COLLEGE CODE : 9111

COLLEGE NAME : SRM Madurai College For Engineering And Technology

DEPARTMENT: Computer Science And Engineering

STUDENT NM-ID : 1AB1AA81B6B40D9B5C94D469B460D04A
1FFEB20A6540161A1F54E258167D1978
877B09181AE2E069AFF4C36AF16D1C4D
FF192F92083F64EB3754B0596244F018

ROLL NO ; 911123104040
911123104016
911123104032
911123104004

DATE : 15/09/2025

Completed the project named as phase 1

TECHNOLOGY PROJECT NAME : REAL TIME STOCK TICKER

SUBMITTED BY:

Rajesh Kumar M mobile no : 8524925826
Harish manikandan R mobile no : 6369119261
Nyson DM mobile no : 8682054707
Arul kumaran M mobile no : 8428853991

Realtime Stock Ticker - Detailed Design Document

1. Tech Stack Selection

The selection of technologies for the Realtime Stock Ticker application has been made keeping scalability, performance, and user experience in mind. Frontend: - **React.js** is chosen for its component-based architecture which promotes reusable UI elements, and seamless state management with Redux ensures that changes in data are reflected efficiently. - **Tailwind CSS** is used for styling, offering utility-first classes that accelerate development and maintain consistency. - **TypeScript** adds type safety, improving maintainability and reducing runtime errors. Backend: - **Node.js with Express** provides a lightweight and performant environment for handling multiple concurrent WebSocket connections. - **WebSocket protocol** allows for
low-latency, full-duplex communication, crucial for real-time updates. - **RESTful API** endpoints are designed to handle user preferences, stock additions, and history retrieval in a structured manner. Database: - **MongoDB** is selected for its flexible schema-less design, allowing fast iteration and easy storage of varied user settings and historical data. Cloud and Deployment: -
Docker ensures consistent environments across development, staging, and production. -
Kubernetes provides orchestration, scalability, and load balancing. -
AWS/GCP provides infrastructure services with auto-scaling, monitoring, and security compliance. Monitoring & Security: - Application logs are sent to centralized logging services. - JWT authentication is used for session management. - HTTPS and encryption protocols ensure secure data exchange.

2. UI Structure / API Schema Design

The UI is designed with simplicity and efficiency, ensuring users can track stocks in real-time without clutter. UI Structure: - **Dashboard View**: Displays all tracked stocks with key metrics such as current price, percentage change, and volume. - **Search Functionality**: Users can search and add new stock symbols via an intuitive search bar with auto-complete suggestions. -
Notification Panel: Allows users to set alerts based on price thresholds or percentage movements. - **User Profile**: Contains settings, preferences, theme options, and account management. API Schema Design: - GET /stocks: Retrieve all tracked stocks. - POST /stocks: Add new stock symbol to track. - PUT /stocks/:id: Update stock details. - DELETE /stocks/:id: Remove a stock from tracking. - GET /user/preferences: Retrieve user-specific settings. - POST /user/preferences: Update preferences like theme, notifications, etc. - WebSocket /stream: Push real-time updates including price changes, market news, and alerts. - Authentication Endpoints: - POST /auth/login - POST /auth/register - POST /auth/refresh Error Handling: - Return structured error messages with proper HTTP codes. - Implement rate-limiting to prevent abuse. - Provide fallback data in case of API failure.

3. Data Handling Approach

Efficient and reliable data handling is crucial for a real-time stock ticker. The approach is structured into multiple layers: Data Ingestion: - WebSocket connections from clients are authenticated before subscribing to stock streams. - Backend maintains subscription lists to avoid redundant connections. Data Normalization: - Incoming data from multiple sources is normalized to a common format. - Timestamps, price fields, and volume metrics are standardized. Caching Strategy: - Recent price updates are cached in memory for quick access. - Frequently accessed stocks are stored in Redis for faster retrieval. Data Persistence: - Historical stock prices are periodically saved to MongoDB. - User preferences and alerts are stored with appropriate indexes for optimized queries. Fault Tolerance: - Automatic reconnection strategies in WebSocket clients. - Circuit breaker patterns in API services to handle failures gracefully. Security and Compliance: - Input validation ensures data integrity. - API keys are rotated and securely stored. - GDPR-compliant data management protocols are applied where necessary. Logging and Monitoring: - Real-time logs are streamed to monitoring dashboards. - Alerts are triggered for unusual traffic or data patterns.

4. Component / Module Diagram

The architecture is composed of interconnected modules, ensuring separation of concerns: Frontend Module: - React Components: StockList, StockItem, SettingsPanel, SearchBar, Notifications - Redux Store: Centralized state management - API Service: Handles requests and maintains WebSocket connections - Authentication: Manages login sessions and token refreshes Backend Module: - Express Server: Handles REST endpoints and WebSocket requests - Authentication Service: Manages user tokens and permissions - Data Aggregator: Normalizes and enriches data from providers - Subscription Manager: Manages stock subscriptions per user Database Module: - MongoDB Collections: - users - stocks - preferences - historical_prices - Indexes applied on timestamp and stock symbol for fast retrieval External Integrations: - Financial API Providers: Alpha Vantage, IEX Cloud - Notification Services: Email or push notifications

5. Basic Flow Diagram

The flow of data and interactions in the Realtime Stock Ticker is as follows:

Step 1: User Access - Users access the dashboard via a browser or mobile app.

Step 2: Authentication - Credentials are validated via JWT. - OAuth integration provides third-party login options.

Step 3: Initialization - The user's preferences are loaded from the database. - Stock symbols are retrieved and cached.

Step 4: WebSocket Connection - A connection is established with the backend. - Subscriptions for selected stocks are activated.

Step 5: Data Streaming - Real-time stock prices, news, and alerts are pushed to clients. - Clients update the UI dynamically without page reloads.

Step 6: User Interaction- Users add/remove stocks, set alerts, and customize views. - Changes are sent to the server and persisted.

Step 7: Error Handling - Connection drops are automatically detected. - Reconnection strategies attempt to restore data streams. - API rate limits are monitored, and fallbacks are provided.

Step 8: Historical Analysis - Periodic snapshots of stock prices are stored. - Users can access historical trends and charts.

Step 9: Logging and Monitoring - Server logs record events for debugging. - Alerts notify admins of unusual behavior. This design ensures scalability, reliability, and user satisfaction while maintaining data integrity and security.