To democratize stock market education and trading by creating a collaborative platform where users can learn, share, and trade with confidence through a supportive community.

Core Features

• User login/signup, google & facebook.

• Connect login with brokerage account.

• Some anonymity for user profile. Use profile image. Show whom he follows and followers to the user.

• Show own feed/posts of the user.

• Read only access to brokerage account.

• Zeroda, icici, hdfc etc. Top 4 brokerages.

• Display table/some UI for all the stocks and options of the account.

• Link the stock in the table of the person sharing the account to google stock chart to provide insights about the stock.

• Stock ticker information. Search feature as main option on the app

• E.g. Zomato real time stock price

• Posts related to a stock.

• Social feed.

• Show all posts of the people the user follows

• Able to do a new post by the user himself.

• Post, text with image and maybe a link to youtube videos.

• # tags and annotations to refer to a user.

• Replies to posts.

• Likes

• Real-Time Alerts & Notifications

• (P1)Market Updates: Live market news and price movements for stocks, ETFs, and more. XSS feed to get market updates from 3rd party apis.

b.

• We will need Cloudflare WAF to thwart attacks. The Securities and Exchange Board of India released the Cyber Security & Cyber Resilience Framework for Stock Brokers/Depository Participants in 2018.

• adheres to SEBI's regulations for internet-based trading (IBT). We need logos and branding to ensure that customers are confident to use our app without any concerns.

• AI integration to convert voice to text and integrate with open ai to generate results specific to stock questions.

• Social Trading Platform

• Trade Sharing: Users can share their real trades and strategies with others in the community.

• ~~Copy Trading: Users can follow and replicate trades from experienced traders, fostering a learning-by-doing approach.~~

• ~~Leaderboards: Highlight top-performing users based on trade success, providing credibility and motivation.~~

2

• (P1)Trade Alerts: Notify users when followed traders make a move.

• (P2)Discussion Forums

• Topic-specific discussion boards for stock ideas, market trends, and

strategies.

• Moderated by experts to ensure quality and relevance.

• (P0)Verified User Base

• Verification of trade activity ensures authenticity and trustworthiness.

• ~~Gamification~~

• ~~Badges & Rewards: Users earn badges for trade successes, contributions to discussions, and mentorship roles.~~

• Challenges and competitions to engage the community and promote activity.

• Engagement activity for the users based on xss feed of the stock/trendy stock/

Additional Key Features

• Educational Content

• Tutorials, blogs, and videos tailored to novice, intermediate, and advanced traders.

• In-app quizzes and certifications for skill-building.

• Portfolio Tracking

• Integrated portfolio management tools to track performance and analyze trends.

• Data-Driven Insights

• AI-driven sentiment analysis on trending stocks and community discussions.

• Integration with Brokers

• Secure APIs to connect existing brokerage accounts for seamless trade execution.

• Privacy Controls

• Users can opt to share anonymously or control visibility of their trades and discussions.

Business Model

• Freemium Subscription

• Free Tier: Basic features like trade sharing, forums, and alerts.

• Premium Tier: Advanced analytics, exclusive leaderboards, and direct mentorship.

• Affiliate Partnerships

• Commissions from brokerage integrations and referrals.

• Sponsored Content

• Advertisements from financial institutions or stock-related tools.

• Gamified Revenue

• In-app purchases for additional features or gamified rewards.

Competitive Edge

• Community-Centric Approach

• Focus on authentic, verified trading discussions sets it apart from generic forums.

• Accessibility & Gamification

• Engages younger demographics through interactive and gamified learning.

• Scalable Infrastructure

• Built for global user adoption with multi-language and multi-market support.

Development Roadmap

• Phase 1: MVP Launch (6 Months)

• Features: Trade sharing, basic forums, real-time alerts, and leaderboards.

• Phase 2: Expansion (12 Months)

• Features: Copy trading, brokerage integration, gamification, and premium subscription.

• Phase 3: AI-Driven Insights (18 Months)

• Sentiment analysis and personalized recommendations.

Ask to Investors

• Funding Required: $1M for a 12-month runway to build and market the MVP.

• Projected ROI: 5x within 3 years based on user acquisition, premium subscriptions, and affiliate revenue.

This pitch balances community-building features with revenue-generating opportunities, aiming to create an engaging, scalable, and profitable platform.

New features:

• We will need Cloudflare WAF to thwart attacks. The Securities and Exchange Board of India released the Cyber Security & Cyber Resilience Framework for Stock Brokers/Depository Participants in 2018.

• adheres to SEBI's regulations for internet-based trading (IBT). We need logos and branding to ensure that customers are confident to use our app without any concerns.

• Create an social media feed, which includes posts for stock trades. Generic communication involving hash tags, annotations and images in the post.

• Basic login integration, for users to login.

