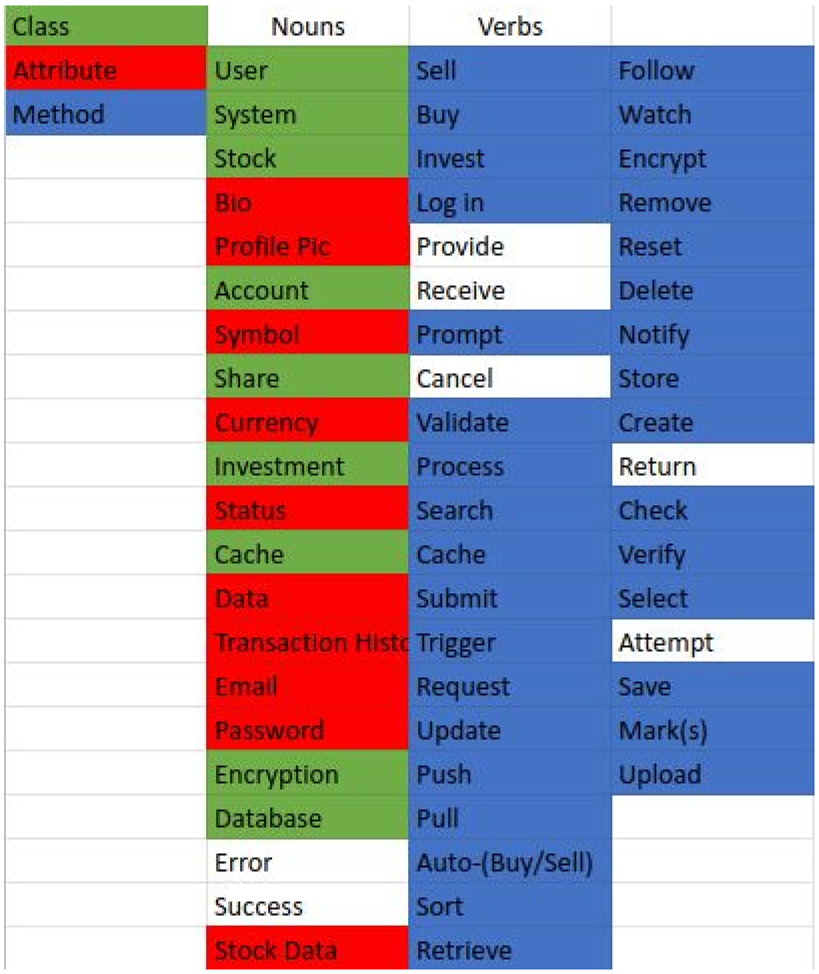
