HD Project

Financial Platform

TRAN DUC ANH DANG

103995439

1. Introduction

This is a HD project on a Vue based platform designed to manage and visualize stock data. The platform fetches real time quotes, daily time series, and top gainers/losers from an external API (AlphaVantage). By storing and monitoring a portfolio of stocks, users can track market performance and make informed financial decisions.

- Primary Goal: Provide an intuitive user interface for browsing market data, viewing detailed stock information, and maintaining a personalized portfolio.
- Scope: Covers searching for stocks, displaying charts/tables, and storing selections in local storage.

2. Project Overview

This application serves two main purposes:

1. Dashboard: Displays real time market movers, including top gainers and losers, as well as active trading sessions.
2. Portfolio: Offers the capacity to add or remove stock symbols, persist them locally, and show performance tracking.

Key Features

- Search Functionality: Typeahead based symbol search that fetches relevant results.
- Market Data Visualization: Chart.js line or bar charts for daily price fluctuations.
- Local Storage: Saves user's selections, removing the need for manual re entry next session.

3. Requirements

Hardware and Software

- Operating System: Windows (compatible with Linux as well, but tested on Windows and Mac OS).
- Node.js: v14 or above.
- npm: v6 or above.

Libraries and Frameworks

- Vue 3: UI framework (core of the project).
- Vue Router: Client side routing and navigation.
- Vuex: State management for data integrity.
- Bootstrap (or other CSS Framework): For styling prebuilt components.
- Chart.js: For visualizing stock trends.

API Keys

- Alpha Vantage API Key: Required for real time quotes and time series data.
  API key can be obtained on the Alpha Vantage website.

4. Design Architecture

Frontend

1. Components: Presentational units (tables, charts, symbol search bar).
2. Views: Container pages grouping components for the dashboard or portfolio screen.
3. Vuex Store:
    a. Dashboard Module: Manages daily top movers.
    b. Portfolio Module: Manages user-saved stocks.
4. Services: Abstract data fetching and local persistence.
5. Router: Handles navigation between Dashboard, Portfolio, and other future views.

Data Flow

1. User Action -> Vue Router -> View -> Vuex Dispatch -> Service (API call) -> Vuex Commit -> Component Renders.
2. Local Storage is used to store and retrieve user's selected portfolios.

5. Technical Aspects

Data Retrieval

- ApiService: Common helper for making HTTP GET requests using fetch or axios.
- StockService: Specialized wrapper to retrieve stock quotes, daily time series, or top performers.
- PortfolioService: Manages reading and writing user portfolio data from localStorage.

State Management

Dashboard Module

- state: Holds arrays of top gainers/losers and main indexes.
- actions: Fetch top stocks on application load.
- mutations: Update the store with new stock data.

Portfolio Module

- state: Maintains symbols the user has added.
- actions: Add or remove symbols from local storage.
- mutations: Update portfolio state.

Performance

- Throttled symbol search to reduce excessive API calls.
- Light caching for recently fetched data in the store to reduce repeated requests.

6. Innovative Features

- Interactive Sorting: Clicking on table headers in the Dashboard or Portfolio view sorts stocks by price change, volume, or symbol.
- Session Persistence: Using Local Storage to keep the portfolio consistent across page refreshes.

7. Technology Used

- Programming Language: JavaScript (ES6+).
- Vue Framework (v3): Progressive framework for building user interfaces.
- Vuex (v4): Centralized state management.
- Vue Router (v4): Routing for single-page applications.
- Chart.js: For interactive charts and data visualization.
- Bootstrap: UI layout and basic styling.