———————————————————————————————————————————————————————————————————————————————————————————————

• User login, google & facebook.

• Connect login with brokerage account. (Zerodha,groww, icici,hdfc)

• Some anonymity for user profile. Use profile image. Show whom he follows and followers to the user.

• Show own feed/posts of the user.

Onboarding flow,

— users comes sees signup page/ login via social platform page

— chooses 1 , on sign up , gives details, give password signup , or connect to soc

Platform login, will get data

— asks users about interests (which stocks, which model (small cap, mid cap, large cap) , investing mode(high risk, low risk)

— based on above suggest users already onboarded ,to follow ,

— connect to brokerage account to fetch data and display

—

——————————————————————————————————————————————————————————————————————————————————————————————

• Read only access to brokerage account.

• • Zeroda, icici, hdfc etc. Top 4 brokerages.

• Display table/some UI for all the stocks and options of the account.

• Link the stock in the table of the person sharing the account to google stock chart to provide insights about the stock.

Stock ticker information. Search feature as main option on the app

• E.g. Zomato real time stock price

• Posts related to a stock.

**1. User Logs into the App (Authentication and Authorization)**

**• Step 1:** User logs into the app using Google or Facebook authentication through **Auth0** (as discussed earlier).

**• Step 2:** Once logged in, the app checks if the user has linked a brokerage account (e.g., Zerodha, ICICI, HDFC). If not, prompt the user to link their brokerage account.

**2. Link Brokerage Account**

**• Step 1:** The user selects their brokerage (e.g., Zerodha, ICICI, HDFC) from a list of supported brokers.

**• Step 2:** Redirect the user to the respective brokerage's API authorization page (if available) or prompt the user to enter their brokerage login credentials (for read-only access).

**◦ Zerodha**: Use **Zerodha's Kite Connect API** (requires API key and secret).

**◦ ICICI**: Use the **ICICI Direct API** (if available or redirect to login page).

**◦ HDFC**: Use the **HDFC Securities API** (if available or redirect to login page).

**• Step 3:** Once authenticated, the app retrieves an **API token** for read-only access to the brokerage account.

**• Step 4:** The app stores this token securely to make future API calls on behalf of the user.

**3. Fetch Account Information**

**• Step 1:** Once the brokerage account is linked, the app makes a **read-only API call** to the brokerage's server to fetch a list of stocks and options held by the user.

**◦ Zerodha API**: Use the **Kite Connect API** to fetch the portfolio details.

**◦ ICICI API**: Fetch the portfolio data using the **ICICI Direct API**.

**◦ HDFC API**: Fetch the portfolio data using the **HDFC Securities API**.

**• Step 2:** Parse the response to extract details like:

◦ Stock symbol (e.g., RELIANCE, TCS)

◦ Stock type (e.g., equity, options)

◦ Quantity held

◦ Purchase price

◦ Current price

◦ Market value

**• Step 3:** Store this data temporarily for display in the app.

**4. Display Portfolio in a Table/UI**

**• Step 1:** The app creates a **UI table** displaying the stocks and options the user holds in their brokerage account. For each stock, the table will show:

**◦ Stock Name** (e.g., RELIANCE, TCS)

**◦ Quantity** held

**◦ Purchase Price**

**◦ Current Price**

**◦ Market Value** (calculated as Current Price \* Quantity)

**◦ P&L (Profit & Loss)** (calculated as (Current Price - Purchase Price) \* Quantity)

**• Step 2:** Add additional columns for **Actions** like:

◦ View **Google Stock Chart** for each stock.

◦ View **Stock Insights** (e.g., news, trends, etc.).

**5. Link Stock to Google Stock Chart**

**• Step 1:** For each stock listed in the table, the app provides a clickable link to the **Google Stock Chart**.

◦ This can be done by generating a URL to Google Finance or another service (e.g., Yahoo Finance) with the stock's symbol as a query parameter.

▪ Example URL for **Reliance Industries**:  
https://www.google.com/finance/quote/RELIANCE:NSE?sa=X&ved=2ahUKEwiDjoqIvcDdAhXq8HMBHQIZD2sQ3N0wE3oECAYQCA#

**• Step 2:** When the user clicks on a stock, the app redirects them to the **Google Stock Chart** page that provides an interactive chart, historical data, and other insights about the stock.

**• Step 3:** You may optionally embed the Google Stock chart directly into the app using an **iframe** or external widget, depending on Google’s terms of service.

**6. Display Insights and Analysis for Stocks**

**• Step 1:** For each stock, provide additional **insights**:

◦ Latest **news** related to the stock.