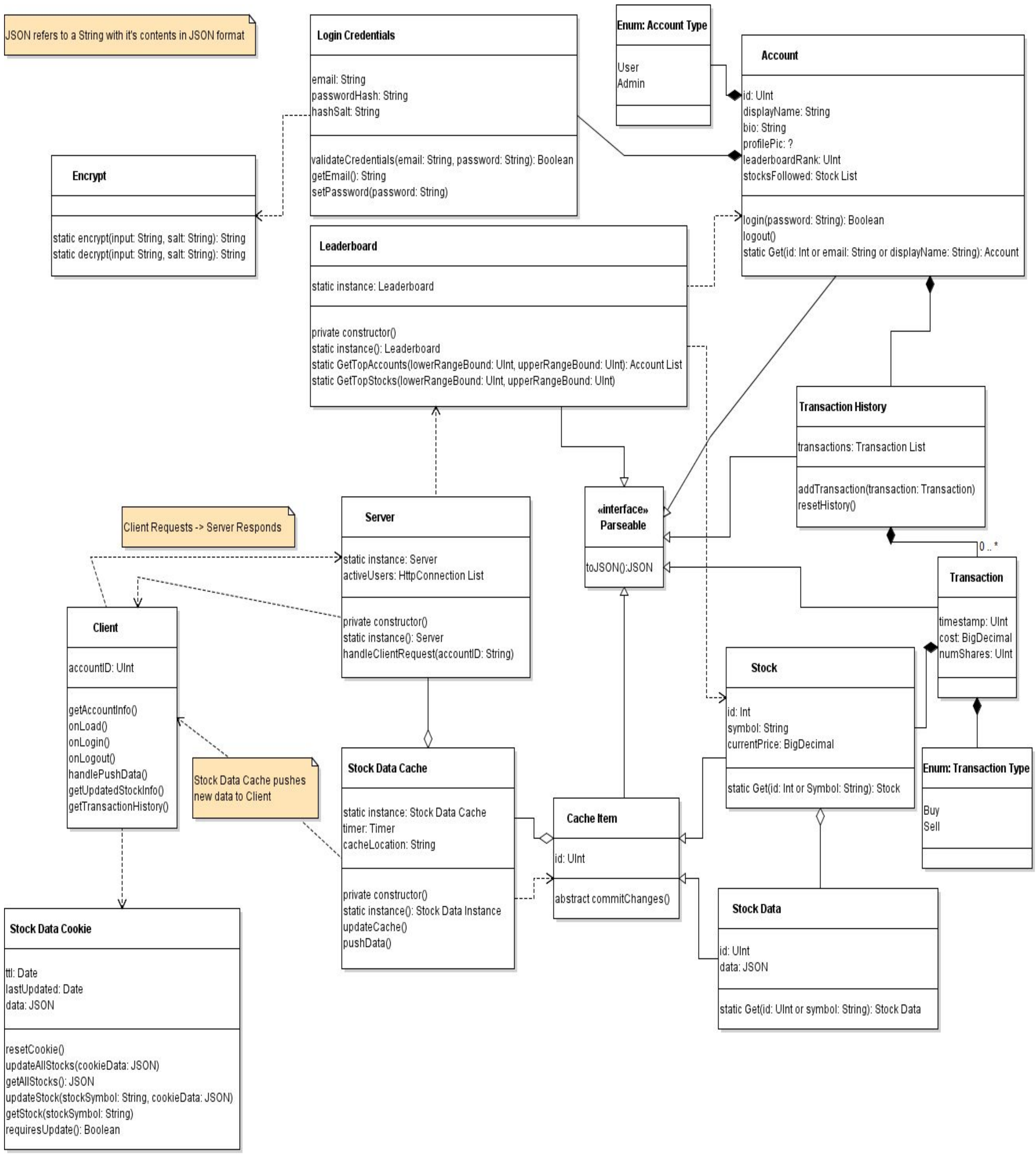


