project requirements

P3+J3^u! Holdings

Table of Contents

1. **purpose:**
   1. This document describes the requirements for the main project of P3+J3^u! Holdings. That project, referred to hereafter simply as “the project,” is an application to enable increased consistency and profitability in trading stocks and options. The project automates the functions of trading, with the intention that the code will ease the burden of day trading.
2. **core requirements:**

The project will increase the ease and consistency of developing a day-trading strategy and automating the consistency of that strategy.

* 1. **framework:** the project uses a flexible, extensible framework.
     1. Classes extend a base class
     2. All data used by the app is stored in primitives owned by a project class.
     3. Objects know what type they are
     4. Objects know what primitives, objects and methods they contain
     5. Object destructors always overwrite all stored data
  2. **testing:** all code is tested whenever it or any code it depends on changes.
     1. Tests are written for each story before code is written to test the functionality required by the story
     2. When development requires unanticipated code not covered by an existing test, tests must be added to fully cover the code before the story can be completed
  3. **performance:** code is optimized for efficiency.
     1. Data analysis will conclude fast enough that an order to act upon it can be created, sent to a broker and acted upon in time to capitalize
     2. Due to the delay between recognizing a market action and acting upon it, some amount of the profit gained will be lost. This amount must be calculated and tracked.
     3. Limits must be defined for the maximum acceptable loss in profit.
     4. The application must be optimized to execute analysis and the order process quickly enough to meet the limits.
  4. **thread-safeness:** code uses safeguards to ensure multi-threading and multi-processing is done safely.
     1. The application guards against concurrent access attempts to shared data
     2. The application is tested for thread blockage and mutual dependency issues
  5. **data acquisition:** the project can download and store data about financial vehicles—equities and derivatives.
     1. Data can be retrieved for a single equity
     2. Data can be retrieved for a set of specified equities
     3. Data can be retrieved from a variety of sources in a standardized manner
     4. Data can be stored in a persistent, organized state
     5. All stored data will be backed up
  6. **data analysis:** the project can analyze data about financial vehicles using technical indicators
     1. Training
        1. Each technical indicator can be configured to run over the historical data for a given financial vehicle
        2. For each technical indicator it will be possible to specify the parameters in a range (i.e. 5/6/7/8 minute moving average, 10% of max/20% of max/30% of max volume relative to peak in the last 17/18/19 days).
        3. For each permutation of the ranges of parameters for a technical indicator, the profit/loss from trading based on that indicator over a given period of time will be output
        4. The permutations of ranges for the algorithm will output to the user in ranked order of performance
        5. For a set of technical indicators it will be possible to analyze historical data in conjunction and assess when, if at all, their results correlate and with which parameters they do so most/least frequently
     2. Employment
        1. Each analysis algorithm will be re-run on a regular, configurable basis for the equity it covers
        2. Each analysis algorithm can be run on-demand at any time
        3. After each run, the analysis algorithm outputs the anticipated behavior of the vehicle (bearish, bullish, sideways) and the level of confidence in that behavior
        4. Each algorithm can be configured with a number of “reportable events” it recognizes, such as bullish reversal, bearish reversal, etc. as appropriate
        5. Whenever the algorithm runs and detects that the market has experienced a reportable event, it will indicate the event to the controller for the program.
        6. Event reports will include the type of event, the equity in question, the type of analysis performed, and a configurable amount of data related to the event.
        7. The analysis will be performed in a short enough period of time that a human could then act on it to enact a trade on the market.
        8. The analysis can be performed on a multitude of stocks simultaneously
        9. Reportable events will be handled in the order they are generated
  7. **data output:** the project will display the results of the chosen data analysis on the chosen equities
     1. Each report from a reportable event will be displayed via both graphical and textual representations
     2. Reports will be capable of stored by being written to permanent storage
     3. Stored reports will be viewable in the app
     4. Stored textual reports will be viewable in any standard text view
     5. The history of the positions the account has held in relation to any given vehicle will be tracked and stored to persistent storage.
     6. The history of positions the account has held will be accessible to the user via text output.
     7. The history of positions the account has held will be accessible to the user via graphical output.
  8. **broker orders:** the project will be able to log in and place market orders based on the results of data analysis
     1. Orders will be created and sent automatically when appropriate when events occur
     2. Orders will be sent via as secure a mechanism as the broker enables
     3. User data will be stored in an encrypted file that is decrypted using a user password on program startup
  9. **position tracking:** the project track each of the positions the account is in and update this based on the results of each order
     1. All positions will be stored in memory and kept updated on permanent storage
     2. A textual interface will enable the user to determine all outstanding positions
     3. A textual interface will enable the user to determine all data about each outstanding position
     4. A graphical interface will enable the user to determine all outstanding positions
     5. A graphical interface will enable the user to determine all data about each outstanding position
     6. All outstanding positions will be analyzed at startup to determine if any events have occurred since analysis last occurred, and orders will be generated to take corrective action as appropriate
  10. **order generation:** the project will use an algorithm to determine when to generate an order based on the results of data analysis
      1. It will be possible to configure the order generation component in any of several ways for each equity, or for them all:
         1. “paper-trading:” orders are generated and output to logs, but not sent to brokers
         2. “min-quantity-trading:” orders are generated with the minimal amount of equities that a trading system will accept, usually one for stocks and one hundred for options. They are then sent to the broker to execute that trade live.
         3. “min-price-trading:” Orders are generated only when they meet price criteria, i.e. the cost of the trade is less than or equal to $5 per order, or exactly equal to $17.71 per order
         4. “market-cap-percentage:” Orders are generated to be under, equal to or above a certain percentage of a company’s market cap (i.e. no orders will be submitted that allow a position to exceed 10% stake in a company)
         5. “portfolio percentage:” Orders are generated to use less than, exactly or more than the given percentage of the trading portfolio.
         6. “max portfolio percentage:” No order will be generated that ever allows any single vehicle to be a greater percentage of the portfolio than this maximum percentage
      2. The order algorithm will determine the outstanding position of the account for the given vehicle based on the tracking of the position’s history.
      3. The order algorithm will determine the optimal position of the account for a given vehicle based on data analysis of it (appropriately weighted, if multiple indicators are used).
      4. The order algorithm will attempt to reconcile the current position with the optimal position by determining the cost of that activity and weighing that against the benefit that is projected from the adjustment
      5. All overhead costs of placing orders must be calculated when determining the cost to adjust, as well as estimations of the actual prices the vehicle will trade at.