**◦ Technical analysis** (e.g., moving averages, RSI).

**◦ Stock performance** over different time periods (1 day, 1 week, 1 month, etc.).

**◦ Social Sentiment** (using APIs that track social media mentions or sentiment analysis).

**• Step 2:** These insights can be fetched from public APIs or news sources:

**◦ News API** (for news updates).

**◦ Yahoo Finance API** (for stock analysis).

**◦ Twitter or Sentiment APIs** (to track stock mentions).

**7. Syncing Data Periodically**

**• Step 1:** Schedule periodic updates (e.g., every 15 minutes) to sync the portfolio data with the brokerage accounts. This can be done using backend cron jobs.

**• Step 2:** If the user makes any changes in the brokerage account (e.g., buying/selling stocks), the app should automatically update the user’s portfolio.

**8. Error Handling and Notifications**

**• Step 1:** If the API calls to the brokerage services fail (e.g., due to network issues or expired tokens), display a friendly error message like:

◦ "Unable to fetch your stock data right now. Please try again later."

**• Step 2:** If the user’s token expires or they need to re-link their brokerage account, notify them with:

◦ "Your session has expired. Please re-link your brokerage account to continue."

**9. Logging Out or Disconnecting Brokerage Account**

**• Step 1:** Allow the user to disconnect their brokerage account from the app through the settings.

**• Step 2:** If the user logs out of the app, ensure their session with the brokerage API is also terminated.

———————————————————————————————————————————————————————————————————————————————————————————

• Social feed.

• Show all posts of the people the user follows

• Able to do a new post by the user himself.

• Post, text with image and maybe a link to youtube videos.

• # tags and annotations to refer to a user.

• Replies to posts.

• Likes

**1. Show All Posts of the People the User Follows**

**Frontend (React Native):**

**1 Feed Display**:

**◦ Design a vertically scrolling feed**:

▪ Each feed item includes:

▪ Profile picture and name of the poster.

▪ Text content of the post.

▪ Image or video (if attached).

▪ Link previews (e.g., YouTube thumbnails for video links).

▪ Time of posting (e.g., "5m ago", "2h ago").

▪ Action buttons for **Like**, **Reply**, and **Share**.

▪ Number of likes and replies displayed under the post.

◦ Use a **Lazy Loading/Infinite Scroll** mechanism to load posts in batches.

**2 API Call**:

◦ On page load or refresh, call an API endpoint like GET /feed to fetch posts by people the user follows.

◦ Include pagination support to fetch additional posts when the user scrolls down.

**2. Create a New Post by the User**

**Frontend (React Native):**

**1 UI for Creating a Post**:

◦ Provide a text box for entering post content.

◦ Include an **image upload option** using file pickers (for photos or videos).

◦ Optionally, allow pasting links (e.g., YouTube videos).

◦ Display **suggested hashtags** and allow tagging other users with @username.

**2 Validation**:

◦ Ensure at least one of these is provided: text, image, or video.

◦ Limit post length to, e.g., 280 characters.

**3 API Call**:

◦ On submission, call an API endpoint like POST /posts with:

▪ content: Text content of the post.

▪ media: Uploaded image or video (multipart).

▪ tags: Array of hashtags.

▪ mentions: Array of mentioned user IDs.

**3. Display Hashtags and Mentions**

**Frontend:**

**1 Highlight Hashtags**:

◦ Parse the post content for #tags and format them as clickable links.

◦ Clicking a hashtag should navigate to a **hashtag feed** showing all posts with that tag.

**2 Highlight Mentions**:

◦ Parse the post content for @mentions and format them as clickable links.

◦ Clicking a mention should navigate to the mentioned user’s profile.

**4. Replies to Posts**

**Frontend (React Native):**

**1 Reply Input**:

◦ Provide a "Reply" button under each post.

◦ On click, display a text box for entering the reply.

◦ Allow tagging other users in replies.

**2 Reply Display**:

◦ Show replies as a nested list under the post.

◦ Use indentation or a collapsible view for hierarchical replies.

**5. Likes**

**Frontend (React Native):**

**1 Like Button**:

◦ Add a like button (e.g., a heart icon) under each post.

◦ Display the number of likes beside the button.

**2 Toggle Behavior**:

◦ On tap:

▪ If the post is already liked by the user, send an **unlike** request.

▪ Otherwise, send a **like** request.

————————————————————————————————————————————————————————————————————————————————————————————————————————————————

Real-Time Alerts & Notifications

• (P1)Market Updates: Live market news and price movements for stocks, ETFs, and more. XSS feed to get market updates from 3rd party apis.

b.

**————————————————————————————————————————————————————————————————————————————————————————————————————————**

**1. User Login (Google & Facebook)**

**UX/UI Requirements**

**• Login Screen:**

◦ Display options for **Google Login** and **Facebook Login** clearly on the screen.

◦ Each button should have its respective logo to make the option visually clear and distinguishable.

◦ Provide a seamless user experience, where logging in through Google/Facebook is the main flow.

◦ Have a "Forgot Password" and "Sign Up" option visible in case the user is new or has issues logging in.

