***Cryptocurrency Ticker Application***



***Testing and Inspection Report***

***Prepared by***

***Ahmed Khan, Edgar Martinez, Rameez Baig, Zakee Jabbar***

***for use in* CS 440**

**at the**

**University of Illinois Chicago**

**November 2018**

**Table of Contents**

[I Project Description 4](#_Toc531470821)

[1 Project Overview 4](#_Toc531470822)

[2 Project Domain 4](#_Toc531470823)

[3 Relationship to Other Documents 4](#_Toc531470824)

[4 Naming Conventions and Definitions*.* 4](#_Toc531470825)

[4a Definitions of Key Terms 4](#_Toc531470826)

[4b UML and Other Notation Used in This Document 5](#_Toc531470827)

[4c Data Dictionary for Any Included Models 5](#_Toc531470828)

[II Testing 5](#_Toc531470829)

[5 Items to be Tested 5](#_Toc531470830)

[6 Test Specifications 6](#_Toc531470831)

[7 Test Results 12](#_Toc531470832)

[8 Regression Testing 15](#_Toc531470833)

[III Inspection 15](#_Toc531470834)

[9 Items to be Inspected 15](#_Toc531470835)

[10 Inspection Procedures 15](#_Toc531470836)

[11 Inspection Results 16](#_Toc531470837)

[IV Recommendations and Conclusions 19](#_Toc531470838)

[V Project Issues 19](#_Toc531470839)

[12 Open Issues 19](#_Toc531470840)

[13 Waiting Room 19](#_Toc531470841)

[14 Ideas for Solutions 19](#_Toc531470842)

[15 Project Retrospective 20](#_Toc531470843)

[VI Glossary 20](#_Toc531470844)

[VII References / Bibliography 21](#_Toc531470845)

# Project Description

## Project Overview

The product is a desktop application that serves as a means to help the user understand the current outlook on various cryptocurrencies on the current market. The product allows the user to access various information about a certain cryptocurrency all on one page. The user can switch between the different supported cryptocurrencies, including Bitcoin, Litecoin, Ethereum, and many more. The user can then choose what currency the user wants to see the data in: US Dollar, British Pound, and Australian Dollar are just some of the supported currencies. The user also sees a graph that shows the average price of the currently selected cryptocurrency for the past 24 hours but has the option to change it to the past month or even all time. Finally, the user is able to see relevant news articles relating to the currently selected cryptocurrency.

## Project Domain

The application can be run on any machine that supports the Java Development Kit and the Java Runtime Environment. The application must have an internet connection since the data is pulled through multiple external application program interface through GET requests.

## Relationship to Other Documents

This document serves as an overview that provides sufficient information about the product and its domain in order to put the materials being tested and inspected into proper context. This document stands as supplemental to and should not be considered independent of outside knowledge of the specified currencies. The application is user friendly, but some background knowledge is necessary in order to understand the full scope of the product

## Naming Conventions and Definitions*.*

### Definitions of Key Terms

Below is the list of some of the most important words which include acronyms and abbreviations.

Ask: ask is the lowest price at which you can buy

Bid: Bid is the highest price at which you can sell;

Last: the price at which the last transaction went through at

High: Means the highest price in a given period of time.

Low: Means the lowest price in a given period of time

Open:  the time at which people can begin trading on a particular exchange

Averages: average price-per-share during a specific period of time

Volume: Volume is the number of shares traded

### UML and Other Notation Used in This Document

This document generally follows the Version 2.0 OMG UML standard, as described by Fowler in [1]. Any exceptions are noted where used.

### Data Dictionary for Any Included Models

API Call = base URL + indices + global + ticker + price symbol

All data received from the API must be parsed specifically from JSON through the mentioned implementations. See source code for more details.

