**Real-time trade updates**

This feature is crucial for providing users with the most up-to-date information on their trades and market conditions.

**1. Programming Languages and Frameworks**

**Frontend:** React.js

->React.js is excellent for building dynamic and responsive user interfaces. It allows for efficient updating and rendering of components, which is essential for real-time data updates.

**Backend:** Node.js with Express.js

->Node.js is ideal for building scalable network applications. Express.js is a minimal and flexible Node.js web application framework that provides robust features for web and mobile applications.

**Database:** MongoDB

-> MongoDB is a NoSQL database that provides flexibility and scalability. It handles large volumes of data and allows for fast read and write operations, which is critical for real-time updates.

**2. High-Level Architecture and Low-Level Design (LLD)**

**High-Level Architecture:**

Client-Server Model:

-> Client: React.js application running on the user's device, which communicates with the backend server to fetch and display real-time trade updates.

-> Server: Node.js server with Express.js handling API requests from the client and communicating with the database.

-> Database: MongoDB for storing user trade data and market conditions.

**Low-Level Design:**

**Frontend:**

-> Components: TradeList, TradeItem, RealTimeUpdates.

-> State Management: Using React hooks and Redux Toolkit to manage state and propagate real-time updates.

-> WebSocket Connection: Establish a WebSocket connection to receive real-time updates from the backend.

**Backend:**

-> API Endpoints: RESTful API endpoints for fetching initial trade data and WebSocket endpoints for real-time updates.

-> WebSocket Server: Implement a WebSocket server using Socket.io to push real-time updates to the client.

-> Database Operations: Use Mongoose for MongoDB operations to fetch and update trade data.

**Database Schema:**

-> User Collection: Stores user information and trades.

-> Trade Collection: Stores trade details, status, and real-time updates.

**3. Third-Party APIs or Services**

**Socket.io:**

-> Socket.io is a library that enables real-time, bidirectional, and event-based communication. It allows the server to push real-time updates to the client efficiently.

**Firebase Authentication:**

-> Firebase Authentication provides secure and easy-to-implement user authentication. It ensures that only genuine users can access and perform trades on the platform.

**Market Data API** (e.g., Alpha Vantage, Yahoo Finance):

-> To fetch real-time market data and conditions, integrating a reliable market data API ensures that users receive accurate and up-to-date information for their trades.

**Implementation Steps:**

1. Set Up React Frontend:

-> Initialize a React project using Create React App.

-> Create components for displaying trade data and real-time updates.

-> Establish a WebSocket connection to listen for updates.

2. Set Up Node.js Backend:

-> Initialize a Node.js project with Express.js.

-> Implement RESTful API endpoints for initial data fetching.

-> Set up a WebSocket server using Socket.io for real-time updates.

-> Integrate Firebase Authentication for user login and registration.

3. Database Integration:

-> Design MongoDB schema for users and trades.

-> Implement database operations using Mongoose.

4. Integrate Third-Party APIs:

-> Connect to a market data API to fetch real-time market conditions.

-> Use Firebase Authentication for secure user management.

5. Testing and Deployment:

-> Thoroughly test the application for performance and security.

-> Deploy the backend on a cloud service (e.g., AWS, Heroku) and the frontend on a static site host (e.g., Netlify, Vercel).

This approach ensures a scalable, secure, and real-time trading feature implementation on the Probo app, providing users with an enhanced and responsive trading experience.