Project Title

Paper Trading

Project Summary

This web-based application will allow users to practice their trading skills on both historical and artificial markets. Users can trade in groups, compete against each other (e.g., users vs. users, groups vs. groups), and participate in tournaments between users and groups, with potential prizes for high-placing participants.

Description

A paper-trading web-based application that will allow people to trade either individually or in groups. Users can take on one of three roles: the owner, trader, or risk manager. Users can place orders or create strategy templates that can be stored, reused, and/or modified. Riskier and/or bigger orders may require approval before execution and all activity is logged. Our app will also support multiple types of assets, including stocks, crypto, options, and futures.

We will use two price datasets: 1) Historical prices for backtests and 2) artificial prices for testing strategies in different market conditions. There will also be a news feed that pulls notable events like company updates, filings, earnings, etc. For example, we may add simple tags for sentiment and impact, as well as show news headlines on the dashboard, symbol pages, and in the approvals view. Templates can include rules such as an earnings lockout.

Creative Components

- Group trading: This will allow users to mutually trade within a group. Group trades would come from the group's balance (which is defined as the sum of all the users' individual account balances), and a group trade's profit/loss (P/L) contributes towards that group's P/L (but does not affect individual account balances of any user in said group).
- News: Notable events that may influence stock prices.
- Leaderboard that shows the most profitable team or individual over a given time frame.

Usefulness

Our paper trading app will serve as individual and group trading practice. Individuals get access to reusable templates, backtests on real data, stress tests on random series, news and earnings reports, and simple risk guards like max loss and

position caps. Groups will have a shared account with the following roles: the owner, trader, risk manager, and an additional role titled the viewer. They will also have:

- Approvals for big and/or risky orders
- Trails to monitor activities
- Ticket searching capabilities
- Chart viewing
- The ability to place market/limit/stop orders
- Position tracking
- View trade history with exports.

The tools that will come along with our app support quick solo iteration and optional team controls to build repeatable, news-aware, and risk-managed habits.

Realness

Our project will use a combination of real and simulated market datasets. For real data, we'll use historical stock prices from Yahoo Finance, accessed through the open-source yfinance library. The data will be stored in CSV format with three columns: time, stock, and price. A typical dataset contains around 3–4 million rows. This data captures authentic market movements for different companies and time periods.

To complement this, we'll incorporate fundamental financial data from the U.S. SEC EDGAR "Company Facts" API, which provides corporate filings and metrics such as revenue, net income, and EPS in JSON/CSV format. Each company has hundreds of numerical fields reported quarterly, resulting in several million records across thousands of firms.

Also, we will be scraping news articles from the top news stations that are related to the stock market or specific companies. We will be using newsdata.io, which provides an API for searching and getting news articles in CSV format. There are a few hundred news articles related to the market written each day.

Finally, we'll generate simulated random-walk price series with parameters learned from the real Yahoo Finance data. These simulations allow users to trade in controlled or hypothetical environments while keeping market behavior statistically realistic. Together, these datasets give our platform both authenticity and flexibility for modeling diverse market conditions.

Functionality

Below are the tables we expect to fully implement and use:

- **users**: id, email, password_hash, created_at
- **accounts**: id, account_type, name, starting_cash, created_at
- **account_memberships**: account_id, user_id, role

- **tickers**: symbol, name, asset_type
- **price_bars**: ticker, time, open, high, low, close, volume, source (REAL from Yahoo Finance; SIM from generator)
- **groups**: id, name, created_at, created_by
- **group_memberships**: group_id, user_id, role(owner|manager|member|viewer), added_at
- **news_articles**: id, published_at, source, title, url, sentiment, impact_tags[]
- **news_ticker_map**: article_id, ticker
- **user_news**: user_id, ticker
- **transactions**: id, account_id, time, ticker, side(BUY/SELL), qty, price, kind(ORDER|FILL), status, requested_by, approved_by NULLABLE

