

FinBoard

FRONT-END ASSIGNMENT ROUND

Problem Statement

Create a Customizable Finance Dashboard where users can build their own real-time finance monitoring dashboard by connecting to various financial APIs and displaying real-time data through customizable widgets.

Objectives

1. Develop a user-friendly finance dashboard builder that supports real-time data visualization.
2. Enable seamless integration with multiple financial APIs (stocks).
3. Provide intuitive widget management with drag-and-drop functionality.
4. Implement robust state management and data persistence capabilities.

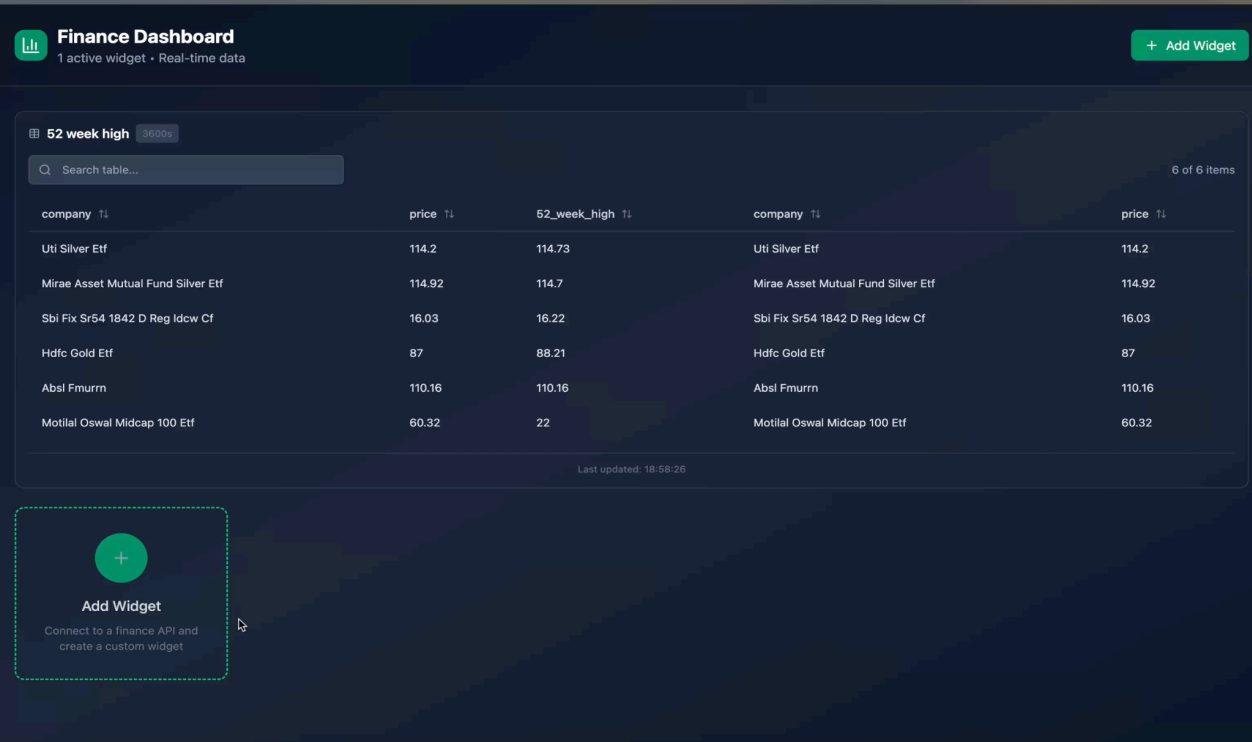
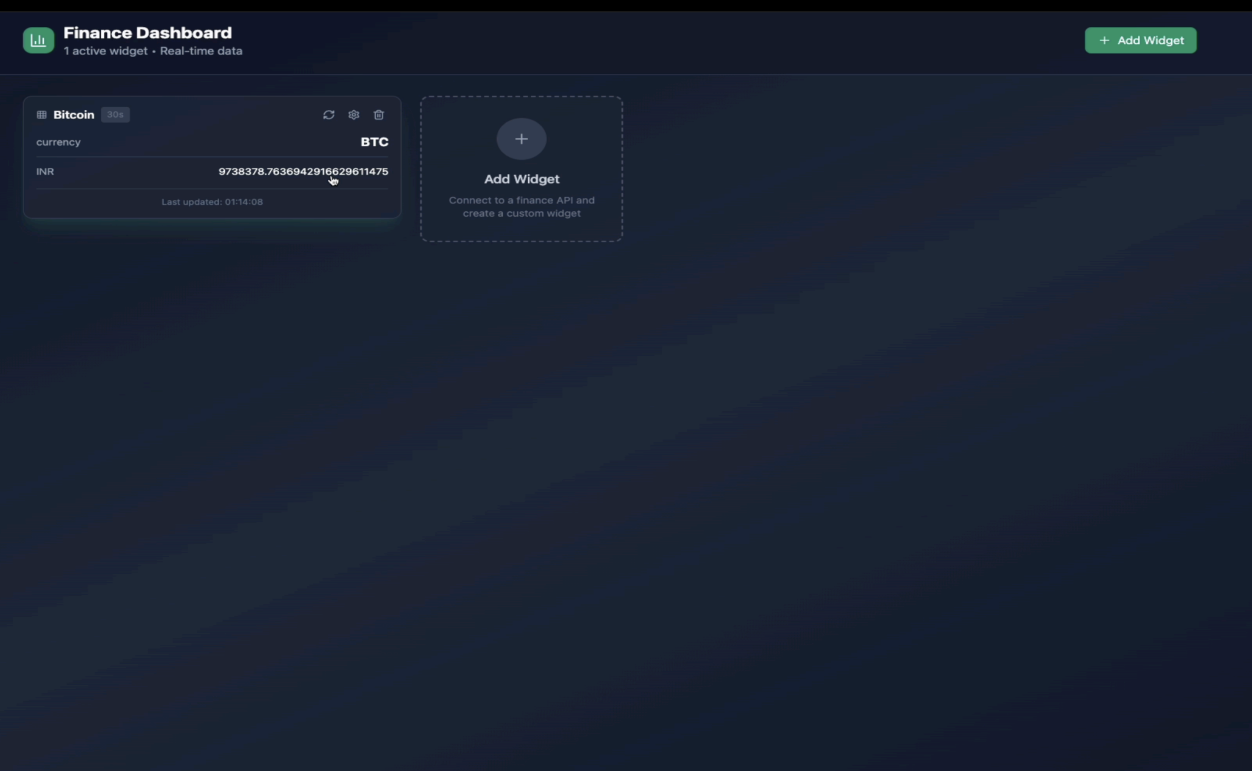
Technologies

