



Stock Market Web Application Using ReactJS: Design, Implementation, and Evaluation

Patil Rushikesh Manish

Student, Department of Computer Science and Engineering
Parul Institute of Computer Science and Technology, Parul University, Vadodara
Under the Guidance of Prof. Keya Patel, Assistant Professor

Abstract: The rapid digitisation of financial markets has created demand for intuitive, high-performance web-based trading platforms accessible to a broad user base. This paper presents the design, architecture, and implementation of a real-time stock market web application developed using ReactJS as the core frontend framework, integrated with Node.js and MongoDB for backend and data-persistence services. The platform replicates the core functionality of production-grade brokerage systems such as Zerodha and Groww, offering features including live stock price display, portfolio management, mutual fund investment, SIP setup, watchlist tracking, and fund deposit and withdrawal. A component-driven Single Page Application (SPA) architecture ensures smooth navigation and real-time state updates without full-page reloads. The system was evaluated against fourteen functional test cases covering user registration, transaction management, gain/loss computation, and notification accuracy, all of which passed successfully. The paper discusses the system's three-tier architecture, key UML artefacts, functional and non-functional requirements, testing strategy, and future enhancements including AI-driven recommendations and blockchain-based settlement.

Index Terms — ReactJS, Stock Market, Single Page Application, Portfolio Management, Real-Time Data, Node.js, MongoDB, Mutual Funds, SIP, Web Application.

I. INTRODUCTION

The global financial ecosystem increasingly relies on digital infrastructure to facilitate equity trading, mutual fund investment, and portfolio management. Traditional broker-mediated trading, which depended on physical presence and telephone-based order routing, has been displaced by browser-accessible platforms that connect retail investors directly to exchanges such as the National Stock Exchange (NSE) and the Bombay Stock Exchange (BSE). This democratisation of market access has been enabled by advances in web technologies, real-time data streaming, and cloud computing. Despite the availability of commercial platforms, many lack the transparency, educational scaffolding, and simplicity required by novice investors. There is a recognised need for an open, lightweight, and pedagogically useful implementation that demonstrates how a modern trading application can be architected using contemporary web technologies. This paper addresses that need by documenting the end-to-end development of a stock market web application built with ReactJS.

The remainder of this paper is structured as follows: Section 2 reviews the problem context and motivation; Section 3 presents the system architecture and planning; Section 4 details the system design artefacts; Section 5 describes the implementation and test results; Section 6 draws conclusions; and Section 7 outlines the future scope.

II. PROBLEM DEFINITION AND MOTIVATION

Traditional stock trading suffers from a series of structural inefficiencies: reliance on intermediaries inflates costs and introduces latency; access to accurate real-time market data is restricted; and user interfaces on many existing platforms are complex, deterring first-time investors. Specific problems addressed by this work include:

Fragmented Investment View: Investors typically operate across multiple applications to track stocks, mutual funds, and cash balances, resulting in cognitive overhead and delayed decision-making.

Complexity of Existing Interfaces: Professional trading terminals expose raw financial data without adequate visual abstractions, making them inaccessible to beginners.

Delayed Data and Poor Responsiveness: Many platforms rely on periodic page refreshes rather than real-time state synchronisation, causing investors to act on stale information.

Limited Transaction Transparency: Deposit, withdrawal, and trade histories are often buried within multiple navigation layers, reducing user trust.

Absence of Educational Simulation: Few platforms offer a risk-free simulation environment in which new investors can practise trading without financial exposure.

The goal of this project is to design and implement a unified, responsive, and educationally oriented stock market platform that resolves the above limitations while adhering to industry-standard software engineering practices.

III. SYSTEM ARCHITECTURE AND PLANNING

3.1 Three-Tier Architecture

The application follows a three-tier client-server architecture comprising a presentation layer, an application layer, and a data layer.

