Ministry of Higher Education and Scientific Research - University of Mila
Institute of Mathematics and Computer Science - Department of Computer Science
Master 1 STIC – Advanced Web Development - 2025/2026

Practical Works 2 – Giving the App a Heartbeat (Hooks & Fetch)

Objective:

Fetch real product data from an external API and render it dynamically using React Hooks (**useState**, **useEffect**).

1. Environment Verification

Before we start, ensure you are in the correct directory. Open your terminal:

```
# 1. Navigate to your project folder
cd Desktop/Master_Web_2026/project-store

# 2. Open in VS Code (if not already open)
code .

# 3. Start the development server
npm run dev
```

2. The Logic: Understanding the API

Instead of creating our own database (that's for the second half of the semester), we will use a **REST API** called **Fake Store API**. It simulates a real backend.

- **URL:** https://fakestoreapi.com/products
- **Format:** It returns a **JSON** array of objects (ID, Title, Price, Description, Image).

3. Creating a Reusable Component: ProductCard.jsx

In TP1, we had a generic box. Now, we create a specialized component that accepts **props** (data passed from a parent).

Create src/components/ProductCard.jsx:

```
export default function ProductCard({ product }) {
  return (
    <div className="bg-white border p-4 rounded-lg shadow-sm hover:shadow-lg transition-shadow duration-300">
      <div className="h-48 flex items-center justify-center overflow-hidden">
        <img src={product.image} alt={product.title} className="max-h-full object-contain" />
      </div>
      <div className="mt-4">
        <h3 className="text-sm font-semibold truncate" title={product.title}>{product.title}</h3>
        <p className="text-gray-500 text-xs mt-1 capitalize">{product.category}</p>
        <div className="flex justify-between items-center mt-3">
          <span className="text-orange-600 font-bold">${product.price}</span>
          <button className="bg-orange-100 text-orange-600 px-3 py-1 rounded text-xs font-bold hover:bg-orange-600
hover:text-white">
            Add to Cart
          </button>
        </div>
      </div>
    </div>
  );
}
```

4. The "Brain" of the Page: Fetching Data

We will now update **src/pages/Home.jsx**. We need to use:

- **useState**: To store the products once they arrive from the API.
- **useEffect**: To trigger the fetch as soon as the page loads.

Update src/pages/Home.jsx:

```
import { useState, useEffect } from 'react';
import ProductCard from '../components/ProductCard';

export default function Home() {
  const [products, setProducts] = useState([]); // State for products
  const [loading, setLoading] = useState(true); // State for the loading spinner

  useEffect(() => {
    // This function runs when the component mounts
    fetch('https://fakestoreapi.com/products')
      .then((res) => res.json())
      .then((data) => {
        setProducts(data);
        setLoading(false);
      })
      .catch((err) => console.error("Error fetching data:", err));
  }, []);

  if (loading) {
    return (
      <div className="flex justify-center items-center h-screen">
        <div className="animate-spin rounded-full h-12 w-12 border-t-2 border-b-2 border-orange-600"></div>
      </div>
    );
  }

  return (
    <div className="p-8">
      <h2 className="text-3xl font-bold mb-8 text-gray-800">Explore Our Collection</h2>
      <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-8">
        {products.map((item) => (
          <ProductCard key={item.id} product={item} />
        ))}
      </div>
    </div>
  );
}
```

5. Critical Checkpoints for Students

- **The Key Prop:** Notice **key={item.id} in the .map()** function. React needs this to track which item is which. Never skip it!
- **Network Tab:** Open **Chrome DevTools (F12)** -> **Network** tab. Refresh the page to see the fetch request happening in real-time.
- **Props Destructuring:** Look at how we used **{ product }** in the **ProductCard** parameters.

Deliverables for the end of TP2:

1. **Dynamic Data:** Your app must display real products with real prices and images.
2. **Loading State:** A spinner must appear while the data is being fetched (especially useful on slow 4G/ADSL connections!).
3. **Responsive Layout:** The grid must still be responsive (4 columns on desktop, 1 on mobile).