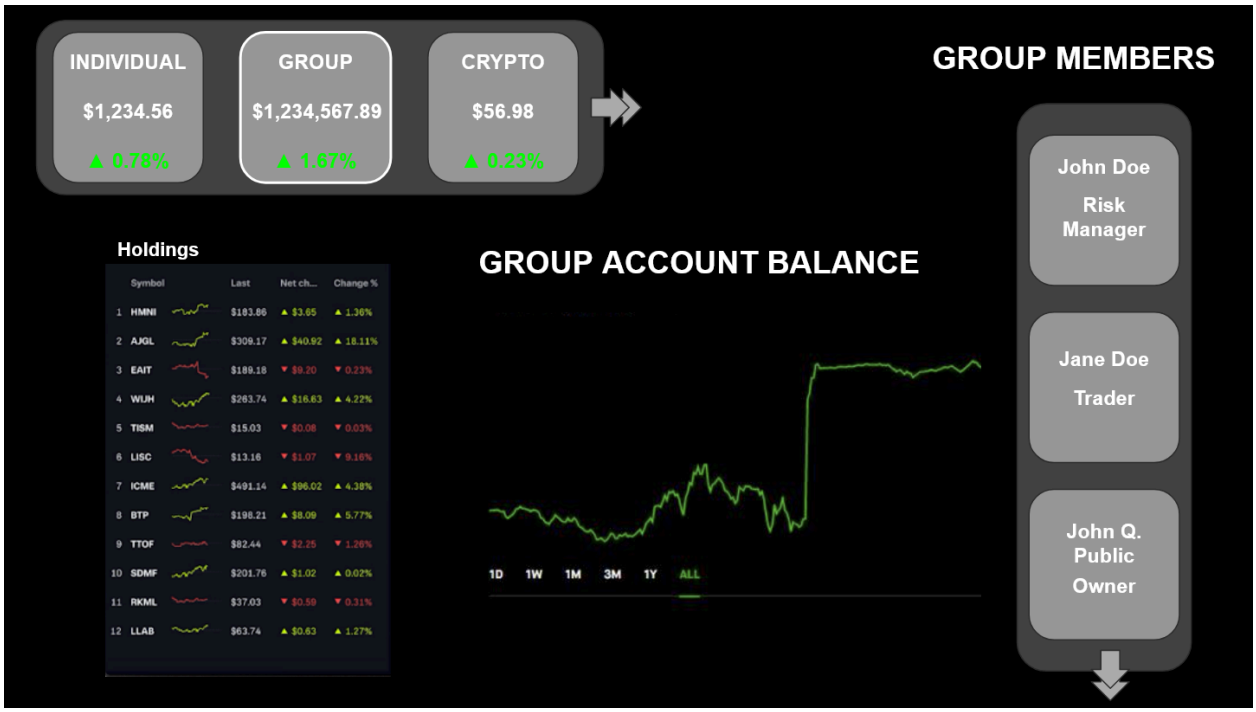
Below are possible changes that can be made:

- **Users** can be added to/updated/deleted. They would be added when making an account, deleted if deleting an account, and updated if a user changes any info (like their password)
- **Accounts** can be added to/updated/deleted. Added/deleted is when user makes/deletes an account and updated if they change their name or account type
- **Account_memberships** can be added to when user assigned to an account, updated if their role changes, and deleted if a user is removed from account or their account is closed
- **Tickers** can be added/deleted for a new ticker or if a ticker is removed, and updated for any ticker name changes
- **Price_bars** can be added to for new ticker data.
- **Groups** can be added to if there's a new group, updated if the name is changed, and deleted if a group is deleted
- **Group_memberships** can be added to when a user joins a group, updated if a role changes, and deleted from if a user leaves a group, or a group is deleted
- **User_news** can be added/deleted. User_news is added when a user flags their first stock to be tracked. The User_news can be deleted from/inserted into every time a user adds or deletes a stock from their watch list, or a stock ticker they have on their watch list is changed (we would delete old tuple then insert new tuple with updated ticker). The User_news is deleted if the user deletes their account or if the user deletes all the stocks from their watch list.
- **News_ticker_maps** can be added or updated. The maps are created when a news article is added. The maps are updated if the article they are mapping is updated or if the ticker its linked to's name is changed.
- **News_articles** can be added/updated. Articles are added when a new article relating to the stock market or a company is published by a news source we

track. Articles are updated if edits are made to an article on the news source we track.