Table 1: Three-Tier Architecture Overview

Tier	Technology	Responsibility
Presentation	ReactJS, HTML5, CSS3, Bootstrap	Dynamic UI rendering, state management via Context API / Redux, real-time stock data display
Application	Node.js, Express.js	Business logic, REST API endpoints, authentication, external API integration
Data	MongoDB (NoSQL)	User profiles, stock holdings, transaction history, mutual fund records

3.2 Technology Stack

ReactJS was selected as the frontend framework on account of its component-based architecture, virtual DOM diffing for efficient re-rendering, and the rich ecosystem of reusable UI libraries. The SPA paradigm eliminates full-page reloads, providing a desktop-application experience within a browser. State management via Context API ensures that real-time data changes — such as updated stock prices and gain/loss calculations — propagate instantly across all subscribed components.

Node.js with Express.js was chosen for the backend because its non-blocking event-loop model is well-suited to I/O-intensive workloads such as concurrent API calls to external financial data providers. MongoDB was selected over relational databases for its schema flexibility, which accommodates the heterogeneous data structures inherent to financial instruments (stocks, mutual funds, SIPs).

3.3 Project Development Timeline

The project was executed across nine weeks following the Software Development Life Cycle (SDLC):

Table 2: Project Development Timeline

Phase	Week	Activity	Duration
1	Week 1	Requirement Analysis and Problem Identification	
2	Week 2	Feasibility Study (Technical, Economic, Operational)	
3	Weeks 3–4	System Design: Architecture, Database Schema, UI/UX	
4	Week 5	Frontend Development using ReactJS	
5	Week 6	Backend Development: APIs and Server Logic (Node.js)	
6	Week 7	Database Integration (MongoDB)	
7	Week 8	API Integration, Testing, and Deployment	
8	Week 9	Documentation and Final Report Preparation	

IV. SYSTEM DESIGN

4.1 System Modules

The application is decomposed into eight interacting modules, each with a distinct responsibility boundary:

User Management Module: Handles registration, authentication, and profile management using JWT-based token authentication and encrypted password storage.

Stock Data Management Module: Continuously fetches and caches real-time price data from external financial APIs, providing live feeds for NSE/BSE-listed equities.

Trading Module: Implements order placement (buy/sell), order validation against available balance, and real-time portfolio update upon execution.

Portfolio Management Module: Maintains a per-user record of stock holdings, computes current market value, and calculates unrealised gain/loss.

Mutual Fund Module: Supports investment in curated fund categories (index, bluechip, small-cap growth) via SIP or lump-sum mechanisms.

Transaction Management Module: Records all financial events (trades, deposits, withdrawals) with timestamps and status, forming an immutable ledger.

Analytics and Reporting Module: Renders candlestick and line charts, displays major indices (NIFTY 50, SENSEX, BANK NIFTY), and summarises performance.

Notification Module: Delivers event-driven feedback (transaction success/failure) via in-app alerts after backend validation completes.

4.2 Functional Requirements

The principal functional requirements of the system are:

- Secure user registration and JWT-based login/logout.
- Real-time display of stock prices, percentage change, volume, and market capitalisation.
- Buy and sell order placement with immediate portfolio and balance update.
- Portfolio dashboard showing holdings, average buy price, current value, and gain/loss.
- Watchlist for tracking selected stocks without committing capital.
- Mutual fund investment via SIP (monthly) or lump-sum, with fund performance display.
- Fund deposit and withdrawal with full transaction history and status tracking.
- Transaction success and failure notifications delivered immediately post-execution.

4.3 Non-Functional Requirements

Table 3: Non-Functional Requirements

Attribute	Requirement
Performance	Sub-second response for API calls; real-time UI updates without full reload
Scalability	Horizontal scaling via cloud deployment (Vercel/Netlify frontend, cloud backend)
Security	JWT authentication, SSL/TLS encryption, input sanitisation, secure API gateways
Reliability	High availability with minimal downtime; backup and disaster recovery provisions
Usability	Responsive design across desktop, tablet, and mobile viewports
Maintainability	Modular component structure enabling independent update and testing of modules

4.4 UML Design Artefacts

Use Case Diagram: The primary actor is the investor, who interacts with the system through login, stock search, portfolio viewing, order placement, watchlist management, and fund operations. An admin actor manages users and monitors system activity. An external Stock Broker actor supplies market data.

