REACT-HTTP REQUEST

IIHT

Time To Complete: 10 to 12 hr

CONTENTS

1 Project Abstract

Modern web applications frequently rely on data fetched from remote APIs. This project introduces HTTP **requests in React** using Axios and React's `useEffect` hook. You will build a simple product inventory viewer that fetches product data from a live API, displays a loading state while fetching, and gracefully handles potential errors. By implementing a custom hook (`useFetchProducts`), this project promotes clean code architecture and encourages separation of concerns between logic and UI rendering.

2 Problem Statement

You are required to build a **Product Inventory** application in React that fetches and displays product data from an external API (`https://fakestoreapi.com/products`). The application should:

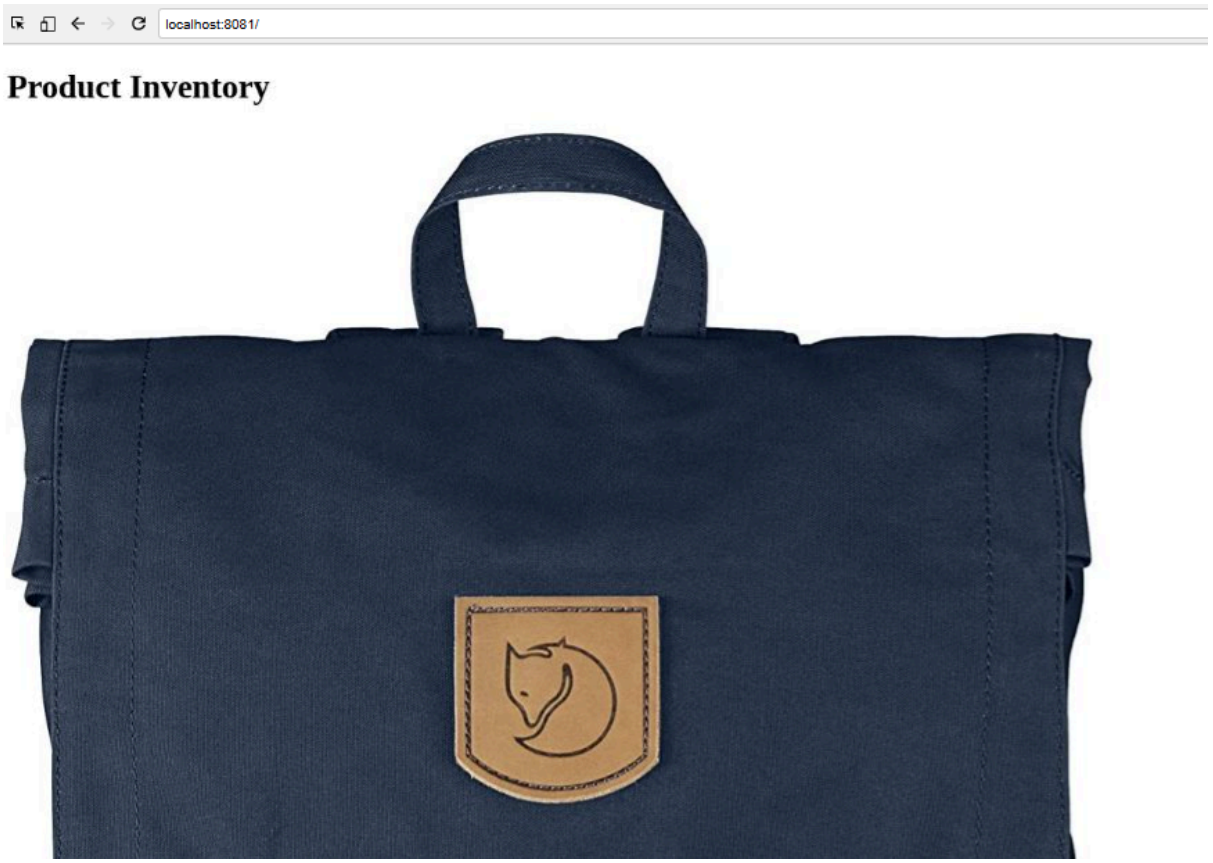
- Show a loading indicator while data is being fetched.
- Display an error message if the fetch fails.
- Render a list of products in a clean layout using reusable `ProductCard` components.

