# React JS Assignment

## 🎯 Objective

Build a small, high-performance React application that displays product data fetched from the Fake Store API. The goal is to assess your ability to work with **APIs, efficient state/data management, UI optimization, and handling live data updates/mutations**.

## 📋 Core Requirements

### 1. User Authentication (Simple Login Page)

* **Functionality:** Implement a simple, static login page. This page should be purely UI/UX, simulating a successful login using local state (e.g., setting an isLoggedIn flag in React state or context) upon successful submission of placeholder credentials (e.g., "user" / "password").
* **Requirement:** The main Product List view **must only be accessible after the user has successfully "logged in."**
* **State Persistence (Optional Bonus):** Use the Browser's Session/Local Storage (or a state management library feature) to remember the logged-in status across page reloads.

### 2. Data Fetching and List Display

* **API Endpoint:** Fetch product data from https://fakestoreapi.com/products.
* **List View:** Display a grid or list of all products. Each product card must clearly show:
  + **Product Image**
  + **Title**
  + **Price** (formatted as currency)
  + **Category**

### 3. Product Detail View

* **API Endpoint:** Fetch product data from https://fakestoreapi.com/products/id.
* Implement a mechanism (e.g., a modal overlay, a slide-out panel, or a separate route/view) to show detailed information when a product card is clicked.
* The detailed view must include:
  + The complete **Product Description**.
  + The **Rating** (score and count).

### 4. Product Update Task

* **Functionality:** Implement an "Edit" feature accessible from the Product Detail View that allows the user to modify and submit changes to a product's details.
* **API Endpoint:** Use the PUT (or PATCH) method on the Fake Store API: https://fakestoreapi.com/products/{id}
* **Request Body (Required fields for update):** The client should send the following structure in the request body, but for a simple test, focus on allowing the user to modify the title and price.  
  {  
   "title": "string",  
   "price": 0.1,  
   "description": "string",  
   "category": "string"  
  }
* **State Management:** After a successful update, the application's cached list of products **must automatically reflect the change** without requiring a manual full list re-fetch. This tests efficient cache invalidation or update logic.

### 5. Product Deletion Task (Mutation)

* **Functionality:** Implement a clear **"Delete Product"** button accessible from the Product Detail View.
* **API Endpoint:** Use the DELETE method on the Fake Store API: https://fakestoreapi.com/products/{id}.
* **UI/UX:** Require a confirmation step (e.g., a modal confirmation) before sending the DELETE request.
* **State Management:** After a successful deletion, the product **must be immediately removed** from the cached list, and the UI should reflect the change instantly.

### 6. UI / UX and Usability

* **Clean Design:** The application must have a clean, user-friendly, and responsive design (MUI Component or Tailwind CSS or standard CSS is acceptable).
* **State Handling:** Implement graceful handling for all network states:
  + **Loading:** Display a clear loading indicator (e.g., a spinner or skeleton loader) while data is being fetched or updated.
  + **Error:** Display a helpful, user-friendly message if the API request or update fails.
* **Bonus (Simple Feature):** Implement a simple **filter by category** or a basic **search input** to narrow down the displayed products.

## ⚙️ Technical Requirements & Optimization

### 7. State Management & Caching

* **MANDATORY:** You must use an external library for data fetching and state management. Choose **one** of the following or a similar alternative:
  + **React Query (TanStack Query)**
  + **Redux Toolkit** (with RTK Query)
* **Caching:** The chosen library must be configured to intelligently cache API responses to prevent redundant network requests when components remount or the user navigates.

### 8. Data Refresh on Window Focus (Revalidation)

* The product data **must automatically refresh** (revalidate) whenever the browser window or tab regains focus (i.e., when the user switches back to the application's tab). This feature ensures the user always sees the most up-to-date data.
  + *(Hint: If using React Query or RTK Query, this feature is often built-in but requires understanding and potential configuration.)*

### 9. Optimization

* **Performance:** Structure components and manage state to **avoid unnecessary re-renders**.
* **Bonus (Scale Handling):** Implement basic **pagination** (e.g., showing 10 products per page) or an **infinite scroll** mechanism to efficiently load and display data, even if the API provided thousands of items.

## 📝 Deliverables & Technical Guidelines

### Technical Guidelines

* **Code Quality:** Structure your code cleanly using functional components, hooks, and reusable components.

### Required Deliverables

1. **Source Code:** The complete, runnable project code(Github repo link).
2. **Documentation (README.md):** A minimal but effective documentation file.
3. **Live Link:** The deployed application on a free hosting service (e.g., Vercel, Netlify, GitHub Pages).

Good luck!