**Project Design Document**

Written by:

Jonathan Martin

Bhagyalakshmi Muthucumar

Amée Stevenson

Last updated 3/22/2018

For CS 3300 - Object Oriented Programming

at the University of Houston - Downtown

**Table of Contents**

[**Classes** 1](#_Toc509560263)

[*Client Classes* 1](#_Toc509560264)

[Wallet 1](#_Toc509560265)

[Currency 1](#_Toc509560266)

[Cryptocurrency 2](#_Toc509560267)

[Trade 2](#_Toc509560268)

[ClientNetworking 3](#_Toc509560269)

[EncryptionService 3](#_Toc509560270)

[*Server Classes* 4](#_Toc509560271)

[ServerNetworking 4](#_Toc509560272)

[FileOperations 4](#_Toc509560273)

[**Modules** 5](#_Toc509560274)

[Input Validation 5](#_Toc509560275)

[Console Printing Utilities 5](#_Toc509560276)

[Background Cryptocurrency Value Updating 5](#_Toc509560277)

[Regularly Saving Data 5](#_Toc509560278)

[**Implementations** 6](#_Toc509560279)

[Prototype 6](#_Toc509560280)

[Final 6](#_Toc509560281)

[**File Operations** 6](#_Toc509560282)

[**UML Diagrams** 7](#_Toc509560283)

[Client Diagram 7](#_Toc509560284)

[Server Diagram 8](#_Toc509560285)

# 

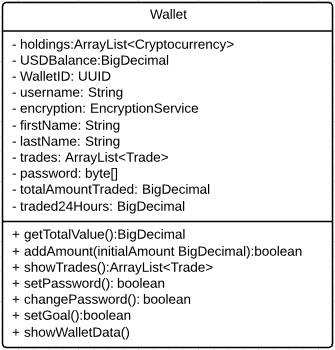
# 

# **Classes**

Classes are split into two parts: client-side classes and server-side classes. Basic details of each class along with a UML class diagram are included below. Full client and server UML diagrams can be found at the end of the document.

## *Client Classes*

### Wallet

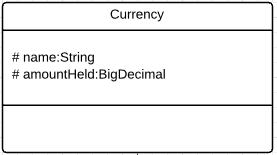


Wallet is the most important class in this program: it is essentially a user profile that stores all the information about a user, their activity, holdings, personal data, and goals. Each will be secured by a password created by its own instance of the EncryptionService. Each will have a list of Trade objects, and plenty of data on the user to be used throughout the program.

## 

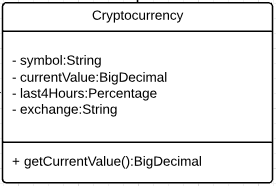
## 

### Currency



Currency is a purely abstract class that derives Cryptocurrency.

### Cryptocurrency



Cryptocurrency is the basis of the contents of the Wallet. Users will trade these for USD, for each other, or trade USD for these. Each of them has its own value, and stores its own performance in the last 4 hours.

## 

## 

### Trade

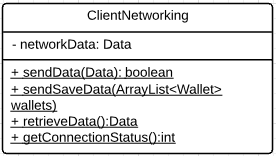


Trade is an object that has details on every trade performed. It also performs the trades themselves within the Wallet. Each wallet has a list of trades, and each of those trading objects performs trading functions inside the wallet.

### 

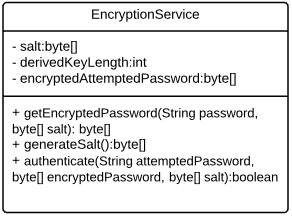
### 

### ClientNetworking



This is a static utility class that will hold all network static functions as defined later. Data is a placeholder, we are unsure of the data type at the moment. This ClientNetworking class holds functions useful to client networking.

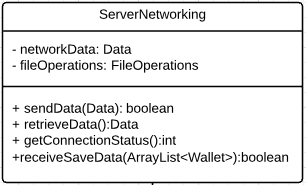
### EncryptionService



Every wallet has an EncryptionService object that performs password creation, changing, and checking abilities. It securely uses the PBKDF2 encryption scheme (now included in Java 8).

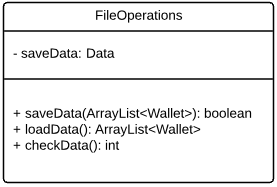
## *Server Classes*

### ServerNetworking



This is a static utility class that will hold all network static functions as defined later. Data is a placeholder, we are unsure of the data type at the moment. This ServerNetworking class holds functions useful to server networking. In addition, because the server is saving data, it has a FileOperations class to handle saving the received data when it is received.

### FileOperations



FileOperations is a static utility class that holds all of the file operations for the application. This includes loading wallets on startup, and saving data at the end of a session before the network connection is terminated. It will work closely with Networking class to save and transmit data. Data is a placeholder for a filetype that we have yet to define.

# 

# 

# **Modules**

Other than the features that classes offer and the operations from the menu that use the classes, the following are useful reusable modules that will help in creating this program.

## Input Validation

Input validation is an annoying but necessary part of every application that requires user input without default validation checking features. Because of this, it would be wise to have a utility class that handles all input validation, checking supplied data by type designated. It would return a boolean.

## Console Printing Utilities

Because we are creating a console application without a formal interface, we want to make it look as good as possible, so that it will not be distracting or annoying. It should be easy to see what data is important, and it should be displayed in a pleasing fashion. Simple things like menu printing, screen clearing, line printing, and common message output would be useful to have in this utility class.

## Background Cryptocurrency Value Updating

Cryptocurrencies are very volatile and require that their values be queried and updated extremely frequently, perhaps every 30 seconds at most. In one section of the application, it will display values of common cryptocurrencies from reputable exchanges. It needs to be updated frequently and stored in the background.

## Regularly Saving Data

The server will store all files long-term, but short-term we need to make sure that the data is secure and safe, even if a connection to the server is broken. It should be locally saved and remotely saved on the server quite often, at least every time a change is made.

# 

# **Implementations**

## Prototype

For the prototyping phase we would like to implement the wallets, local saving, very simple trading and real-time crypto data features of the modules.

## Final

For the final phase, password storage and encryption, remote saving of data, complete virtual trading, and network transfer/operations will be implemented.

# **File Operations**

Because the application uses wallets to store data, we need to save the wallets in an array and serialize them. We will be using the Serializable package for this project. The serializable package will store all the data in a wallet object in a file specified by the user. When a wallet object is created for a new user, the object is saved to a file using an output stream for the wallet object and a output stream that will save it to the file specified. When conducting validation, this file is called using an input stream, which will compare the username and the password against the existing data. As the user conducts trading, the changes to the fields in the wallet object is updated using an input stream to the object. Saving a wallet will be implemented locally in the prototype, and remotely in the final version.

# **UML Diagrams**

## Client Diagram

## Server Diagram