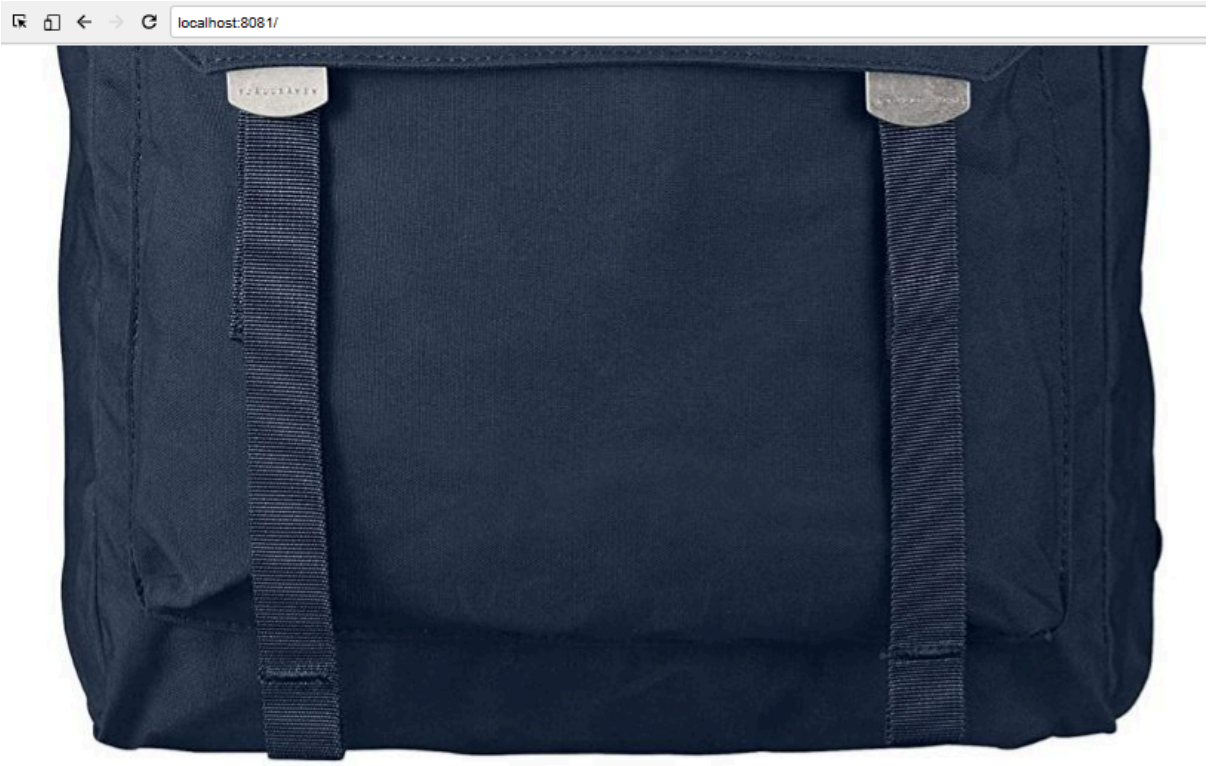
The data fetching logic should be encapsulated in a custom hook named `useFetchProducts`, promoting modularity and reusability. This project helps reinforce the fundamentals of making asynchronous HTTP requests, managing loading and error states, and dynamically rendering data in a React application.

3 Proposed HTTP Request Application Wireframe

UI needs improvisation and modification as per given use case and to make test cases passed.