# Testing

## Items to be Tested

1. Backend
   1. API Calls
      1. #1 – API Response
         1. Covers cryptocurrency data, news articles, and currency data
   2. Persistent Data
      1. #2 – Read Write to CSV File
         1. Focuses on the data being written an read. Doesn’t test the functionality of the CSV Reader and Writer classes as those have been tested in another project.
2. Frontend
   1. Persistent Data
      1. #3 – Last Selected Items
         1. Covers having the right cryptocurrency, currency, and the time period selected
   2. Data Loading and Displaying
      1. #4 – News Article Opening
         1. Covers opening of news article in default browser.
      2. #5 – Refresh Button
         1. Covers having all the right components being updated on the refresh button
      3. #6 – Currency Changed
         1. Covers having all the right components being updated when the currency is changed.
      4. #7 – Cryptocurrency Changed
         1. Covers having all the right components being updated when the cryptocurrency is changed.
      5. #8 – Graph Changed
         1. Covers having all the right components being updated when the graph tab is changed.
      6. #9 – Search Filter
         1. Covers having the proper cryptocurrencies being shown when a query is typed into the search box.

## Test Specifications

**ID #1 - API Response**

**Description:** Response from Application Program Interface

**Items covered by this test:** Price data, historical data, currency data, response data

**Environmental needs:** Internet connection, Java Development Kit, Java Runtime Environment

**Intercase Dependencies:** NA

**Test Procedures:**

* Create the proper URL needed to create request
* Call the function and pass the URL to it
* Verify data

**Input Specification:** The URL desired, mainly that follows the pattern under the data dictionary.

**Output Specifications:** A properly formatted JSON object containing all necessary fields.

**Pass/Fail Criteria:**

* Pass: If the response contains all expected data
* Pass: If the product handles missing data elegantly
* Fail: if the response does not contain all expected data
* Fail: If the product does not handle missing data and throws exception

**ID #2 - Read Write to CSV File**

**Description:** Persistent Data

**Items covered by this test:** Crypto type, currency type, time period

**Environmental needs:** Java Development Kit, Java Runtime Environment

**Intercase Dependencies:** NA

**Test Procedures:**

* Start the application
* Change the crypto type
* Change the currency type
* Change the time period type
* Close the Application
* Launch the Application
* Repeat twice

**Input Specification:** The currently selected price symbol, crypto symbol, and time period

**Output Specifications:** If initial launch, creates data.csv file that stores all data. On relaunch, must successfully read and store data from file.

**Pass/Fail Criteria:**

* Pass: Creates data.csv file, opens file, writes headers to file, writes data to file, closes file, opens file, reads from file, saves data from file
* Pass: Opens file, writes to file, closes file, opens file, reads from file, saves data from file
* Fail: Can’t create file
* Fail: Can’t read from file
* Fail: Writes wrong data to file
* Fail: Reads and saves wrong data from file

**ID #3 - Last Selected Items**

**Description:** Application loads with last selected currency

**Items covered by this test:** Last selected currency, last selected graph, last selected cryptocurrency

**Environmental needs:** Java Development Kit, Java Runtime Environment

**Intercase Dependencies:** NA

**Test Procedures:**

* Open Application
* Select Currency
* Select Time Period, Currency, Cryptocurrency
* Close Application
* Reopen Application
* Check For Last Selected

**Input Specification:** The currency type, the time period

**Output Specifications:** On relaunch of the application, it should load the users last selections

**Pass/Fail Criteria:**

* Pass: Application loads with parts of the last saved preferences
* Fail: Application loads with parts of the last saved preferences
* Fail: Application loads with none of the last saved preferences

**ID #4 - News Article Opening**

**Description:** Testing the news portion of the Application

**Items covered by this test:** News Window

**Environmental needs:** Java Development Kit, Java Runtime Environment

**Intercase Dependencies:** NA

**Test Procedures:**

* Open Application
* Launch Article From News Section
* Browser Launches Relative News Article

**Input Specification:** Selection of Article

**Output Specifications:** Crypto related article should automatically open within the default browser

**Pass/Fail Criteria:**

* Pass: Application launches correct news article in default browser
* Fail: Application launches incorrect article in the browser
* Fail: Application never triggers default browser to launch

**ID #5 – Refresh Button**

**Description:** All applications features are updated when the refresh

**Items covered by this test:** Whole application window

**Environmental needs:** Java Development Kit, Java Runtime Environment

**Intercase Dependencies:** NA

**Test Procedures:**

* Open application
* Hit refresh button
* Verify all components were updated

**Input Specification:** Clicking of refresh button.

**Output Specifications:** Price information, quick information, converted prices, news articles, graph, and last updated time is updated.