8. Project File Structure

```
<financial-platform>
├── public/
│   └── index.html               # Root HTML template
├── src/
│   ├── components/
│   │   ├── Navbar.vue            # Top navigation bar
│   │   ├── StockTable.vue        # Reusable table for displaying stock info
│   │   └── ChartContainer.vue    # Chart.js data visualization
│   ├── views/
│   │   ├── Dashboard.vue         # Main dashboard with top gainers, losers
│   │   └── Portfolio.vue         # User portfolio displaying saved stocks
│   ├── services/
│   │   ├── ApiService.js         # Generic HTTP requests
│   │   ├── StockService.js       # Fetch and parse data for stocks
│   │   └── PortfolioService.js   # Manage local storage for user’s portfolio
│   ├── store/
│   │   ├── index.js              # Vuex store setup
│   │   ├── dashboard.js          # Dashboard Vuex module
│   │   └── portfolio.js          # Portfolio Vuex module
│   ├── models/
│   │   └── Stock.js              # Class structuring individual stock data
│   ├── scripts/
│   │   ├── chartUtils.js         # Shared chart configuration utilities
│   │   └── sortingUtils.js       # Helpers for table sorting
│   ├── App.vue                   # Root component
│   └── main.js                   # Application entry point
├── .env                         # Environment variables (API keys)
├── package.json                 # Project dependencies and scripts
└── README.md                    # Basic setup instructions
```

9. Data Storage and Structures

Local Storage Structure

Key: userPortfolio
Value: Serialized JSON array of stock symbols or objects

```
[
    {
        "symbol": "AAPL",
        "displayName": "Apple Inc.",
        "shares": 10
    },
    {
        "symbol": "TSLA",
        "displayName": "Tesla Inc.",
        "shares": 5
    }
]
```

API Data Structure

Stock Quotes

```
{
    "symbol": "AAPL",
    "price": 135.64,
    "change": 1.52,
    "volume": 90873456,
    "timestamp": "2023-10-01T14:30:00Z"
}
```

Daily Time Series

```
[
    {
        "date": "2023-09-30",
        "open": 134.00,
        "high": 136.75,
        "low": 133.50,
        "close": 135.64,
        "volume": 90873456
    },
    {
        "date": "2023-09-29",
        "open": 130.10,
        "high": 134.20,
        "low": 129.50,
        "close": 133.95,
        "volume": 87509381
    }
]
```

Vuex Store Structure

dashboard.js (Module)

```
export default {
    namespaced: true,
    state: () => ({
        gainers: [],
        losers: [],
        activeStocks: []
    }),
    mutations: {
        SET_GAINERS(state, payload) { state.gainers = payload; },
    },
    actions: {
        fetchTopGainers({ commit }) {
        },
    }
}
```

portfolio.js (Module)

```
export default {
    namespaced: true,
    state: () => ({
        symbols: []
    }),
    mutations: {
        SET_SYMBOLS(state, payload) { state.symbols = payload; },
    },
    actions: {
        addSymbol({ commit, state }, symbol) {
        },
    }
}
```

10. Discussion

Key Concepts

- Usability: Ensuring the interface is easy to use and navigate.
- Accessibility: Making the platform accessible to users with disabilities.
- Responsiveness: Designing the interface to adapt to different screen sizes and devices.
- Data Visualization: Presenting complex data in a clear and understandable format using charts and tables.
- State Management: Using Vuex to manage the application's state and ensure data consistency.

Deep Understanding

- Comparison of UI Frameworks: Vue.js was chosen over React or Angular due to its simplicity, ease of integration, and excellent documentation. Vue's component-based architecture made it easier to manage the UI and ensure code reusability.
- Best Practices:
    - Component Based Architecture: Breaking down the UI into reusable components to improve maintainability and scalability.
    - Single Source of Truth: Using Vuex to manage the application's state and ensure data consistency across components.
    - Asynchronous Data Fetching: Using async/await to handle asynchronous API calls and prevent blocking the UI.
- Applications: The platform can be extended to include more advanced features such as real-time stock alerts, portfolio analysis, and integration with brokerage APIs.

Challenges and Solutions

1. Challenge: Handling asynchronous API calls and updating the UI efficiently.
    - Solution: Used async/await to simplify asynchronous code and Vuex to manage the application's state.
2. Challenge: Ensuring the interface is responsive and adapts to different screen sizes.
    - Solution: Used Bootstrap's grid system and CSS media queries to create a responsive layout.
3. Challenge: Implementing the search functionality with typeahead suggestions.
    - Solution: Used a combination of API calls and local caching to provide fast and accurate search suggestions.