3.1 ScreenShots





Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops

$109.95

localhost:8081/

John Hardy Women's Legends Naga Gold & Silver Dragon Station Chain Bracelet

$695

Solid Gold Petite Micropave

$168

4 BUSINESS-REQUIREMENT:

As an application developer, develop the HTTP Request (Single Page App) with below guidelines:

User Story #	User Story Name	User Story
US_01	Welcome Page	As a user I should be able to visit the welcome page as the default page. Acceptance criteria: 1. Use a **custom hook** to fetch product data from an external API (`https://fakestoreapi.com/products`). 2. Show a **loading spinner** while the data is being fetched. 3. If the request fails, display an error message. 4. Once the data is fetched, render the product list using `ProductList` and `ProductCard` components. **State & Hook Overview:** **useFetchProducts (Custom Hook):** **State Variables:** 1. `products`: array – stores the fetched product data. 2. `loading`: boolean – tracks loading state. 3. `error`: object | null – holds error details (if any). **Behavior:** 4. Makes an HTTP GET request to the API on component mount using `axios`. 5. Updates `products`, `loading`, and `error` accordingly. 6. Returns these values to the calling component. **Component Breakdown:** **App Component** 1. Calls `useFetchProducts()` to fetch data. 2. Based on returned values: ● Shows `<LoadingSpinner />` while loading. ● Displays error if present. ● Renders: ➔ `<h1>` with heading

		➔ `<ProductList />` with `products` passed as props **LoadingSpinner Component** 3. Displays a message: **"Loading..."** **ProductList Component** 4. Accepts `products` array as a prop. 5. Maps over each product and renders a `ProductCard` for each. **ProductCard Component** 6. Displays: ● Product image ● Product title ● Product price 7. Uses **prop validation** (`PropTypes`) to enforce type safety. **HTML Structure:** **1. App Component:** ● Use a top-level `<div>` to contain the layout. ● Add: ➔ `<h1>` with the heading: **"Product Inventory"** ➔ Display of: ➢ Loading spinner (`LoadingSpinner`) if data is being fetched ➢ Error message if request fails ➢ Product list once loaded **2. LoadingSpinner Component:** ● Use a `<div>` that contains: ➔ Text: **"Loading..."** **3. ProductList Component:** ● Use a `<div>` to wrap the list of products. ● Inside, loop through each product and render a `ProductCard`. **4. ProductCard Component:** ● Use a `<div>` to represent an individual product card. ● Inside that: ➔ `<img>` tag to show the product image ➔ `<h3>` tag to show the product title

		➔ `<p>` tag to show the product price (e.g., `$25.99`) **Dynamic Behavior:** 1. When the app loads: • The custom hook triggers a GET request to the API. • While waiting, a **Loading...** message is shown. • On success, products are displayed. • On error, a message like **"Error fetching products: [message]"** is displayed. 2. UI updates in real-time by flipping the value of `completed`. 3. State is updated immutably using `.map()` and `setTasks`. **** Kindly refer to the screenshots for any clarifications. ****

5 CONSTRAINTS

1. You should be able to press the “TAB” key and “SHIFT + TAB” to navigate from top field to bottom field and vice-versa.

6 MANDATORY ASSESSMENT GUIDELINES

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.**
3. **This editor Auto Saves the code.**
4. **If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.**
5. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**

6. **This is a web-based application, to run the application on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**

   Note: The application will not run in the local browser

7. **You can follow series of command to setup React environment once you are in your project-name folder:**
   a. **npm install -> Will install all dependencies -> takes 10 to 15 min.**
   b. **npm run start -> To compile and deploy the project in browser. You can press the <Ctrl> key while clicking on localhost:4200 to open the project in the browser -> takes 2 to 3 min.**
   c. **npm run test -> to run all test cases. It is mandatory to run this command before submission of workspace -> takes 5 to 6 min.**
8. **Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on "Submit Assessment" after you are done with code.**
9. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**