**Pass/Fail Criteria:**

* Pass: All new data is pulled and updated on the UI.
* Fail: All new data is not pulled.
* Fail: Pulled data is not reflected in the UI.

**ID #6 – Currency Changed**

**Description:** All currency related items are updated when the user selects a different currency.

**Items covered by this test:** Top info bar, quick information, graph, converted prices.

**Environmental needs:** Java Development Kit, Java Runtime Environment

**Intercase Dependencies:** NA

**Test Procedures:**

* Open application
* Select different currency from dropdown
* Verify all components were updated

**Input Specification:** Select currency from dropdown

**Output Specifications:** Price information on top bar, quick information, converted prices, graph, and last update time are updated.

**Pass/Fail Criteria:**

* Pass: All new data is pulled and updated on the UI.
* Fail: All new data is not pulled.
* Fail: Pulled data is not reflected in the UI.

**ID #7 – Cryptocurrency Changed**

**Description:** All cryptocurrency related items are updated when the user selects a different currency.

**Items covered by this test:** Top info bar, quick information, converted prices, graph.

**Environmental needs:** Java Development Kit, Java Runtime Environment

**Intercase Dependencies:** NA

**Test Procedures:**

* Open application
* Select different cryptocurrency from list
* Verify all components were updated

**Input Specification:** Select cryptocurrency from list

**Output Specifications:** Price information on top bar, quick information, converted prices, graph, and last update time are updated. The new cryptocurrency selected is highlighted.

**Pass/Fail Criteria:**

* Pass: All new data is pulled and updated on the UI.
* Fail: All new data is not pulled.
* Fail: Pulled data is not reflected in the UI.

**ID #8 – Graph Changed**

**Description:** The graph view is updated when a different graph tab is selected.

**Items covered by this test:** The graph in the middle.

**Environmental needs:** Java Development Kit, Java Runtime Environment

**Intercase Dependencies:** NA

**Test Procedures:**

* Open application
* Select different graph tab
* Verify the graph view changed

**Input Specification:** The selected graph tab is changed

**Output Specifications:** The graph is changed to display the selected time frame.

**Pass/Fail Criteria:**

* Pass: All new data is pulled and displayed in the UI.
* Fail: All new data is not pulled.
* Fail: Pulled data is not reflected in the UI.

**ID #9 – Search Filter**

**Description:** The list of cryptocurrencies is updated to reflect the input query

**Items covered by this test:** List of cryptocurrencies and search bar.

**Environmental needs:** Java Development Kit, Java Runtime Environment

**Intercase Dependencies:** NA

**Test Procedures:**

* Open application
* Type input query in search bar
* Verify results match input query

**Input Specification:** Type query in search bar

**Output Specifications:** List is updated with matching search query

**Pass/Fail Criteria:**

* Pass: Only matching cryptocurrencies are shown.
* Fail: Unmatched cryptocurrencies are shown.

## Test Results

**ID #1 - API Response**

**Date(s) of Execution:** 11/28/2018

**Staff conducting tests:** Ahmed Khan

**Expected Results:** Handle missing data properly. No exceptions thrown

**Actual Results:** Handled missing data properly. No exceptions were thrown.

**Test Status:** Pass

**ID #2 - Read Write to CSV**

**Date(s) of Execution:** 11/28/2018

**Staff conducting tests:** Ahmed Khan

**Expected Results:** New file created if first launch. Otherwise file updated with new user selections.

**Actual Results:** New file created when application ran the first time. On second run file was updated based on user selection.

