Systematica Hackathon Coding Challenge

- **Real-time market data and execution management systems.**

A day-long coding challenge to provide an insight into the world of financial markets software development.

You are free to use any programming language, any UI technology and any open-source libraries.

You will need to use a Socket.IO client in your programming language of choice as the WebSockets protocol will be used for the real-time communications for this challenge.

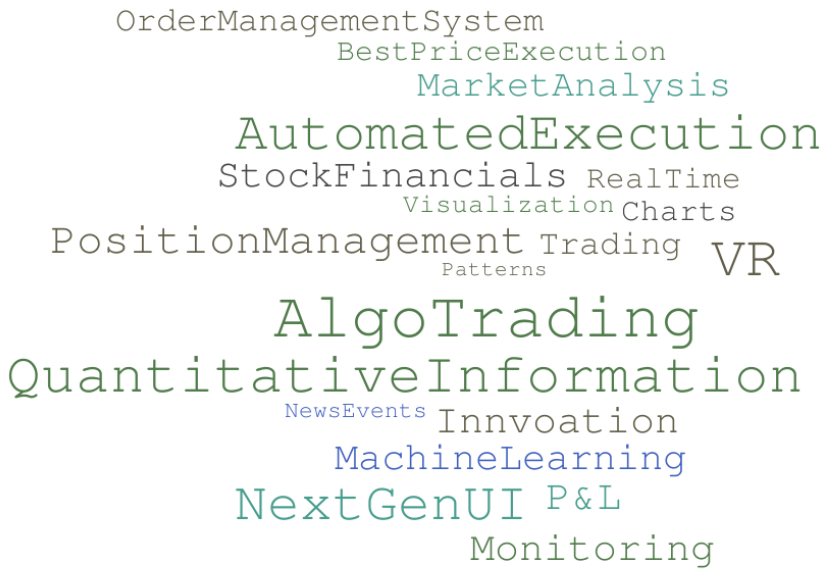
Teams of up to 4 people are allowed.

Agenda
8:30AM - 9:30AM - Breakfast & Introduction.
9:30AM - 12:00AM - Teams work on solution.
12:00AM - 12:30PM - Lunch break
12:30PM - 5:30 PM - Teams work on solution.
5:30PM - 6:30PM - Teams present and deliver solutions.
6:30PM - 7:00PM - Judges finalise and present prizes to the winning teams.
7:00PM - 8:30PM - Dinner, drinks & networking

The Challenge!! Build a real-time trading platform!

- **The competition will be judged on innovation and features.**

We have a mock server API available over the internet to stream real-time market data (Last Trade and Best Bid Offer messages) and to provide a trade execution facility for you to submit orders and received order update messages. Using this API you can build a trading platform.



The Server API

Server API Socket URL: http://emsapi.eu-west-2.elasticbeanstalk.com

Market Data

Available stock symbols: AAPL, AMD, BAC, BMY, C, CSCO, CYH, FB, FCX, GE, INTC, MDLZ, MSFT, WMT, MU, INTC, PFE, VZ, WFX, WMT, XOM

```
//Subscribe for streaming market data
socket.emit("subscribe", stringArrayofSymbols);
//Register a callback function for market data
socket.on("onMarketData", (marketData) => { });

//Last Trade Message
{"time":"1487688301984405000","type":"TRADE","symbol":"AMD","lastPrice":"13.4500"}

//Best Bid Offer Message
{"time":"1487688301265314000","type":"BBO","symbol":"FB","bid":"133.5100","ask":"133.4
900"}
```

Time is the number of milliseconds elapsed since 1 January 1970 00:00:00 UTC.

Trade Execution

```
//Send a NewOrder message for execution
socket.emit("submitOrder", newOrder);

//Register a callback function for order update messages
socket.on("onOrderMessage", (orderMsg) => { });
```

New Order Fields	Data Type	value
type	string	'NewOrder'
clientOrderId	string	{Your unique id string used to identify and link messages for a given order}
symbol	string	{Stock Symbol}
buySell	string	'BUY' or 'SELL'
qty	Integer	{Order Quantity value}

Message sequence:

