Development Scenario – Product Warehouse

Technologies

Backend: NodeJS with TypeScript/JavaScript

Frontend: ReactJS

Database: PostgreSQL

API: GraphQL (Apollo Server) & REST API

Abstract

A customer has a number of warehouses. They wish to track the movement of stock in and out of each warehouse. A warehouse has a particular size, this represents the maximum stock amount allowed in this warehouse. Each warehouse is stocked with products. Products have a size per unit amount. Products can be imported or exported from a warehouse at which point the product, amount and date are recorded. The customer has hazardous and non-hazardous products. It is very important that hazardous products are not kept in the same warehouse as non-hazardous products. Imports can be in the future or the past. The customer will want to see what the current stock level is in a warehouse and what stock space is remaining.

Requirement

Build a small stock management application consisting of two pages. The frontend must be written in ReactJS, consuming a GraphQL API using Apollo Server. The GraphQL API should communicate with a separate REST API solely for calculation operations. You have the option to either use an existing open-source REST API or create one yourself, depending on your preference. The application data should be stored in a PostgreSQL database. The entire application should be developed using either TypeScript or JavaScript, based on your choice.

Screen 1: Product Entry Screen

Allows the user to quickly add products to the master products list and see a full list of the product list.

Screen 2: Warehouse Stock Movement Screen

A user can switch between warehouses within the page. The page must show the current stock amount, free stock space remaining. The user can see a historic list of imports and exports as well as add a new import or export.

Note

This is an open scenario designed to test your knowledge and use of both front end and back end technologies. The requirements are simplistic in order for you to use some imagination. Assume that you already have a data structure containing your list of warehouses.

Bonus Feature ideas (optional):

- Add unit tests for critical components
- Containerize with Docker and provide compose setup
- Set up GitHub Actions or similar CI/CD workflow
- Include API documentation

Evaluation Criteria:

- Code Quality and Architecture
    - Clear separation of concerns and modular design
    - Strong TypeScript typing with minimal 'any' usage
    - Consistent coding style and formatting
    - Effective use of design patterns and best practices
- Bonus points for additional features or optimizations

Submission Guidelines:

- Provide a GitHub repository link with your solution
- Include a README with:
    - Setup instructions
    - Any assumptions made
    - List of completed features and known issues
    - Brief explanation of your technical choices

What We're Looking For:

Technical Excellence:

- Clean, maintainable code with proper TypeScript usage
- Efficient React component composition and state management
- Well-organized GraphQL schema with proper resolvers
- Smart database design and query patterns

Professional Quality:

- Use of semantic commit messages and demonstration of proficiency with Git
- Robust error handling and logging
- Clear technical communication and documentation

Remember, the focus is not on completing every feature, but rather on showcasing your approach, code quality, and decision-making skills within the given time constraint.