**Test Status:** Pass

**ID #3 - Last Selected**

**Date(s) of Execution:** 11/26/2018

**Staff conducting tests:** Rameez Baig

**Expected Results:** Application should relaunch with all last saved preferences.

**Actual Results:** Application relaunched with all last saved preferences**.**

**Test Status:** Pass

**ID #4 - News Article**

**Date(s) of Execution:** 11/26/2018

**Staff conducting tests:** Rameez Baig

**Expected Results:** Application launches correct article within the default browser

**Actual Results:** Application launches correct article within the default browser

**Test Status:** Pass

**ID #5 – Refresh Button**

**Date(s) of Execution:** 11/27/2018

**Staff conducting tests:** Zakee Jabbar

**Expected Results:** All components should update properly as defined in the test

**Actual Results:** All components updated properly as defined in the test.

**Test Status:** Pass

**Test Status:** Pass

**ID #6 – Currency Changed**

**Date(s) of Execution:** 11/27/2018

**Staff conducting tests:** Zakee Jabbar

**Expected Results:** All components should update properly as defined in the test

**Actual Results:** All components updated properly as defined in the test.

**Test Status:** Pass

**Test Status:** Pass

**ID #7 – Cryptocurrency Changed**

**Date(s) of Execution:** 11/27/2018

**Staff conducting tests:** Zakee Jabbar

**Expected Results:** All components should update properly as defined in the test

**Actual Results:** All components updated properly as defined in the test.

**Test Status:** Pass

**Test Status:** Pass

**ID #8 – Graph Changed**

**Date(s) of Execution:** 11/27/2018

**Staff conducting tests:** Zakee Jabbar

**Expected Results:** All components should update properly as defined in the test

**Actual Results:** All components updated properly as defined in the test.

**Test Status:** Pass

**Test Status:** Pass

**ID #9 – Search Filter**

**Date(s) of Execution:** 11/27/2018

**Staff conducting tests:** Zakee Jabbar

**Expected Results:** Only cryptocurrencies matching the search query should be updated

**Actual Results:** Cryptocurrencies matching the search query are updated.

**Test Status:** Pass

**Test Status:** Pass

## Regression Testing

When any new features are added all tests should be run again and ensure that they pass. This is done to make sure that the new feature hasn’t broken any of the old functionality.

# Inspection

## Items to be Inspected

The following pieces of code were submitted by the following people to the rest of the group to be inspected.

1. Ahmed Khan: PersistentData.java
2. Edgar Martinez: Crypto.java, News articles parts
3. Rameez Baig: Controller.java, News articles parts
4. Zakee Jabbar: Controller.java, Graphing and Updating Charts

## Inspection Procedures

The following checklist was used to when doing the inspections:

<https://dzone.com/articles/java-code-review-checklist>. [2]

There were couple meetings help during the inspection process. At the first meeting all members shared and explained their pieces of code that were to be inspected by the other members. After the meeting all members inspected the code using the checklist in their own time. The second meeting was used as a way to discuss the results of the inspections. All members took turns on the information they discovered to the member that submitted that specific code. After the second meeting follow up questions were answered electronically via a group chat.