🟩 Class	Nouns	Verbs	
🟥 Attribute	🟩 User	🟦 Sell	🟦 Follow
🟦 Method	🟩 System	🟦 Buy	🟦 Watch
	🟩 Stock	🟦 Invest	🟦 Encrypt
	🟥 Bio	🟦 Log in	🟦 Remove
	🟥 Profile Pic	Provide	🟦 Reset
	🟩 Account	Receive	🟦 Delete
	🟥 Symbol	🟦 Prompt	🟦 Notify
	🟩 Share	Cancel	🟦 Store
	🟥 Currency	🟦 Validate	🟦 Create
	🟩 Investment	🟦 Process	Return
	🟥 Status	🟦 Search	🟦 Check
	🟩 Cache	🟦 Cache	🟦 Verify
	🟥 Data	🟦 Submit	🟦 Select
	🟥 Transaction Histo	🟦 Trigger	Attempt
	🟥 Email	🟦 Request	🟦 Save
	🟥 Password	🟦 Update	🟦 Mark(s)
	🟩 Encryption	🟦 Push	🟦 Upload
	🟩 Database	🟦 Pull	
	Error	🟦 Auto-(Buy/Sell)	
	Success	🟦 Sort	
	🟥 Stock Data	🟦 Retrieve	


JSON refers to a String with it's contents in JSON format
Login Credentials
email: String
passwordHash: String
hashSalt: String
validateCredentials(email: String, password: String): Boolean
getEmail(): String
setPassword(password: String)
Enum: Account Type
User
Admin
Account
id: UInt
displayName: String
bio: String
profilePic: ?
leaderboardRank: UInt
stocksFollowed: Stock List
login(password: String): Boolean
logout()
static Get(id: Int or email: String or displayName: String): Account
Encrypt
static encrypt(input: String, salt: String): String
static decrypt(input: String, salt: String): String
Leaderboard
static instance: Leaderboard
private constructor()
static instance(): Leaderboard
static GetTopAccounts(lowerRangeBound: UInt, upperRangeBound: UInt): Account List
static GetTopStocks(lowerRangeBound: UInt, upperRangeBound: UInt)
Transaction History
transactions: Transaction List
addTransaction(transaction: Transaction)
resetHistory()
Client Requests -> Server Responds
Server
static instance: Server
activeUsers: HttpConnection List
private constructor()
static instance(): Server
handleClientRequest(accountID: String)
«interface»
Parseable
toJSON():JSON
0 .. *
Transaction
timestamp: UInt
cost: BigDecimal
numShares: UInt
Client
accountID: UInt
getAccountInfo()
onLoad()
onLogin()
onLogout()
handlePushData()
getUpdatedStockInfo()
getTransactionHistory()
Stock
id: Int
symbol: String
currentPrice: BigDecimal
static Get(id: Int or Symbol: String): Stock
Stock Data Cache pushes new data to Client
Stock Data Cache
static instance: Stock Data Cache
timer: Timer
cacheLocation: String
private constructor()
static instance(): Stock Data Instance
updateCache()
pushData()
Cache Item
id: UInt
abstract commitChanges()
Enum: Transaction Type
Buy
Sell
Stock Data
id: UInt
data: JSON
static Get(id: UInt or symbol: String): Stock Data
Stock Data Cookie
ttl: Date
lastUpdated: Date
data: JSON
resetCookie()
updateAllStocks(cookieData: JSON)
getAllStocks(): JSON
updateStock(stockSymbol: String, cookieData: JSON)
getStock(stockSymbol: String)
requiresUpdate(): Boolean

Our UML diagram has two major components, Server and Client. They are used to show separation between the backend (server-side) and frontend (client-side) components of the application. The server will handle client communication and manage the cache of stock data. If the user makes an investment or sells stock, the server talks to the Stock Data Cache as well as the Transaction and Account classes in order to process and update the information.

One of the most central pieces of our UML diagram on the server-side is the stock class. Stocks in our system consist of real-world data pulled from actual stock markets. The Stock class holds the stock symbol for a company and a unique integer identifier that we can use server-side to organize and more easily manage stocks. Stock contains attributes for frequently requested data, such as current prices. It also contains a reference to a StockData class which contains all available data for a given Stock, such as price history. They are separated to reduce load and search times for the frequently requested data. Both Stock and StockData have get() methods that return a respective class instance based on the search parameter(s) provided as an argument.

The main piece to the client-side portion of the UML diagram is the Client class. The client class handles the user interaction portion of the application, such as logging in and out, getting account information and transaction history, and requesting updated stock information. Infrequently modified stock data such as company information, price history, and it's stock symbol are stored in a Stock Data Cookie. If the Client looks at the Cookie's time to live and it is expired, the Client asks the Server for updated information about the Stocks.

Users are required to have an account to interact with any of the main application features. Each account will have an associated user identifier, a display name, a short bio, a profile picture, a leaderboard rank, a list of stocks that the account is following, and a transaction history to track investments. Three methods will be included in the account class: logout(), login(), and get(). These methods interact with the server in order to grant or deny access to the website.

When a user invests in a stock, data about the transaction is stored in the transaction history as a Transaction object. The Server will interact with the database and update the user's shares for that stock, as well as their funds. It will also populate the transaction history with the newly completed transaction. The Server then sends back that information to the Client, which will push that data to the user via the handleDataPush() and getTransactionHistory() methods.

An essential part of our data management method is to cache information in order to minimize redundant requests. The StockDataCache class is responsible for the caching of all system-wide stock data. StockDataCache will periodically update the stock cache with the latest real-word market data for every stock within the cache.