Coding Challenge: React & ASP.NET Core API Integration

Title: "Product Inventory Management System"

📝 Problem Statement:

You are building a **Product Inventory Management System** where users can:

1. **View a list of products**
2. **Add new products**
3. **Update product details**
4. **Delete products**

Your task is to **create a full-stack solution** using **React (Frontend)** and **ASP.NET Core Web API (Backend)**, along with SQL Server for data storage.

📌 User Stories & Tasks:

1️⃣ User Story: View Product List

As a user, I want to see a list of available products so that I can manage my inventory.

✅ Tasks:

- Fetch product data from the **ASP.NET Core API** using **Axios** in React.
- Display products in a table with columns: `Product ID`, `Name`, `Price`, and `Actions`.

🎯 Expected Outcome:

✅ When the page loads, users see a list of products fetched from the API.

⏳ Difficulty Level:

⭐ Beginner

🔍 Constraints & Edge Cases:

- Handle **loading state** while fetching data.
- Show an **error message** if the API call fails.

2️⃣ User Story: Add a New Product

As a user, I want to add new products to my inventory so that I can expand my stock.

✅ Tasks:

- Create a **form** with fields: `Product Name`, `Price`.
- When users click "Add Product", **send a POST request** to the API.
- Show a **success message** after adding.

🎯 Expected Outcome:

✅ Users can add a new product, and it appears in the product list.

⏳ Difficulty Level:

⭐⭐ Intermediate

🔍 Constraints & Edge Cases:

- Prevent **submitting empty fields**.
- Show an **error message** if the product already exists.

3️⃣ User Story: Update Product Details

As a user, I want to modify product details so that I can correct mistakes or update prices.

✅ Tasks:

- Add an **"Edit" button** next to each product.
- Show a **pre-filled form** with the product details.
- On clicking "Update", send a **PUT request** to update the database.

🎯 Expected Outcome:

✅ Users can edit product details, and the changes reflect instantly.

⏳ Difficulty Level:

⭐⭐⭐ Intermediate

🔍 Constraints & Edge Cases:

- Prevent **saving empty values**.
- Ensure **updated data is reflected immediately**.

4️⃣ User Story: Delete a Product

As a user, I want to delete products that are no longer available.

✅ Tasks:

- Add a **"Delete" button** next to each product.
- Show a **confirmation dialog** before deleting.
- Send a **DELETE request** to the API.

🎯 Expected Outcome:

✅ Users can remove a product from the inventory, and it disappears from the UI.

⏳ Difficulty Level:

⭐⭐⭐ Intermediate

🔍 Constraints & Edge Cases:

- Ensure users **confirm before deleting**.
- Show an **error message** if the deletion fails.

📊 Self-Evaluation Rubric:

Criteria	Beginner (1-2 pts)	Intermediate (3-4 pts)	Advanced (5 pts)
API Integration	Can fetch data but no error handling	Uses Axios with error handling	Uses Axios + loading state + retries
UI/UX	Basic UI with form & buttons	Well-structured UI with CSS	Fully responsive UI with animations
State Management	Uses `useState()` only	Uses `useEffect()` for API calls	Implements **useReducer() or Context API**
CRUD Operations	Can fetch products only	Can fetch, add, and update	Can fetch, add, update, delete with validations
Error Handling	No error handling	Basic error handling (alerts)	Full error handling (toasts, validation, API retries)

- 🟢 **18-25 points → Expert** 🚀
- 🟡 **10-17 points → Intermediate** 💡
- 🔴 **0-9 points → Beginner** 🛠️

🛠️ Tech Stack Required:

✅ **Frontend:** React, Axios, Bootstrap/TailwindCSS
✅ **Backend:** ASP.NET Core Web API, Entity Framework Core
✅ **Database:** SQL Server