- **Transactions** can be added/updated. They would be added when a trade is placed. The transactions would be updated when the trade is executed/filled/not filled/ approved/denied/or cancelled.

UI Mock-Up



Work Distribution

- Member 1 (David)
    - Responsible for the following tables:
        - Users
        - Accounts
        - Account_memberships
    - Responsible for the following views:
        - User_accounts_view
        - Pending_approvals_view
    - Responsible for the following triggers and/or routines:
        - Role validation on account_memberships
    - Responsible for the following constraints/indexes
        - Unique users.email
        - Composite PK (account_id, user_id) on account_memberships
        - Composite PK (group_id, user_id) on group_memberships

    - Responsible for the following imports/exports:
        - Seed script for users, accounts, memberships, and groups
    - General coding responsibilities:
        - Write migrations for identity and membership tables
        - Build stored procedure/trigger to validate roles on insert/update
        - Implement seed script to auto-generate sample users, accounts, and groups
        - Write backend routes/queries for listing user accounts and pending approvals
- Member 2
    - Responsible for the following tables:
        - Tickers
        - Price_bars
    - Responsible for the following views:
        - Latest_close_per_ticker
        - Account_positions_view
    - Responsible for the following triggers and/or routines:
        - Refresh positions on fills
    - Responsible for the following constraints/indexes:
        - Index (ticker, time)
        - Composite PK (ticker, time) on price_bars
    - Responsible for the following imports/exports:
        - CSV loader for tickers and price bars
    - General coding responsibilities:
        - Build CSV import utility for loading tickers and price_bars
        - Write stored routine/trigger to update positions on transaction fills
        - Implement backend API endpoints to fetch latest close and historical OHLCV data
        - Optimize queries with appropriate indexing and test with EXPLAIN
- Member 3
    - Responsible for the following tables:
        - Transactions
        - Groups
        - Group_memberships
    - Responsible for the following views:
        - Open_orders_view
        - Account_pnl_basic
    - Responsible for the following triggers and/or routines:
        - Order status transition routine
    - Responsible for the following constraints/indexes:

        - FK transactions.account_id → accounts.id
        - FK transactions.ticker → tickers.symbol
    - Responsible for the following imports/exports:
        - Export trades to CSV
    - General coding responsibilities:
        - Implement order/transaction lifecycle state machine (order → fill → close)
        - Write triggers or routines to reject invalid transitions
        - Build API endpoints for creating, canceling, and viewing orders
        - Develop CSV export utility for trades by account/date range
        - Unit tests for order state validation and error handling
- Member 4
    - Responsible for the following tables:
        - News_articles
        - News_ticker_map
    - Responsible for the following views:
        - News_by_symbol_view
        - News_sentiment_view
    - Responsible for the following triggers and/or routines:
        - Auto-populate news_ticker_map on insert
    - Responsible for the following constraints/indexes:
        - FK news_ticker_map.article_id → news_articles.id
        - FK news_ticker_map.ticker → tickers.symbol
    - Responsible for the following imports/exports:
        - CSV import for news
    - General coding responsibilities:
        - Write CSV import utility for news_articles and mapping to tickers
        - Implement routine to detect tickers in headlines and populate news_ticker_map
        - Build backend API for querying news by symbol and sentiment
        - Write sentiment aggregation function/view for reporting
        - Add unit tests for news → ticker linking logic
- Shared Integration (all members)
    - Consolidated ERD and migration order; cross-team foreign key checks.
    - Basic performance verification on primary join paths (EXPLAIN/ANALYZE).