**Project Proposal: simpleBacktester**

**Ashwin Rajesh**

**On Campus, Undergraduate**

**Abstract**

Implementing automated trading strategies has been increasingly popular over the past couple of decades and now with a large increase in the number of retail investors, automation and digital trading strategies are being made available for all to participate in. However, the barriers to engage in these opportunities are still relatively high.

There aren’t too many user-friendly (no-code) interfaces for building, testing, executing, and deploying basic trading strategies. But for the ones that do exist, there is a lack of backtesting functionality. Backtesting is used to analyze the performance of a automated trading strategy using historical performance indicators.

Using existing trading bot platforms, such as the likes of [trellis](https://www.trytrellis.co/), we can start to gage what input parameters and expected outputs may be expected from a potential backtesting solution. The goal of this project is to implement a viable backtesting tool compatible with trellis inputs while applying software architecture techniques from class.

**Overview & Scope of Requirements**

This project will aim to provide users with visual and quantitative insight into the performance of a potential trading strategy that they implement through a bot service. Using Trellis as a standard for bot input parameters we can see how logical components can be designed to define buy and sell triggers as seen in the screenshot below:

Graphical user interface, text, application, chat or text message

Description automatically generated

The proposed backtesting program would then take these parameters and simulate an execution of a strategy using one week of historical stock and indicator data pulled from polygon.io.

There are two main parts of functional component for this workflow:

1. Generation of indicator & historical values

2. Implementation of simulated trading & results compilation

There are many different formats/mediums in which output can be presented to end users, which is why I will be implementing both a graphical visualization of strategy performance benchmarked against the base security and also an API response that will return the expected PnL over a week.

Upon preliminary exploration, we are able to determine what a sufficient backtest should resemble. In the graphic below, we give the model $5,000 to trade and set a single buy trigger that initiates when “Price” “is Above” “Simple Moving Avg 50 Day” using data pulled from polygon’s API for AAPL. The darker blue line simulates the performance of the trading bot and the lighter blue line simulates the performance of APPL stock over the past week:

Chart, line chart

Description automatically generated

Below is a simple setup of form elements using ruby on rails. This will eventually be set up to PUT and GET data from the Golang server.

Graphical user interface

Description automatically generated

**Project Design:**

There are many approaches that could be taken when looking at specific architecture styles. For the functional aspect of server side requirements, it would make sense to consider both pipe & filter and batch sequential styles. Pipe & filter requires the components of the program to act as filters and would mandate the transformation of input data streams into output data streams and while there is a lot of data handover between functions, they are not independent enough to separate and connect via pipes. There are simply too many shared variables and blended function calls. Batch sequential enables the sequential approach to pull, generate, and compare data by invoking separate functions in a fixed order. The “human hand” element of the connectors in batch sequential will work well in this project

When it comes to the client side, ruby already has a model view controller that is native to the framework however this will not be implemented since the model and controller will have too many overlapping responsibilities for this use case that it makes sense to treat them as one. Ideally a client-server in the form of a REST API would be an ideal implementation as the web server can push data to the server which will then be processed and tested, and the server can return the one-week PnL upon request back to the client. Peer to peer is another viable option but there would only be two participants in a given scenario which reinforces the case for client-server.

A hybrid combination of Batch-sequential and Client-server (via REST API) architecture styles would seem best fit for the functional criteria and premises outlined in the project overview. This is because this project requires multiple iterations of processing and calculation while needing a viable style that communicates the results back to the end user.

A picture containing text, screenshot, businesscard

Description automatically generated

Diagram

Description automatically generated

**Project Implementation Plan:**

The tech stack of the project will include:

* Golang for the backend functionality, graphing output, and API interface
* Ruby on rails for the web client to take in input parameters and display PnL output
* Polygon.io is a stock API which will be used to collect stock and indicator historical data