## Inspection Results

1. Ahmed Khan: PersistentData.java
   1. Edgar Martinez’s Inspection: (11/26/18, 5:00 pm)
      1. Overall the code was written in a manner that is easy to read which allows the logic to also be easily followed. Functions had meaningful names and performed one task. There was also a good amount of comments that explained what the purpose of the code. Only issue, which wasn’t major was that there are some unused getter in the code that could have been deleted as they aren’t used.
   2. Rameez Baig’s Inspection: (11/27/18, 12:00 pm)
      1. Code is very readable and well-spaced out. Comments are very useful and allow the user to easily tell what is going on. The two helper classes help in hiding how all the csv reading and writing is going on. The only suggestion is that the local variable names to be more descriptive as it is a bit hard to tell what they exactly represent. Some field are represented by numbers, and it is unclear what that means. More comments need to be added or more descriptive names need to be used.
   3. Zakee Jabbar’s Inspection: (11/25/18, 1:30 pm)
      1. There weren’t too many concerns with the following piece of code. All functions were small and only performed one action. Variable names were chosen carefully and were self-explanatory. Exceptions were used properly, however, it would be better if more was done when an exception was caught rather than just printing the stack trace. Perhaps giving an error message would have been better. There were also a few exceptions that weren’t needed and were included just as a catch all exception.
2. Edgar Martinez: Crypto.java, News articles part
   1. Ahmed Khan’s Inspection: (11/26/2018, 4:00 pm)
      1. There was not too much to be concerned about in this part. The biggest problem was that the class was not error checking if the given crypto string was valid or even null before being used. This could cause problems if the string was different than expected because it could lead to inaccurate articles being received. If the string was null, it could lead to a high load of articles that could potentially crash the application. An if statement in the class constructor is needed to check the string, and then defaulting to a safer word if the string is invalid or null.
   2. Rameez Baig’s Inspection: (11/27/18, 12:20 pm)
      1. The code is well commented and easy to understand. The news articles information is returned in an unmodifiable list which makes sure no data can be changed that could potentially crash the application. One possible modification is to use more efficient string building rather than using the ‘+’ operator to concatenate the strings. This would speed up the application and use less memory.
   3. Zakee Jabbar’s Inspection: (11/25/18, 2:00 pm)
      1. There are a few minor changes that can be made for this piece of code. There a few strings that are constants which should be declared in the class scope as a final variable to avoid it being modified and getting incorrect results. This way on each function call new variables are also not created and destroyed. Good use of exception catching has been used but a little more than printing the stack trace should be done when an exception has been caught. Perhaps printing the what part of the news data caused the exception so it’s easier to debug later.
3. Rameez Baig: Controller.java, News articles parts
   1. Ahmed Khan’s Inspection: (11/27/2018, 3:30 pm)
      1. This code is overall very good. The biggest concern is more simple code. Multiple nested loops and conditions can make debugging a nightmare and could potentially cause unseen problems in the future. Breaking the tasks to do one thing at a time instead of multiple things at once could drastically speed up debugging time in the future.
   2. Edgar Martinez’s Inspection: (11/26/18, 5:20 pm)
      1. Really well written code that’s formatted correctly with good function names and small functions that perform only one task. Could improve by having better variable names and more comments that show what the code is doing at that point. Could have avoided creating unnecessary objects because they were already available.
   3. Zakee Jabbar’s Inspection: (11/25/18, 2:30 pm)
      1. This piece of code is well written, but a few tweaks can be made to improve it. There are several temporary variables that are used that can be avoided. These variables store values from variables that already exist at the class scope. Those class variables could be used directly and avoid the creation of local objects when the function is called.
4. Zakee Jabbar: Controller.java, Graphing and Updating Charts
   1. Ahmed Khan’s Inspection: (11/28/2018, 10:30 am)
      1. This code was very well written and is self-explanatory just by looking at the naming conventions used. The only suggestion is to group common lines of code more closely together and use more spaces to make that distinction clear. This way it will be easier to understand when one task has been finished and the next one is being worked on. Other than that, very good and efficient code.
   2. Edgar Martinez’s Inspection: (11/26/18, 6:00 pm)
      1. Really good function name that allows person to know what function does just by the name. Code is formatted correctly and does just one task which is updating the chart. Only complaint would be the lack of comments in the code that could help people understand what exactly the code is doing. Overall the code is structured really well because it’s split into many tiers where this code just updates the front end and calls other functions to perform the job of connect to API and parsing the data that’s received. Which makes all the code more scalable and helps improve the application security.
   3. Rameez Baig’s Inspection: (11/27/18, 12:40pm)
      1. The code is split up into handling the three different graphs, but this is not very noticeable at a first glance. If comments could be added to the code that would be very beneficial to make it very clear which part does what. The code also avoid creating unnecessary local variables and uses the class variables as much as possible. Code is very efficient and organized, but adding a few comments won’t hurt.