- **Frontend Framework:**Next.js
- **Styling:** CSS, Tailwind CSS, Styled-Components
- **State Management:** Redux/Redux Toolkit, Zustand, Jotai
- **Data Visualization:** Chart.js, Recharts, or similar charting libraries
- **Deployment:** Vercel, Netlify, or AWS

Features

1. Widget Management System:

- **Add Widgets:** Users can create new finance data widgets by connecting to any financial API-
 - **Table:** A paginated list or grid of stocks with filters and search functionality.
 - **Finance Cards:** A list or single card view for:
 - Watchlist
 - Market Gainers
 - Performance Data
 - Financial Data
 - **Charts:** Candle or Line graphs showing stock prices over different intervals (*Daily, Weekly, Monthly*)
- **Remove Widgets:** Easy deletion of unwanted widgets from the dashboard
- **Rearrange Layout:** Drag-and-drop functionality to reorganize widget positions on the dashboard
- **Widget Configuration:** Each widget includes a configuration panel for customization



2. API Integration & Data Handling:

- **Dynamic Data Mapping:** Users can explore API responses and select specific fields to display
- **Real-time Updates:** Automatic data refresh with configurable intervals
- **Data Caching:** Intelligent caching system to optimize API calls and reduce redundant requests

3. User Interface & Experience:

- **Customizable Widgets:** Each widget displays as a finance card with editable titles and selected metrics
- **Responsive Design:** Fully responsive layout supporting multiple screen sizes
- **Loading & Error States:** Comprehensive handling of loading, error, and empty data states

4. Data Persistence:

- **Browser Storage Integration:** All widget configurations and dashboard layouts persist across sessions
- **State Recovery:** Complete dashboard restoration upon page refresh or browser restart
- **Configuration Backup:** Export/import functionality for dashboard configurations

5. Advanced Widget Features:

- **Field Selection Interface:** Interactive JSON explorer for choosing display fields
- **Custom Formatting:** Support for different data formats (currency, percentage, etc.)
- **Widget Naming:** User-defined widget titles and descriptions
- **API Endpoint Management:** Easy switching between different API endpoints per widget

API Integration Guidelines

- Use reliable financial APIs ([Alpha Vantage](#), [Finnhub](#), [IndianAPI](#) etc.)
- You'll need an API key to access the endpoints. Generate an API key for the app and check the limits on requests per minute and requests per day.

- Implement proper API key management and security practices
- Handle rate limiting and API quota management gracefully

Technical Requirements

- Follow a well-defined, scalable folder structure
- Write clean, maintainable, and well-documented code
- Optimize for performance with lazy loading, code splitting, rendering strategies (CSR/SSR)

Brownie Points

Note: Focus on core functionality first. Once basic features are working, explore these advanced features:

1. **Dynamic Theme Switching:** Implement a dynamic theme-switching feature that allows users to toggle between Light and Dark modes seamlessly.
2. **Real-time data:** Implement sockets for live data updates over widget.
3. **Dashboard Templates:** Pre-built dashboard templates for

Evaluation Criteria

This assignment evaluates:

- **Frontend Development Skills:** React proficiency, component architecture, and modern JavaScript
 - **State Management:** Effective use of Redux/Zustand for complex application state
 - **API Integration:** Handling dynamic JSON data and implementing robust data fetching strategies
 - **User Experience Design:** Creating intuitive interfaces that non-technical users can navigate easily
 - **Problem-Solving Ability:** Implementing flexible solutions for diverse API structures and data formats
 - **Code Quality:** Clean, maintainable, and scalable code architecture
-

FAQ

API and Data Handling

Q: How do I handle different API response formats?

A: Implement a flexible data mapper that can handle various JSON structures. Consider creating adapters for common API formats.

Q: What happens if the API request limit is reached?

A: If the API request limit is reached, the website may temporarily stop fetching new data until the limit resets. You may see a message indicating that the API limit has been reached and suggesting you try again later.

Q: How do I create a new API key?

A: To create a new API key, follow these steps :

1. Visit the API provider's website (e.g., [Alpha Vantage](#)).
2. Sign in to your account or create a new account if you don't have one.
3. Navigate to the API section or dashboard.
4. Look for an option to generate a new API key.
5. Follow the prompts to create and copy the new API key.
6. Update your web application's configuration with the new API key.

Q: What are common API errors and what should I do when API rate limits are exceeded?

A: Common API errors include exceeding the rate limit, invalid API keys, CORS and network issues. To avoid rate limit cache API responses. Also, each API provider has specific rate limits, which are typically documented on their website.

Q: What should I do when API rate limits are exceeded?

A: For support or feedback, you can contact the team through the provided contact information in the email.

All the best !!