Day 11: React Custom Hooks – A Complete Guide 🚀

Today, you'll **master custom hooks in React**. You'll learn:
✅ What custom hooks are
✅ Why they are useful
✅ How to create and use them
✅ Practical examples with step-by-step explanations

1️⃣ What Are Custom Hooks?

A **Custom Hook** is a JavaScript function that:
- Uses **React hooks** (e.g., `useState`, `useEffect`, etc.).
- **Encapsulates reusable logic** so it can be shared between components.
- **Starts with "use"** (e.g., `useFetch`, `useDarkMode`).
- Makes React components **cleaner** and **more maintainable**.

Think of custom hooks as **"utility functions for components."** Instead of writing the same logic **multiple times**, you **write it once and reuse it**.

2️⃣ Why Use Custom Hooks?

Without custom hooks, you'd **repeat the same logic** in different components.
With custom hooks, you can **write once and reuse** in multiple places.

Benefits of Custom Hooks:

✅ **Avoid code duplication** – Write logic once, use it everywhere.
✅ **Improve code readability** – Keep components small and focused.
✅ **Encapsulate side effects** – Keep API calls, event listeners, and state handling separate.
✅ **Easier to test and maintain** – Debugging becomes simpler.

3️⃣ Creating a Simple Custom Hook – useCounter

Step 1: Define useCounter Hook

We'll create a **reusable counter hook** that manages a count state.

```
import { useState } from "react";

function useCounter(initialValue = 0) {
  const [count, setCount] = useState(initialValue);

  const increment = () => setCount(count + 1);
  const decrement = () => setCount(count - 1);
  const reset = () => setCount(initialValue);

  return { count, increment, decrement, reset };
}

export default useCounter;
```

✅ **Encapsulates state and functions (count, increment, decrement, reset)**
✅ **Reusable in any component**

Step 2: Use useCounter in a Component

Now, let's use the hook inside a React component.

```
import React from "react";
import useCounter from "./useCounter";

function CounterComponent() {
  const { count, increment, decrement, reset } = useCounter(10); // Initial value =
10

  return (
    <div>
      <h2>Count: {count}</h2>
      <button onClick={increment}>Increase</button>
      <button onClick={decrement}>Decrease</button>
      <button onClick={reset}>Reset</button>
    </div>
  );
}

export default CounterComponent;
```

✅ **Uses useCounter without worrying about logic**
✅ **Easier to maintain and reuse**

4️⃣ Creating a `useFetch` Hook (Fetching API Data)

A **reusable custom hook** for making API calls.

Step 1: Define useFetch Hook

```
import { useState, useEffect } from "react";

function useFetch(url) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    setLoading(true);
    fetch(url)
      .then((response) => response.json())
      .then((data) => {
        setData(data);
        setLoading(false);
      })
      .catch((err) => {
        setError(err);
        setLoading(false);
      });
  }, [url]);

  return { data, loading, error };
}

export default useFetch;
```

✅ **Handles fetching, loading, and errors**
✅ **Reusable for any API request**

Step 2: Use useFetch in a Component

```
import React from "react";
import useFetch from "./useFetch";

function UsersList() {
  const { data, loading, error } = useFetch(
    "https://jsonplaceholder.typicode.com/users"
  );

  if (loading) return <p>Loading...</p>;
  if (error) return <p>Error loading data.</p>;

  return (
    <ul>
      {data.map((user) => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}

export default UsersList;
```

✅ **Fetches user data dynamically**
✅ **Reuses `useFetch` for any API request**

5️⃣ More Advanced Custom Hooks

Let's explore **more useful hooks**!

🔹 `useLocalStorage` (Store Data in Local Storage)

This hook stores data **persistently** in `localStorage`.

```
import { useState, useEffect } from "react";

function useLocalStorage(key, initialValue) {
  const [storedValue, setStoredValue] = useState(() => {
    const savedItem = localStorage.getItem(key);
    return savedItem ? JSON.parse(savedItem) : initialValue;
  });

  useEffect(() => {
    localStorage.setItem(key, JSON.stringify(storedValue));
  }, [key, storedValue]);

  return [storedValue, setStoredValue];
}

export default useLocalStorage;
```

Using `useLocalStorage`

```
import React from "react";
import useLocalStorage from "./useLocalStorage";

function ThemeSwitcher() {
  const [theme, setTheme] = useLocalStorage("theme", "light");

  return (
    <div className={theme}>
      <button onClick={() => setTheme(theme === "light" ? "dark" : "light")}>
        Toggle Theme
      </button>
    </div>
  );
}

export default ThemeSwitcher;
```

✅ **Persists theme selection even after page reload**

6️⃣ Summary of Day 11

✔ **Custom Hooks** allow reusable logic
✔ **Encapsulated logic** makes code cleaner
✔ **Examples:** `useCounter`, `useFetch`, `useLocalStorage`
✔ **Custom hooks follow naming convention `useSomething`**
✔ **Hooks improve maintainability & reusability**

7️⃣ Challenge: Build Your Own Custom Hook!

Try creating a **new custom hook** on your own. Here are some ideas:

- **`useToggle`** – Toggles between `true` and `false`
- **`useOnlineStatus`** – Detects if the user is online or offline
- **`useWindowSize`** – Tracks window width and height

🚀 Next Step: Day 12 - React Forms & Form Validation