◦ Include a loading indicator while authentication is in progress.

**React Native Implementation**

• Use libraries like react-native-firebase for Google login or react-native-facebook-login for Facebook login.

• Implement methods that call the relevant authentication APIs to initiate Google and Facebook login flows.

• After successful login, fetch user data (e.g., name, email, profile picture) from Google/Facebook.

**Backend (Java) Implementation**

• Implement an API endpoint /auth/login that accepts social login tokens from Google and Facebook.

• On successful login, use **OAuth 2.0** to validate the token and fetch the user profile from the respective platform.

• Create or update a user record in the database (if not already existing) with details like email, name, profile\_picture, and a unique user\_id.

**2. Connect Login with Brokerage Account**

**UX/UI Requirements**

**• Brokerage Account Connection Screen:**

◦ Provide an option for users to link their brokerage account via an easy-to-use "Connect Brokerage" button.

◦ This screen should guide the user on how to connect the brokerage account, including entering credentials (if required).

◦ Display any required details from the brokerage account (such as account type, recent trades, etc.) once the connection is successful.

◦ Provide a “Disconnect” option to allow users to unlink the brokerage account when desired.

◦ For security, use a modal or secure authentication method to connect with the brokerage account (i.e., two-factor authentication).

**React Native Implementation**

• Use OAuth or API integrations provided by the brokerage platform to authenticate and link the user's brokerage account.

• Display the brokerage account details fetched after connection in a separate UI screen, ensuring a secure flow.

• Provide error handling in case the connection fails (e.g., invalid credentials, API failure).

**Backend (Java) Implementation**

• Implement a secure API /auth/connect-brokerage where users can send their credentials to link their brokerage account.

• Ensure secure communication via encryption (HTTPS).

• Store the brokerage account connection details (token or authentication key) securely in the database.

**3. Anonymity for User Profile**

**UX/UI Requirements**

**• Profile Screen:**

◦ Show the user's **profile image**, but keep user details like **full name** and **email** private to maintain anonymity (display only the username or handle).

◦ Display a list of **followers** and **whom the user follows**. This list should be visible to the user but kept private to others, ensuring the user has control over who sees these connections.

◦ Allow users to edit their **profile image**, **display name**, and **username** but without showing personal details.

**React Native Implementation**

• Use a user settings screen to enable users to edit their profile and manage privacy settings.

• Implement a secure way to upload and display the user's profile picture (e.g., use cloud storage).

• Display "followers" and "following" with minimal information, possibly showing only avatars and usernames (without personal info).

**Backend (Java) Implementation**

• Store minimal user profile details in the database, including:

◦ user\_id (unique identifier)

◦ profile\_picture\_url (URL to the profile image)

◦ username (publicly visible name)

◦ followers and following lists, referencing other user\_ids.

• Implement privacy logic to ensure sensitive information like email, real name, and any trading history remains private.

• API endpoints for updating and fetching profile data should be designed to keep user anonymity intact.

**4. Show Own Feed/Posts of the User**

**UX/UI Requirements**

**• Feed Screen:**

◦ Display the user's own posts/feed in a vertical scroll format (like Instagram or Twitter).

◦ Posts should include **content** (text, images, videos) and an **action bar** with options like "like", "comment", and "share".

◦ Provide a clear and simple way to **create new posts**, allowing the user to either post text, images, or video (depending on your platform’s features).

◦ Allow users to **edit** or **delete** their posts by providing options within the post action menu.

**React Native Implementation**

• Implement a scrollable feed using components like FlatList for performance.

• Provide an input area where users can post updates (text and media).

• Add an action bar with options such as "like", "comment", and "share" for interaction.

• Handle the creation of posts with state management (e.g., using useState and useEffect hooks).

**Backend (Java) Implementation**

• Create endpoints like /post/create, /post/edit, /post/delete, and /post/fetch to interact with the feed.

• Ensure data consistency by storing the post's content, media (URLs), timestamps, and interaction data (e.g., likes, comments).

• Store the posts in a database table (posts) with a reference to the user\_id who created it.

**Additional Considerations for All Features**

**• Security and Privacy**:

◦ Implement **SSL encryption** for all API calls.

◦ For social login, use OAuth tokens and ensure they are valid before performing any actions.

◦ Ensure sensitive information like brokerage details and personal information is encrypted in the database.

**• Scalability**:

◦ Design backend APIs to handle large-scale data efficiently (pagination for posts, followers/following lists).

◦ Use a NoSQL or relational database based on performance requirements (e.g., for user data and posts).

**• Testing**:

◦ Conduct both **unit testing** for backend services (e.g., validating user login, post creation) and **integration testing** for React Native features (e.g., login flows and post interactions).

Backend use auth0, store users details , have its own id and recognise from second time,

Backend one hour is enough for this

(Forgot password and signup to be added)

-