README.md

Coinbase Project

Create a web service that provides quotes for digital currency trades using data from the GDAX orderbook.

Web Service (AxQuoter/API)

Description

The web service receives a json with `action` , `base_currency` , `quote_currency` and `amount` and responds with the best quote available based on GDAX API order book.

Documentation

QuoteController (Controller)

Responsible for receiving the POST request, validate parameters and call the AxQuoter API lib. Handles response as a json.

AxQuoter (Library)

Provide quotes for digital currency trades based on GDAX API order book.

- Quoter
  - Retrieves order book calling GDAX Api lib and provides the best quote for desired amount.
- Calculator
  - Calculates total price based on the GDAX API order book.
- ErrorHandler
  - Rescues known errors on the controller rendering them as json.

Gdax API (Lib)

Responsible for communicating with GDAX API

- API
  - order_book
    - Requests order book for specified currency-pair on GDAX API. Responsible for inverting the order book if the currency pair is inverted.
- Order
  - Class to store orders from order book retrieved from GDAX API.
- Request
  - Handles HTTP requests.
- OrderBookRequest
  - Retrieves order book from GDAX API.
- StandardOrderBook
  - Class to store order books retrieved from GDAX API.
- InvertedOrderBook
  - Class to store inverted order books retrieved from GDAX API.

How it works:

The service receives a currency-pair and needs to retrieve the order book from GDAX API for that currency-pair.

If the order book exists in the requested direction (`base_currency-quote_currency`), it is returned by the GDAX API and its values are stored on `StandardOrderBook` . `StandardOrderBook` stores the price and size of orders on asks and bids arrays exactly as they come from the retrieved order book.

If the order book only exists in the opposite direction (`quote_currency-base_currency`), it is stored on `InvertedOrderBook` . In this case, when the action is buy, we are actually selling the quote currency, so we need to consider the bids orders as asks orders and invert the prices and sizes.

`InvertedOrderBook` calculates the inverted price (`1 / original_price`) and inverted size (`original_size * original_price`) and stores these orders on the exchanged asks and bids arrays.

```
Example:
action: buy
base_currency: USD
quote_currency: BTC
amount: 10
```

This means we want to buy 10 units of USD paying with BTC. In this case, as we only have the BTC-USD order book, we need to consider that we are actually selling BTC receiving USD.

Version

- Ruby 2.2.2
- Rails 4.1.15

How to use it:

1. Run a local server:

```
bundle exec rails s
```

2. Make the POST request

a) Curl

- Examples:

```
curl -X POST \
  http://localhost:3000/quote \
  -H 'accept: application/json' \
  -H 'content-type: application/json' \
  -d '{
 "action": "buy",
 "base_currency": "BTC",
 "quote_currency": "USD",
 "amount": "1.0"
}'
```

Response

```
{"total":"5897.29","price":"5897.29","currency":"USD"}
```

```
curl -X POST \
  http://localhost:3000/quote \
  -H 'accept: application/json' \
  -H 'content-type: application/json' \
  -d '{
 "action": "sell",
 "base_currency": "USD",
 "quote_currency": "BTC",
 "amount": "10.0"
}'
```

Response

```
{"total":"0.00169569","price":"0.00016957","currency":"BTC"}
```

b) Postman

- Select POST action for `http://localhost:3000/quote`
- Headers: `Accept: application/json` `Content-Type: application/json`
- Body(raw):

```
{
 "action": "buy",
 "base_currency": "BTC",
 "quote_currency": "USD",
 "amount": "1.0"
}
```

- Response

```
{
    "total": "0.00171527",
    "price": "0.00017153",
    "currency": "BTC"
}
```

Running the tests suite

To run the test suite, run the command bellow:

`rspec -fd spec`

The result will be a documented tree of the project:

```
API::QuoteController
  POST #quote
    when request is json
      with valid params
        returns a success response
        returns quote in json
      with missing params
        returns parameter missing error with status 400
      with invalid action
        returns invalid params error with status 400
      with invalid amount
        returns invalid params error with status 400
    when request is not json
      returns not acceptable error with status 406

AxQuoter::Calculator
  #total
    when action is buy
      gets asks from order book
      returns sum of weighted prices from asks orders for desired amount
    when action is sell
      gets bids from order book
      returns sum of weighted prices from bids orders for desired amount

AxQuoter::Quoter
  #quote
    when action is buy
      returns lowest price to buy this amount in the quote currency and unit price
    when action is sell
      returns highest price to sell this amount in the quote currency and unit price
    when amount is exceeded
      raises amount exceeded error

GdaxAPI::API
  .order_book
    when currency-pair is valid
      returns order book from api
    when currency-pair is inverted
```

```
      returns inverted order book from api
    when currency-pair is invalid
      raises not found error

GdaxAPI::InvertedOrderBook
  #bids
    returns an array with inverted bids (asks) orders
    returns bids (asks) orders with inverted price and size
  #asks
    returns an array with inverted asks (bids) orders
    returns asks (bids) orders with inverted price and size
  #type
    returns inverted

GdaxAPI::StandardOrderBook
  #bids
    returns an array with bids orders
    returns bids orders with standard price and size
  #asks
    returns an array with asks orders
    returns asks orders with standard price and size
  #type
    returns standard

GdaxAPI::Order
  #price
    returns orders price
  #size
    returns orders size
  #total
    returns orders price multiplied by size

GdaxAPI::OrderBookRequest
  .perform
    requests order book based on currency-pair

GdaxAPI::Request
  .get
    when request is valid
      performs to path at gdax api with query params
      returns a parsed json
    when request is invalid
      returns nothing

Finished in 0.08847 seconds (files took 2.49 seconds to load)
33 examples, 0 failures
```

Hours spent

- Understanding the problem: 2 hours
- Designing class structure: 2 hours
- Implementing: 6 hours
- Refactoring: 1 hour
- Testing: 6 hours
- Error handling: 2 hours
- Documenting: 1 hour