Example messages:

```
//Send a NewOrder message for execution
{"type":"NewOrder","clientOrderId":"EMS-
1489676741770","buySell":"BUY","qty":"1791","symbol":"FB"}

//Receive OrderNew Acknowledgement message - your order is accepted
{"type":"OrderNew","time":1489676741875,"clientOrderId":"EMS-
1489676741770","buySell":"BUY","qty":"1791","symbol":"FB"}

//Receive a series of ExecutionReport messages notifying you of executed quantity at a
price.
{"type":"ExecutionReport","time":1489676742274,"executionReportId":"EXEC-
1489676742274","clientOrderId":"EMS-
1489676741770","qty":179,"price":"133.7900"}
{"type":"ExecutionReport","time":1489676742774,"executionReportId":"EXEC-
1489676742774","clientOrderId":"EMS-
1489676741770","qty":179,"price":"133.7900"}
{"type":"ExecutionReport","time":1489676743274,"executionReportId":"EXEC-
1489676743274","clientOrderId":"EMS-
1489676741770","qty":179,"price":"133.6400"}
{"type":"ExecutionReport","time":1489676743774,"executionReportId":"EXEC-
1489676743774","clientOrderId":"EMS-
1489676741770","qty":179,"price":"133.6400"}
{"type":"ExecutionReport","time":1489676744274,"executionReportId":"EXEC-
1489676744274","clientOrderId":"EMS-
1489676741770","qty":179,"price":"133.6400"}
{"type":"ExecutionReport","time":1489676744774,"executionReportId":"EXEC-
1489676744774","clientOrderId":"EMS-
1489676741770","qty":179,"price":"133.6400"}
{"type":"ExecutionReport","time":1489676745274,"executionReportId":"EXEC-
1489676745274","clientOrderId":"EMS-
1489676741770","qty":179,"price":"133.6400"}
{"type":"ExecutionReport","time":1489676745774,"executionReportId":"EXEC-
1489676745774","clientOrderId":"EMS-
1489676741770","qty":179,"price":"133.6400"}
{"type":"ExecutionReport","time":1489676746274,"executionReportId":"EXEC-
1489676746274","clientOrderId":"EMS-
1489676741770","qty":179,"price":"133.6500"}
```

```
{"type":"ExecutionReport","time":1489676746774,"executionReportId":"EXEC-
1489676746774","clientOrderId":"EMS-
1489676741770","qty":179,"price":"133.6500"}
{"type":"ExecutionReport","time":1489676747273,"executionReportId":"EXEC-
1489676747273","clientOrderId":"EMS-
1489676741770","qty":1,"price":"133.6500"}

//If your NewOrder is not accepted you will receive an OrderReject message.
{"type":"OrderReject", "clientOrderId":"EMS-1489676741770", "reason":"problem
buySell missing."}
```

Feature Ideas

Market Data Analysis & Visualization
Show stock symbols and last trade prices
Calculate last trade price change from previous last trade price for each symbol
Visualize last trade price change (up/down)
Track high and low last trade prices for each symbol
Show bid and ask prices
Visualize bid and ask price changes (up or down)
Visualize bid/ask spread (difference) value
Visualize prices over time
Visualize price change relationships over time

Trade Execution
Submit orders for the available stocks
Process the order messages and compute/visualize the correct order state. Order state logic: 1. New (executed qty == 0) 2. Partially Filled (executed qty > 0) 3. Filled (executed qty == order.qty)
Show the total executed qty as it updates in real-time
Show execution reports for the orders
Calculate the average price for the executed qty
Calculate the unrealized P&L - the difference between the current last trade price and the average price for the executed qty. **Unrealized P&L =** (LastTradePrice – AvgExecutedPrice) * signedExecutedQty
Visualize P&L changes
Track total position in a given stock
Track total position P&L

Reference implementation

A basic reference implementation of the trading platform using this API built using JavaScript ES6 and React is at the link below.

Web EMS: http://emsclient.s3-website.eu-west-2.amazonaws.com/

Socket.IO Client Libraries

Java / Android: https://github.com/socketio/socket.io-client-java

JavaScript: https://www.npmjs.com/package/socket.io-client

Swift / iOS / OS X: https://github.com/socketio/socket.io-client-swift

C++: https://github.com/socketio/socket.io-client-cpp

C#: https://www.nuget.org/packages/SocketIoClientDotNet/

Python: https://pypi.python.org/pypi/socketIO-client

This document is available for download at:

https://s3.eu-west-2.amazonaws.com/hackathonmarch2017/Hackathon.pdf