# CurrencyHFT

Cha Ching

## Transition

Matthew Cordone
Jack Cusick
Patrick Engelsman
Ayushi Mishra
James Nakashian

# Final Release

The final release of CurrencyHFT can be found at http://currencyhft.com. The source code of the project is available at https://github.com/jmcusick/currencyhft.

## Installation

Detailed installation instructions can be found on the Github Wiki's installation page. This installation is outlined below:

1. Run on Ubuntu Desktop 16.04
2. Create an sdd folder in Documents
3. Clone the CurrencyHFT repository into the sdd folder
4. Install node.js and npm
```
sudo apt-get install nodejs
sudo apt-get install npm
sudo ln -s /usr/bin/nodejs /usr/bin/node
```
5. Install MySQL
```
sudo apt-get install mysql-server
```
6. Import MySQL table
```
cd ~/Documents/sdd/currencyhft/database
mysql -u username -p database_name < currencyhft.sql
```
7. Install C++ boost library
```
sudo apt-get install libboost-all-dev
```
8. Install C++ Connector
```
sudo apt-get install libmysqlcppconn-dev
```
9. Install Python libraries
```
sudo pip install python-dateutil --upgrade
sudo pip install schedule
sudo apt-get install python-lxml
sudo pip install requests
```
10. Make a MySQL credentials.txt file in the sdd folder with desired MySQL username, password and database, separated by new lines. Ensure this user has access to the imported database.
11. Install back-end node modules
```
cd ~/Documents/sdd/currencyhft/node
npm install
```
12. Compile data server
```
cd ~/Documents/sdd/currencyhft/node/src
node-gyp configure build
```
13. Start data server
```
cd ~/Documents/sdd/currencyhft/node
npm start
```
14. Install front-end node modules

```
cd ~/Documents/sdd/currencyhft/frontend
npm install
```

15. Start front-end server

```
cd ~/Documents/sdd/currencyhft/frontend
npm run dev
```

16. Access website

```
http://127.0.0.1:3000
```

## Final Test Results

The table below shows the final test results, as tested by all members of the development team. Based on the results of these tests, we believe that CurrencyHFT is ready for production use.

The main source of failed tests is from using bank rates. While this feature was implemented successfully, the use of bank rates was removed from the frontend due to a lack of data. We decided that the final release would be considered fully functional even without the bank rates.

| Test Case | Final Results | Comments |
|---|---|---|
| **TC1** | Pass. | This test case works properly. The user is able to input a different starting and ending currency along with a limit for the number of currencies that can be used in the path. The site will return a display of the optimal path to the user along with all the related information. |
| **TC2** | Pass. | This test case works properly. When the user selects the same starting and ending currency the site will return the currency itself with all other information unchanged. |
| **TC3** | Pass. | This test case works properly. The user is shown zero for the monetary profit and conversion fields, and is shown the actual values for the Optimal Rate and Direct Rate fields. |
| **TC4** | Pass. | This test case works properly. When users select currencies to exclude, the selected currencies are not included in the optimal path. |
| **TC5** | Pass. | This test case works properly. When users select currencies to exclude that are the same as the starting or ending currency, an optimal path is not calculated and is not shown. |
| **TC6** | Fail. | It was discovered that in certain cases, the limit did not work correctly. The returned path would sometimes be limited by the wrong number, e.g. limit by 4 when 3 was entered. It was decided that this was not a serious issue at this time. |
| **TC7 -** | Fail. | It was decided to not include bank rates on the |

| | | |
|---|---|---|
| **TC12** | | frontend, due to a lack of available data. This functionality exists and works, but is not available, so the test case fails. |
| **TC13** | Pass. | User is able to select two different currency tickers. User can freely change the currencies tracked by the tickers and the display graph will change accordingly. |
| **TC14** | Fail. | The same currency is able to be selected and the graph does not change at all. This was not considered a major issue. |
| **TC15** | Pass. | User can select and deselect all the data sets and they would display if selected or not display if they are deselected |
| **TC16** | Pass. | Users can select two different currencies and the conversion is correctly displayed. Conversion updates correctly if user switches currencies. |
| **TC17** | Pass. | The same value is returned whenever the same two currencies are selected. |

## Contribution Summary

All team members contributed to editing, reviewing, and approving the Transition deliverable for final submission. Breakdown of individual contributions is as follows:

Matthew Cordone
Status report, coding standards

Jack Cusick
Installation guide, best practices

Patrick Engelsman
Testing, best practices

Ayushi Mishra
Best practices, status report

James Nakashian
Wrote up final testing results, linked to evidence of best practices

# Status Report

The Status Report Document outlines the team's progress throughout development. The team added the profit page to the main website to demonstrate the actual opportunities to profit from the arbitrage. The team added stylistic elements to the webapp, and updated the homepage/about page. Further, on the dashboard page, time functionality was added for user to select desired timeframes. The team put bank rate functionality on the backend server, but chose not to use the feature in the front end due to lack of data. Tickers were added to the arbitrage page. Front-end display of the website was optimized and styles. The team gave the final demonstration and presentation. The team went back through the test cases outlined in the Stakeholder Review #2 in order to test bug fixes and improve functionality. The team ensured that the code base was well documented and understandable and added a small help and information section on the website's home page. The team fixed many small bugs throughout the implementation.

# Best Practices

## Mandatory

1. Project Management web site

   Github was used as the team's project management website. The CurrencyHFT Github page hosts a number of the other required best practices below. Through github, issues were assigned to team team members on the Issues page, milestones were tracked on the Milestones page, and a wiki developed on the Wiki page.

2. Code Repository

   The CurrencyHFT Github page served as the team's code repository. The team's commits can be seen on the Github commit page, while the various branches can be viewed on the branches page. CurrencyHFT's releases, including the final release, can be seen on the releases page.

3. Documented Coding Standards

   The team utilized a well defined coding standard that documenting coding practices both globally and for several specific languages used by the team. They are also attached at the end. Examples of the adherence to our coding standards include path.cpp and dbapi.py.

4. Use the basic concepts behind object oriented design

   The backend was designed with object oriented principles in mind, as evidenced by the design of the Graph, Path, and Cycle classes. These classes were designed to have high cohesion and low coupling.

## Other Best Practices (must use three or more)

9. Build Tools

   The team used node-gyp, a cross platform command-line tool for compiling native Node.js modules. The binding.gyp file that we used when running the backend server can be found here.

10. Bug Tracking System

    The team used GitHub issue tracking for reporting bugs and the status of their fixes. Bug were tracked as issues on the Issues page with the label “Bug”.

11. Third Party Components / Tools

    The team used the V8 engine to connect the Node.js and C++ code. This is utilized in our C++ rest api located in this file, rest_api.cpp. We also used a MySQL database to store all of our data. A file that connects to this database is api.h.

    We used Vue.js and ElementUI to create a single page web application as our frontend.

# Coding Standards

## Global rules

- Tab spacing = 2
- Variables camel case
- Comments on their own line
- Comments before function
- Variable names are descript
- No extraneous includes or imports
- Informative output
- Shorter functions
- Spaces on either side of any operator
- Double quotes for strings
- Class names capital

## C++

- Use namespace std
- Same line first brackets
- Pound define guard on .h files
- Use const protection when applicable
- Pass parameters by reference whenever possible
- Functions are capital case

## Python

- Functions are camel case
- Use name == main method
- Global variables are all caps

## JavaScript

- ES6 syntax
- “===” for comparison operator
- https://github.com/elierotenberg/coding-styles/blob/master/es6.md
- Minify files