Sequence Diagram: The buy-stock sequence initiates when the user clicks the buy button on the ReactJS interface. The frontend dispatches a request to the Node.js backend, which queries MongoDB to verify user balance and stock holdings. Concurrently, the latest price is fetched from the external API. Upon successful validation, the trade is committed, the portfolio is updated in MongoDB, and a confirmation is returned to the UI.

Class Diagram: Core classes include User (userID, name, email, password), Stock (stockID, symbol, name, price), Portfolio (holdings per user), Transaction (type, quantity, price, date, status), and Watchlist (stockID references per user). Associations reflect a user owning one portfolio, performing many transactions, and maintaining one watchlist.

E-R Diagram: The Investor entity participates in an M:M invest relationship with Investment, which has a 1:M relationship with Transactions. Each Investor also owns a Portfolio (1:M). Income and Expense entities are related to both Investment and Transactions to enable comprehensive financial tracking.

V. IMPLEMENTATION AND TESTING

5.1 Key Implemented Features

User Registration and Authentication: A Create Account form collects email, phone number, and password (minimum eight characters), with a confirm-password field. Backend validation rejects duplicate emails and weak passwords before issuing a JWT.

Dashboard: The central hub displays live NIFTY 50, SENSEX, and BANK NIFTY index values alongside four summary cards: total stock value, aggregate gain/loss, total mutual funds invested, and number of distinct holdings. Top gainers and losers are highlighted with colour-coded percentage badges.

All Stocks Listing: A searchable and sortable table lists all available equities with columns for symbol, company name, current price, daily change, volume, market capitalisation, and P/E ratio. A View button opens the detailed trade chart.

Trade Chart: Individual stock detail panels render a time-series line chart (17 Feb 2026 to 18 Mar 2026 in the evaluated build) alongside day high, day low, volume, and market cap. A tabbed Buy/Sell widget accepts quantity and price inputs and executes the order via the Place BUY/SELL Order button.

Portfolio Management: The My Portfolio screen tabulates each holding's symbol, company, quantity, average buy price, current price, total current value, gain/loss in currency, and gain/loss percentage.

Mutual Fund Investment: Three fund categories are presented — Nifty 50 Index Fund, Bharat Bluechip Fund, and Small Cap Growth Fund — each displaying 1Y/5Y returns, expense ratio, and total AUM. A modal collects investment type (SIP Monthly vs. One-time Lumpsum) and amount.

Banking Module: The My Bank screen displays available cash and exposes Deposit Funds and Withdraw Funds controls. A Recent Transactions ledger records every monetary event with type, amount, timestamp, and completion status.

Notifications: Transaction success and failure notifications appear as in-app alerts immediately following backend validation, providing event-driven user feedback without requiring manual page interrogation.

5.2 Test Cases and Results

Fourteen test cases were executed covering the full functional scope of the application. All fourteen cases passed, confirming correctness of the implemented logic. A representative selection is presented in Table 4 below.

Table 4: Selected Test Cases and Results (All 14 cases passed)

TC-ID	Description	Input	Expected Output	Status
TC-001	User Registration	Valid email, phone, password	Account created; redirected to dashboard	Pass
TC-002	Dashboard Display	Logged-in user opens dashboard	Market indices, summary cards, gainers/losers displayed	Pass
TC-006	Gain/Loss Calculation	Buy price = Current price	Gain/Loss = 0.00, Gain% = 0.00	Pass
TC-007	Total Price Calculation	Qty=1, Price=8950.5 USD	Correct INR equivalent displayed	Pass
TC-009	Deposit Funds	Amount = 10,000	Balance increases by 10,000; success message shown	Pass
TC-011	Withdraw Validation	Amount = 5,000 (sufficient balance)	Balance decreases by 5,000; confirmation shown	Pass
TC-013	Transaction Success Notification	Successful BUY/SELL	Popup: Transaction Successful	Pass
TC-014	Failed Transaction Notification	BUY with insufficient balance	Popup: Transaction Failed / Insufficient Balance	Pass

5.3 Technology Comparison

Table 5 compares the adopted technology stack against alternative approaches evaluated during the feasibility phase.

