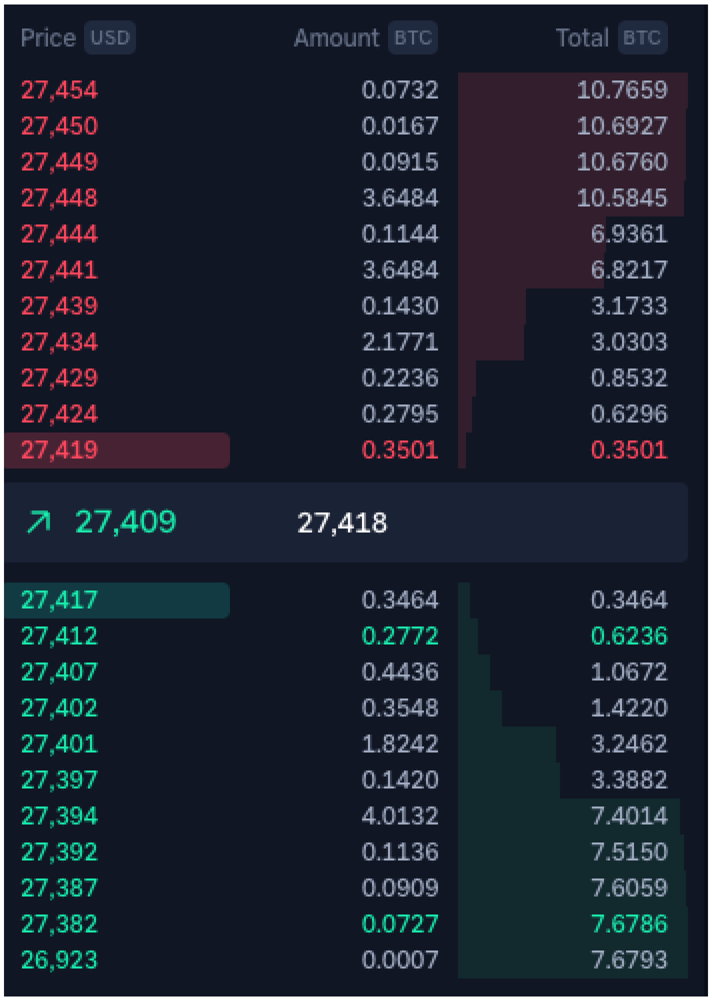
RABBITX

Orderbook Synchronization Test Assignment

Summary: Create a ReactJS UI component to synchronize and display an orderbook using websocket updates.

Task Description: Your task is to implement an isolated UI component that represents an orderbook, which displays bids and asks by traders on an exchange. Utilize websocket updates to construct and update the orderbook (bids and asks array) in an optimized manner without causing memory leaks. Implement logic to handle network connection loss and resubscribe automatically, as well as manage lost packages based on incorrect sequence numbers.
You don’t have to fully finish the assignment; spend at most one week on it.

Example Orderbook UI:



Price USD	Amount BTC	Total BTC
27,454	0.0732	10.7659
27,450	0.0167	10.6927
27,449	0.0915	10.6760
27,448	3.6484	10.5845
27,444	0.1144	6.9361
27,441	3.6484	6.8217
27,439	0.1430	3.1733
27,434	2.1771	3.0303
27,429	0.2236	0.8532
27,424	0.2795	0.6296
27,419	0.3501	0.3501
↗ 27,409	27,418	
27,417	0.3464	0.3464
27,412	0.2772	0.6236
27,407	0.4436	1.0672
27,402	0.3548	1.4220
27,401	1.8242	3.2462
27,397	0.1420	3.3882
27,394	4.0132	7.4014
27,392	0.1136	7.5150
27,387	0.0909	7.6059
27,382	0.0727	7.6786
26,923	0.0007	7.6793

Resources:

1. Orderbook synchronization via websockets: https://docs.rabbitx.io/api-documentation/websocket/orderbook
2. Subscribing to websockets: https://docs.rabbitx.io/api-documentation/websocket
3. Orderbook response data structure (OrderbookData): https://docs.rabbitx.io/api-documentation/data-structure
4. Centrifuge-js SDK for websocket connection: https://github.com/centrifugal/centrifuge-js
5. JWT for subscribing to public websockets channels:
   - wss://api.testnet.rabbitx.io/ws: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIwIiwiZXhwIjo1MjYyNjUyMDEwfQ.x_245iYDEvTTbraw1gt4jmFRFfgMJb-GJ-hsU9HuDik"
   - wss://api.prod.rabbitx.io/ws: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI0MDAwMDAwMDAwIiwiZXhwIjo2NTQ4NDg3NTY5fQ.o_qBZltZdDHBH3zHPQkcRhVBQCtejIuyq8V1yj5kYq8"

Requirements:

1. Use ReactJS for building the UI component
2. Integrate the centrifuge-js SDK for websocket connections

Technical Specifications:

1. Develop a ReactJS UI component to represent the orderbook, including bids and asks.
2. Establish a websocket connection using the centrifuge-js SDK and subscribe to the orderbook updates.
3. Process the incoming websocket data to construct and maintain an optimized bids and asks array, ensuring efficient memory management (https://docs.rabbitx.io/api-documentation/websocket/orderbook).
4. Merge existing bids and asks with incoming websocket updates, keeping the orderbook up-to-date.
5. Implement logic to handle network connection disruptions, including automatic reconnection and resubscription.
6. Account for lost packages by verifying the correctness of sequence numbers in the websocket updates.

Deliverables:

1. A ReactJS application containing the orderbook UI component.
2. Source code with clear and concise comments explaining the implementation.
3. A brief document outlining the approach taken, challenges faced, and possible improvements.

Evaluation Criteria:

Your submission will be evaluated based on the following:

1. Functionality: The UI component should accurately display and update the orderbook using websocket data.
2. Code Quality: The source code should be clean, well-organized, and properly commented.
3. Optimization: The application should handle memory efficiently and avoid memory leaks.
4. Robustness: The solution should account for network disruptions and lost packages.