

Orderbook Test

The Problem

Stakeholders are requesting that we add a live orderbook to our trading interface that allows users to visualize the interest in their selected market whilst they create new orders, aligning our trading interface with what our competitors in the market offer.

Description of what an orderbook is can be found [here](#).

The Solution

Create a small application containing an orderbook to display to stakeholders that we can deliver on the functionality required by the business. We have the freedom to use any build toolchain or helper libraries necessary, but we must stick to our core technologies of TypeScript + React/React Native and work under the assumption that the patterns and implementation details used here will set the standards for the rest of our user interface.

The orderbook will be made up of four main parts and mirror the following design:

Desktop Design

Order Book					
Spread: 17.0 (0.05%)			Group 0.50 ▾		
TOTAL	SIZE	PRICE	PRICE	SIZE	TOTAL
1,200	1,200	34,062.50	34,079.50	3,356	3,356
13,451	12,251	34,062.00	34,080.00	23,999	27,355
15,329	1,878	34,059.50	34,080.50	10,000	37,355
17,829	2,500	34,056.00	34,083.50	4,999	42,354
25,039	7,210	34,055.00	34,086.50	1,015	43,369
26,916	1,877	34,054.00	34,087.00	10,000	53,369
29,031	2,115	34,053.50	34,088.50	26,780	80,149
31,131	2,100	34,053.00	34,089.00	22,171	102,320
32,063	932	34,052.50	34,089.50	1,013	103,333
37,412	5,349	34,051.50	34,092.00	2,190	105,523
114,438	77,026	34,050.00	34,092.50	3,343	108,866
287,826	173,388	34,049.50	34,093.00	2,500	111,366
319,888	32,062	34,049.00	34,093.50	77,184	188,550
326,297	6,409	34,048.50	34,094.00	16,222	204,772
331,298	5,001	34,048.00	34,095.00	2,655	207,427
333,306	2,008	34,047.50	34,095.50	6,699	214,126

Toggle Feed

Kill Feed

Mobile Design

Order Book		Group 0.50 ▾
PRICE	SIZE	TOTAL
34,092.00	990	122,554
34,091.00	2,482	121,564
34,090.50	2,500	119,082
34,089.50	1,013	116,582
34,087.50	1,200	115,569
34,087.00	58,951	114,369
34,086.50	4,393	55,418
34,086.00	7,000	51,025
34,082.50	3,299	44,025
34,080.50	14,999	40,726
34,080.00	23,999	25,727
34,074.00	1,728	1,728
Spread: 13.0 (0.04%)		
34,061.00	1,958	1,958
34,060.00	12,944	14,902
34,056.50	3,216	18,118
34,055.00	9,131	27,249
34,052.50	4,146	31,395
34,052.00	2,500	33,895
34,051.50	2,053	35,948
34,051.00	1,994	37,942
34,050.50	5,304	43,246
34,050.00	77,026	120,272
34,049.50	173,388	293,660
34,049.00	30,000	323,660
Toggle Feed Kill Feed		

Part 1: The Orderbook

The orderbook consists of two sides: the buy side and the sell side. Both sides contain information about the amount of orders open at each price level.

Each level displays the:

- **Price** - this is what defines the level. As orders must be placed at a price that is a multiple of the selected markets tick size (0.5) each level will be an increment of 0.5 (as long as there is an order open at that level).
- **Size** - the total quantity of contracts derived from open orders that have been placed at this level.
- **Total** - the summed amount of contracts derived from open orders that reside in the book at this level and above. To calculate the total of a given level we take the size of the current level and sum the sizes leading to this price level in the order book. The total is also used to calculate the depth visualizer (colored bars behind the levels), the depth of each level is calculated by taking that level's total as a percentage of the highest total in the book.

To retrieve the data feed necessary to build the orderbook we'll be using the following public WebSocket: `wss://www.cryptofacilities.com/ws/v1` and sending the following message to this WebSocket: `{"event":"subscribe","feed":"book_ui_1","product_ids":["PI_XBTUSD"]}`.

This data feed first returns a snapshot of state representing the existing state of the entire orderbook followed by deltas representing singular updates to levels within the book. The orders returned by the feed are in the format of `[price, size]`. If the size returned by a delta is 0 then that price level should be removed from the orderbook, otherwise you can safely overwrite the state of that price level with new data returned by that delta.

Part 2: Grouping Select Box

By default the orders are grouped by the selected markets ticket size (0.5). We need to be able to toggle this grouping between 0.5, 1 and 2.5. To group levels we combine the levels rounded down to the nearest group size e.g. if we change our grouping from 0.5 to 1 then we would combine the data from prices 1000 and 1000.5 and display it under a single level in the orderbook with the price 1000.

Part 3: Toggle Feed Button

Clicking this button should toggle the selected market between `PI_XBTUSD` and `PI_ETHUSD`. To switch contracts we need to unsubscribe from our existing data feed by sending the following message: `{"event":"unsubscribe","feed":"book_ui_1","product_ids":["PI_XBTUSD"]}` and resubscribe to the new feed using the same subscribe originally sent but with the new product_id `PI_ETHUSD`.

Please note that these markets use different tick sizes, 0.5 for `PI_XBTUSD` and 0.05 for `PI_ETHUSD`

so your grouping logic must be dynamic and handle groups for XBT (0.5, 1, 2.5) and groupings for ETH (0.05, 0.1, 0.25).

Part 4: Kill Feed Button

Clicking this button should force the WebSocket feed to throw an error. This is purely so we can see how you're handling error events. Clicking this button a second time should restart the feed.

Success Criteria

1. Application architecture and framework best practices for the framework in use are followed and understood.
2. All necessary areas of the codebase are strictly typed, and definitions are logically named and reused where possible.
3. Code is formatted well and easy to follow. Variable and function names make sense
4. Any dependencies that have been added have logical reasons for their inclusion, and have been implemented correctly.
5. Understanding of client side state management is displayed.
6. Rendering performance is clearly considered and the application UI runs smoothly on all evergreen browsers and low end devices.
7. Application gracefully handles WebSocket error cases and is resistant to unexpected messages.
8. Application considers that the WebSocket data throughput can fluctuate and dramatically increase during high trading volatility. It has mechanisms in place to ensure the UI can not be overworked and slow down due to processing this data.
9. Test coverage of the orderbook responses is sufficient and thought has been put into what areas of the application should and shouldn't be tested.

Submission Requirements

1. Send an email to your recruiter with a link to a public git repository (in GitHub/BitBucket/GitLab, etc) with the name **[firstname]-[submission date]**.
2. Provide a link to a hosted version of the project Vercel.
3. **Do not mention Kraken in the submission anywhere - including the code!**
4. You will be asked to screen share and walk through this app & code in your next interview, please have it ready to be run prior to the interview.