Table 5: Technology Stack Comparison

Approach	Technology	Outcome
Traditional Web System	PHP + MySQL	Basic performance; limited real-time capability
Desktop Application	Java + SQL	Limited cross-device accessibility; high deployment overhead
Full-Stack Web (Adopted)	React + Node.js + MongoDB	High performance; real-time updates; scalable SPA

VI. CONCLUSION

This paper has presented the design and implementation of a full-featured, real-time stock market web application built with ReactJS, Node.js, and MongoDB. The system successfully addresses the key limitations of traditional trading platforms by providing a unified, responsive, and pedagogically accessible interface for stock trading, portfolio management, mutual fund investment, and banking operations.

The three-tier SPA architecture, combined with real-time API integration and JWT-based security, delivers a platform that is fast, secure, and scalable. All fourteen functional test cases passed, demonstrating the correctness and reliability of the implemented modules. The component-based ReactJS approach yielded a maintainable codebase in which individual UI modules can be updated or replaced independently.

The project confirms that modern web technologies are capable of delivering production-quality financial applications that are accessible to a broad and diverse user base, bridging the gap between complex financial data and user comprehension.

VII. FUTURE SCOPE

Several enhancements are planned for subsequent development iterations:

AI-Driven Recommendations: Integration of machine learning models for personalised stock suggestions, sentiment analysis of financial news, and predictive price analytics.

Blockchain Settlement: Adoption of distributed ledger technology to reduce settlement times, eliminate intermediaries, and provide cryptographic proof of all transactions.

Global Market Integration: Extension beyond NSE/BSE to include international exchanges, enabling portfolio diversification across US, European, and Asian markets.

Progressive Web App (PWA): Conversion of the web application into a PWA to enable offline capability, push notifications, and native-app-like installation on mobile devices.

Advanced Data Visualisation: Introduction of candlestick charts, technical indicators (RSI, MACD, Bollinger Bands), and customisable chart overlays for experienced traders.

Social Trading: Implementation of a community layer enabling users to follow, share, and replicate the trading strategies of experienced investors.

Enhanced Security: Biometric authentication, multi-factor authentication (MFA), and continuous anomaly detection to protect user accounts and financial data.

REFERENCES

- [1] Stockler Inc., "Stockler Official Documentation," stockler.com. [Online]. Available: <https://www.stockler.com/>
- [2] W3Schools, "HTML, CSS and JavaScript Tutorials," w3schools.com. [Online]. Available: <https://www.w3schools.com/>
- [3] Bootstrap, "Bootstrap 5 Official Documentation," getbootstrap.com. [Online]. Available: <https://getbootstrap.com>
- [4] Mozilla Developer Network, "Web Development Documentation," MDN Web Docs. [Online]. Available: <https://developer.mozilla.org/>
- [5] GitHub Inc., "GitHub Documentation," github.com. [Online]. Available: <https://docs.github.com/>
- [6] Lucide, "Lucide Icons," lucide.dev. [Online]. Available: <https://lucide.dev>
- [7] Google Developers, "Web Technologies and APIs," developers.google.com. [Online]. Available: <https://developers.google.com/>
- [8] Stack Overflow, "Programming Q&A Platform," stackoverflow.com. [Online]. Available: <https://stackoverflow.com/>
- [9] Eraser, "Eraser for Technical Design," eraser.io. [Online]. Available: <https://www.eraser.io>
- [10] OpenAI, "ChatGPT Documentation," OpenAI. [Online]. Available: <https://openai.com/>
- [11] React Documentation, "React - A JavaScript Library for Building User Interfaces," reactjs.org. [Online]. Available: <https://reactjs.org/>
- [12] MongoDB, "MongoDB Documentation," mongodb.com. [Online]. Available: <https://www.mongodb.com/docs/>