# Recommendations and Conclusions

All current features have been properly tested and inspected. All features work as expected. When new features are added test will need to be rerun.

# Project Issues

## Open Issues

There are certain factors that remain to be dealt with in order to fully optimize the product. Although there were no issues that inherently got in the way of the apps functionality, there were issues that slowed down performance. The application was designed without the use of multithreading. The issue with not having done this, is that long API calls that take longer time to get a response can block the main UI thread and cause the program to hang and become un responsive. This was not much of an issue since most of the calls got a response fairly quickly. This would be a more apparent problem if the API calls got bigger or multiple API calls were made on a single user action. Additionally, if the user had a slower internet connection the user would notice the application hanging quite often when making clicks.

## Waiting Room

There are certain implementations which have yet to be followed up with which are worth looking into. Although the application has an impressive amount of functionality in the allotted time, there are things which could be improved upon. One open ended issue that we had yet to resolve was the in-application browser. This would have been a significant addition due to the fact that the application would then be completely self-contained and not need to access outside resources in order to run some of the functionalities. Another extension for this application is the support of stocks. These days a few applications already incorporate the two together. For example Robinhood allows you to trade stocks and cryptocurrencies. It would be ideal if this application can provide the user information on both so they would only need on application to keep track of both.

## Ideas for Solutions

The Cryptocurrency Ticker application was initially meant to be a solution for providing a way on stop shop for everything one needs to know when monitoring Cryptocurrency. Although the application does a great job of analyzing the current market through prices and graph analysis, it lacks the ability for the user to take action. Chase Bank for example, now has the ability for their user to invest within the application. Robinhood which is of a similar design, also allows users to invest within the application. In order for the Cryptocurrency Ticker Application to be a top competition cryptocurrency resource, it is necessary for this functionality to be added in a future release.

The application also needs to be properly multithreaded. This is possible by making all the API calls in an additional thread using the concurrency class. This way the while the user is waiting for the data to be downloaded they can still make other clicks on the UI, as well as cancel the application as needed. This will also the solve the problem of the UI being unresponsive for a long API call.

## Project Retrospective

Over the course of this project, much organization was needed in order to meet weekly objectives. The application itself turned out to meet the standards of the group, and this was due to the fact that the work was divided up in a fair, and efficient manner. During the last few months, the group members continuously divided into pairs of two people. Each of these groups would have a focus on the front-end or the back-end of the application. Through the use of pair programming, it became much more efficient to move through with the development process since it was not as easy to get stuck as it would be had we divided the project into four individual tasks each week. Had there been more time, the group could have improved on the application by adding a few more features. All in all, the work was very efficient and the project over the course of the semester went smoothly. Something to consider in a future endeavor would be to learn a bit more about the crypto market. We chose the product based on that fact that it was interesting. In the case that we devoted a bit more time to research before starting on implementation, perhaps we could have come up with even more unique ideas.

# Glossary

Ask: Ask is the lowest price at which you can buy

Bid: Bid is the highest price at which you can sell;

Last: The price at which the last transaction went through at.

High: The highest price in a given period of time.

Low: The lowest price in a given period of time

Open:  The price at when trading begins on a specific day.

Averages: Average price-per-share during a specific period of time

Volume: The number of shares traded in a specific time period.

Cryptocurrency: An electronic medium representative of monetary value which is encrypted

Bitcoin: A type of digital currency.

# References / Bibliography

|  |  |  |
| --- | --- | --- |
| [1] |  | M. Fowler, UML Distilled, Third Edition, Boston: Pearson Education, 2004. |
| [2] |  | M. Chopker, "Java Code Review Checklist," DZone, 20 June 2014. [Online]. Available: https://dzone.com/articles/java-code-review-checklist. [Accessed 21 November 2018]. |