* 1. **Kill switch:** a secondary, stand-alone failsafe application can exit all positions
     1. The kill switch cancels all pending orders.
     2. The kill switch attempts to exit all existing positions at the market price
     3. If the kill switch is unable to exit at market price, it will attempt to do so at a loss—how much so is configurable
     4. It will attempt to exit the positions until it succeeds, by increasing the loss it accepts until all positions have been exited.
     5. It will report the status of the account after completion and log each action—each cancelled pending order, and each attempt to exit each outstanding position.
  2. **logging:** the project will log relevant operations
     1. which operations are logged will be determined by a configurable log level

1. **stretch requirements:**

These requirements are for a completely functional version of the project. Initial release can be made when only the core requirements are met, with these stretch requirements to be added as later features.

* 1. **security:** the project aggressively encrypts data.
     1. All data stored in main memory is encrypted via a block cipher or comparable, save for components of the decryption key
     2. Data is visible in cleartext only in the registers and, if unavoidable, the cache.
     3. The data is encrypted using a dynamically generated key which changes between sessions
     4. The encryption key is protected using public/private key encryption. The private key is derived from some information available only to the application from the kernel, such as its start time in milliseconds rounded down to the nearest prime. At no point is the private key stored in the application’s memory or accessible to any other application or user without root privileges.
     5. All data is securely deleted when memory is released
     6. All input is validated, from config files to recorded stock data to user commands
     7. All account positions stored in memory are encrypted using a block cipher or comparable
     8. The cipher used for “g” uses for its key a user-supplied password on startup
  2. **data acquisition:** in addition to the core requirements for analysis, the following will be included in a future version of the product.
     1. Data can be retrieved by selecting a type of equity and retrieving all equities which match that type specification (i.e. energy)
     2. This “crawling” can be configured to ascertain which equities to pull from a configurable source of equity information
  3. **distributed computing:** the multi-threaded-ness of the application is extended to enable distributed computing.
     1. Methods enable data to be shared securely, using public-private key encryption for keys to block ciphers or the like between components
  4. **data analysis:** in addition to the core requirements for analysis, the following will be included in a future version of the product.
     1. In addition to back-testing technical indicators on older data for profitability, indicators will be extended to estimate the repeatability (stability) of an algorithm’s performance with a given stock
     2. The project will be able to use indicators from third-party libraries
     3. All functionality provided by native indicators will be implemented for analysis performed by third-party libraries