: **FAIL**

Test Name: **CheckEveryMinuteSell**

Test Description: **We wanted to test the check clock function of the sell-by-close method.**

Test Instructions:

**def schedule\_sell\_before\_market\_close():**

**while True:**

**# Check if it is 15 minutes before market close**

**print('Checking Time')**

**current\_time = time.localtime()**

**if current\_time.tm\_hour == 15 and current\_time.tm\_min == 45:**

**sell\_all\_stocks()**

**break**

**else:**

**time.sleep(60)**

Expected Outcome: **The bot will check every minute if the designated sell-time has been reached.**

Test Name: **SellBeforeClose**

Test Description: **We wanted our algorithm to sell all stocks that have been ordered**

**for the day to be sold minutes before the market closes.**

Test Instructions:

**def schedule\_sell\_before\_market\_close():**

**while True:**

**# Check if it is 15 minutes before market close**

**print('Checking Time')**

**current\_time = time.localtime()**

**if current\_time.tm\_hour == 15 and current\_time.tm\_min == 45:**

**sell\_all\_stocks()**

**break**

**else:**

**time.sleep(60)**

Expected Outcome: **All purchased stocks will be sold at 3:45pm before the**

**market closes.**

Test Name: **SellAllStocksQuantity**

Test Description: **We wanted to test our revised code that sells all stocks because of previous errors. Previously the algorithm would sell a particular stock an abundance of times.**

Test Instructions:

**def sell\_all\_stocks():**

**stocks\_to\_sell = ['AAPL', 'TSLA', 'AMZN', 'GOOGL', 'META', 'MSFT']**

**for stock in stocks\_to\_sell:**

**create\_order(symbol=stock, qty=1, side='sell', type='market', time\_in\_force='day')**

Expected Outcome: **All stocks in the stocks\_to\_sell list will be sold in the right quantity when the function activates.**

Date: **04/09/2024**

Results: **PASS**

Test Name: **ConfirmedStocksList**

Test Description: **We wanted to test our revised code that sells all stocks because of previous errors. Previously the algorithm wouldn’t sell all stocks listed.**

Test Instructions:

**def sell\_all\_stocks():**

**stocks\_to\_sell = ['AAPL', 'TSLA', 'AMZN', 'GOOGL', 'META', 'MSFT']**

**for stock in stocks\_to\_sell:**

**create\_order(symbol=stock, qty=1, side='sell', type='market', time\_in\_force='day')**

Expected Outcome: **All stocks listed will be sold when the algorithm activates.**

Test Name: **NewOrders1**

Test Description: **Only show the orders placed that day on congrats screen**

Test Instructions:

**from datetime import date**

**NEW\_ORDERS = ‘{}?status=filled.format(ORDERS\_URL)**

def congratScreen():

//code regarding screen sizing, button placement, and table placement is before this

def get\_neworders():

r=requests.get(NEW\_ORDERS, headers=HEADERS)

return json.loads(r.content)

response = get\_neworders

**today = date.today()**

**if (filled\_at == today)**

for order in response:

symbol = order[‘symbol’]

qty= order[‘qty’]

side= order[‘side’]

status= order[‘status’]

filled\_at = order[filled\_at]

table.insert(‘’,’end’,values(symbol, qty, side, status))

Expected Outcome: **The table on the congrats screen will only show orders placed that day**

Name: **CenteredTrade**

Test Description: **Get the trade confirmation to be centered within the screen**

Test Instructions:

**empty = tk.Label(root, text=' \n ', width=10)**

**empty.grid(column=5, row=0)**

**# Display the user's cash amount**

**cash = get\_cash()**

**cashAmount = tk.Label(root, text=f'You currently have: ${cash}', font=('TkDefaultFont', 18, 'bold'))**

**cashAmount.grid(column=6, row=3)**

**genmess = tk.Label(root, text='Currently buying one share at market price using: day time in force ',font=('TkDefaultFont', 18, 'bold'))**

**genmess.grid(column=6, row=4)**

**empty = tk.Label(root, text=' \n ', width=10)**

**empty.grid(column=6, row=5)**

**[...]**

**# AAPL**

**AAPLprice = get\_tradePrice(AAPLTRADE\_PRICE)**

**buymess = tk.Label(root, text=f'AAPL at ${(AAPLprice)}',font=('TkDefaultFont', 18, 'bold'))**

**buymess.grid(column=6, row=7)**

**[...]**

Expected Outcome: **The information will be centered on the screen**

Date: **4/22/2024**

Result: **PASS**

Test Name: **BUYButton**

Test Description: **Show buy button is properly confidured and displaying stocks**

Test Instructions:

**# activate\_bot, orderingGui, welcome.destroy**

**buyBtn = Button(left, text='BUY Stocks', bd='5', command=lambda:[orderingGUI(), welcome.destroy(), activate\_bot()],font=('TkDefaultFont', 18))**

**buyBtn.pack(pady=40)**

Expected Outcome: **The buy button**

Test Name: **SellButton**

Test Description: **Show sell button is properly confidured and displaying stocks**

Test Instructions:

**# activate\_bot, orderingGui, welcome.destroy**

**btn\_deactivate = Button(left, text='SELL Stocks', bd='5', command=lambda:[deactivate\_bot(), congratScreen()], font=('TkDefaultFont', 18))**

**btn\_deactivate.pack(pady=40)**

# **4.0 Software Testing**

All software testing will be done locally on the personal computers of team members and/or the computers in the lab of S&T 127. When using the computers located in the lab of S&T 127, will be logged in on the admin account to be able to pip and puppet install using the terminal to effectively run the software features. These tests will be performed using the Virtual Studios IDE compiler and debugger, and corrections will be made to fix discovered issues as they appear.

# **5.0 Response To Failure**

If any unresolved issues or failed tests arise, Profit Prophets will look into all involved components to resolve them. During testing, if a fail were to occur, it will be noted.

# **6.0 Revision Log**

| Revision | By | Date | Description |
| --- | --- | --- | --- |
| 1.0 | Profit Prophets | 09/09/2024 | Project Plan for Iteration 1. |
| 2.0 | Profit Prophets | 10/01/2024 | Project Plan for Iteration 2. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |