CurrencyFair

CurrencyFair is a peer-to-peer marketplace for currency exchange which allows people to save up to 90% when transferring money internationally when compared with the banks. We have had exceptional growth over the last number of years, and are aiming for even further growth in 2015. As a result of this, engineering at CurrencyFair is looking to hire people who are excited by the domain and are looking to work with a team which aims to be a leader in the space.

CurrencyFair Engineering

Our engineering team is a diverse group of people who share the same aim. We want to build a secure, scalable system which delivers for our customers. We want to achieve this with respect for each other taking into account different perspectives. We try to have open minds in relation to implementations and will look at the best technology for a given problem.

Process

Our development process goes a little something like this:

1. Product team create requirements
2. Engineers begin work with the item being managed via Sprint.ly
3. Once complete, work goes into code review where at least two other engineers need to thumbs up
4. Once the work is reviewed and merged, it is automatically deployed to staging
5. One final check and we deploy via Jenkins to production

In Production

- Java
- PHP
- Ruby
- Python

Infrastructure

- Dedicated data centres
- Global AWS resources
- Management via Chef and Vagrant

Aims

- Horizontally scalable applications
- Distributable services
- Secure implementations
- Performant

Why a test?

We wanted a method of measuring both applicants' interest and ability. The work below can be implemented very quickly in an hour or two, or can be done over a number of days - the amount of effort put in is up to you.

There are a number of components and you decide if you want to put most effort into one component, or all components. In each section, we have Easy, Average, and Hard examples. These are just that - examples, and you don't have to use them. In fact, it would be better if you don't.

Feel free to implement the work below with whatever language / framework you prefer.

Market Trade Processor

CurrencyFair processes a large number of market interactions per second. We would like you to build a market trade processor which consumes trade messages via an endpoint, processes those messages in some way and delivers a frontend of processed information based on the consumed messages.

Message Consumption

Goal

Expose an endpoint which can consume trade messages. Trade messages will be POST'd (assume by CurrencyFair during review) to this endpoint and will take the JSON form of:

{"userId": "134256", "currencyFrom": "EUR", "currencyTo": "GBP", "amountSell": 1000, "amountBuy": 747.10, "rate": 0.7471, "timePlaced" : "24-JAN-15 10:27:44", "originatingCountry" : "FR"}

Some implementation ideas

Easy

Consumed messages are written to disk for rendering in the frontend.

Average

You have implemented rate limiting in your message consumption component.

Hard

The message consumption component is the main piece of work you focus on, and can handle a large number of messages per second.

Message Processor

Goal

Process messages received via the message consumption endpoint. Depending on what you wish to do, these messages can be processed in different ways.

Some implementation ideas

Easy

Carry out no processing, and let messages filter to frontend directly.

Average

Analyse incoming messages for trends, and transform data to prepare for a more visual frontend rendering, e.g. graphing currency volume of messages from one particular currency pair market (EUR/GBP).

Hard

Messages are sent through a realtime framework which pushes transformed data to a Socket.io frontend.

Message Frontend

Goal

Render the data from the output of the other two components.

Some implementation ideas

Easy

Render a list of consumed messages.

Average

Render a graph of processed data from the messages consumed.

Hard

Render a global map with a realtime visualisation of messages being processed.

What we look for

Just because you take an easy approach won't set you back in our eyes. Easy might just mean you're busy - we understand. Either way, we look at the same areas:

- Approach taken
- Architecture
- Code structure and clarity
- Performance
- Security
- Testing

Submission

Once you're ready, click on the link below. You will be asked for three things:

1. The endpoint we should POST messages to during our review process
2. The frontend endpoint we should load to view the output
3. The **public** GitHub repository where we can fork and review your code

If you would like to include any other information or points, please include a README.md in the GitHub repo.

Submission link: http://goo.gl/forms/GRxFaOzUib

If we are happy with the work submitted, we will arrange for an interview which is the final step in the process.

Good luck.