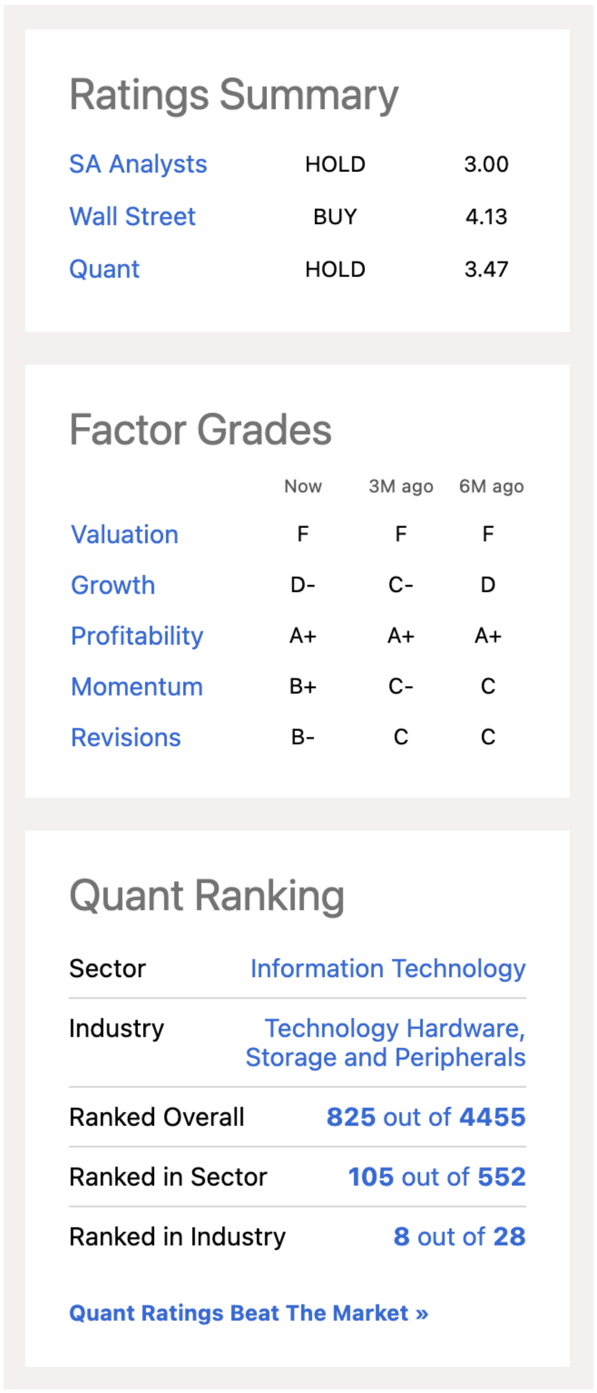
Front-End Developer Assignment



Ratings Summary

SA Analysts	HOLD	3.00
Wall Street	BUY	4.13
Quant	HOLD	3.47

Factor Grades

	Now	3M ago	6M ago
Valuation	F	F	F
Growth	D-	C-	D
Profitability	A+	A+	A+
Momentum	B+	C-	C
Revisions	B-	C	C

Quant Ranking

Sector	Information Technology
Industry	Technology Hardware, Storage and Peripherals
Ranked Overall	**825** out of **4455**
Ranked in Sector	**105** out of **552**
Ranked in Industry	**8** out of **28**

Quant Ratings Beat The Market »

Objective

The goal of this assignment is to assess your ability to implement a financial card component located in the right rail similar to the provided screenshot. The component should be highly performant, user-friendly, and maintainable. Additionally, you should demonstrate good practices in handling data from an API, ensuring a smooth user experience, and writing quality code.

Mock API

To complete this assignment, you will need to fetch data from a mock API. This simulated API will provide the necessary data for rendering the card component.

Base URL: seekingalpha.free.beeceptor.com

User: GET /user
Rating Summary: GET /ratings-summary
Factor Grades Now: GET /factor-grades/now
Factor Grades 3M ago: GET /factor-grades/3m
Factor Grades 6M ago: GET /factor-grades/6m
Quant Ranking: GET /quant-ranking

Expected Result

Create a repository for your project on GitHub or any other repository hosting service of your choice. Upload all your code, including the implementation of the card component, unit tests, and any additional documentation. Include a README file with clear instructions on how to set up and run the project.

Requirements

- Component Design

Implement a card component that resembles the provided screenshot. Ensure the component is responsive and looks good on various screen sizes. Display a loader or skeleton screen while data is being fetched. The order of cards must be respected.

- Premium Users

Consider that some users might not be eligible to see specific cards. Ensure the component gracefully handles the absence of data or permissions. A non-premium user sees only the "Quant Ranking" card. The rest of the cards must be hidden for non-premium ones.

- Data Fetching

Load the content from a simulated API. Render the content as soon as it is available to avoid layout shifts or blinking.

- User Experience

Implement loaders to indicate data fetching. Avoid layout shifts and blinking to ensure a smooth experience. Ensure the component is performant and does not introduce significant delays.

- Code Quality

Cover the code with unit tests and demonstrate a good understanding of testing best practices. Use type checking to ensure type safety, preferably with TypeScript.

- Code Structure

Ensure the code is modular and the build is split into chunks for better maintainability and performance. Follow best practices for code organization and readability.

Criteria and Metrics Not Taken into Account

- Stylistic Preferences

Personal stylistic choices such as specific color schemes, font styles, or design aesthetics not explicitly required by the assignment will not be considered. Pixel perfect is not required.

- Code Formatting Preferences

Specific code formatting styles (e.g., single vs. double quotes, indentation style) will not be evaluated. Provide readable and consistent code.

- Browser-Specific Implementations

Implementation specifics for supporting older or non-mainstream browsers will not be taken into account. Focus on modern, commonly-used browsers (e.g., latest versions of Chrome, Firefox, Safari, and Edge).

- Advanced Performance Optimization

Extremely advanced performance optimizations, such as fine-tuning Webpack configurations or in-depth browser rendering optimizations, will not be required. Basic performance best practices should be followed.

- Server-Side Rendering (SSR)

Implementation of server-side rendering will not be evaluated. The focus is on client-side rendering and user experience.