Module 5 — React + Vite: Component Lifecycle & `useEffect` (80 minutes)

> Audience: Kids 13–14. This module builds **directly** on Modules 1–4. All code stays in the same project. Today's work goes into `src/day5/`.

Learning Objectives

By the end of this module, students will: 1. Understand what a **component lifecycle** means (birth, update, death). 2. Know **when and why** React runs code automatically. 3. Use the `useEffect` **hook** correctly. 4. Understand the **dependency array** (`[]`, `[value]`, no array). 5. Fetch data from an API (fake API example). 6. Clean up effects (timers / listeners). 7. See **complete solutions** for Module 4 homework.

80-Minute Agenda

- **0–10 min** — Review: state & forms (Module 4)
- **10–25 min** — Component lifecycle (kid-friendly explanation)
- **25–45 min** — `useEffect` explained step by step
- **45–60 min** — Hands-on: timer & API fetch
- **60–70 min** — Cleanup in `useEffect`
- **70–80 min** — Homework solution + recap + Q&A

Recap from Module 4 (0–10 min)

Ask students: - What is state? - Why do we use `useState` instead of normal variables? - What is a controlled input? - Why do forms need `preventDefault()`?

> **Key reminder:** State changes → React re-renders.

What Is Component Lifecycle? (10–25 min)

Simple Explanation (Very Important)

Every React component has a **life**:

1. **Mounting** → Component appears on the screen
2. **Updating** → Component changes (state or props change)
3. **Unmounting** → Component is removed from the screen

Real-Life Analogy

Think of a **mobile app**: - You open the app → mounting - You click buttons / data updates → updating - You close the app → unmounting

React lets us run code at **specific moments** in this lifecycle.

Why Do We Need `useEffect`? (25–30 min)

Some code should NOT run during rendering: - Fetching data from a server - Starting a timer - Listening for keyboard/mouse events

That's why React gives us `useEffect`.

> **Rule:** Rendering = UI only. Side-effects = `useEffect`.

`useEffect` Basics (30–45 min)

Syntax

```
import { useEffect } from 'react';

useEffect(() => {
  // code here runs automatically
}, []);
```

Three Important Variants

1️⃣ Run ONCE (on mount)

```
useEffect(() => {
  console.log('Component mounted');
}, []); // empty array = only once
```

2️⃣ Run when something changes

```
useEffect(() => {
  console.log('Name changed:', name);
}, [name]); // runs when name changes
```

3️⃣ Run on every render (rarely used)

```
useEffect(() => {
  console.log('Runs every render');
});
```

> **Teaching rule:** 90% of the time you'll use **option 1 or 2**.

Hands-On 1: Timer Example (45–55 min)

Create `src/day5/Timer.jsx`:

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

function Timer() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds(s => s + 1); // increase every second
    }, 1000);

    // cleanup when component is removed
    return () => {
      clearInterval(interval);
    };
  }, []); // start once

  return (
    <div className="card">
      <h2>Timer</h2>
      <p>Seconds passed: {seconds}</p>
    </div>
  );
}

export default Timer;
```

Teaching Points

- `setInterval` is a side-effect
- Cleanup prevents memory leaks
- Cleanup runs when component unmounts

Hands-On 2: Fetching Data (55–60 min)

Create `src/day5/UsersFetch.jsx`:

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

function UsersFetch() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/users')
      .then(res => res.json())
      .then(data => {
        setUsers(data);
        setLoading(false);
      });
  }, []); // fetch only once

  if (loading) {
    return <p>Loading users...</p>;
  }

  return (
    <div className="card">
      <h2>Users</h2>
      <ul>
        {users.map(user => (
          <li key={user.id}>{user.name}</li>
        ))}
      </ul>
    </div>
  );
}

export default UsersFetch;
```

Using Components in App.jsx

```
import Timer from './day5/Timer.jsx';
import UsersFetch from './day5/UsersFetch.jsx';
```

```
<main className="container">
  <Timer />
  <UsersFetch />
</main>
```

Cleanup Explained (60–70 min)

Why Cleanup Matters

Without cleanup: - Timers keep running - Memory leaks happen - App becomes slow

Cleanup Runs When:

- Component unmounts
- Effect re-runs

Example Cleanup Pattern

```
useEffect(() => {
  // start something
  return () => {
    // stop it
  };
}, []);
```

✅Module 4 Homework — FULL SOLUTION (70–80 min)

Homework Recap

Create `RegisterForm.jsx` with: - Username - Password - Age - Validation rules

✅Complete Solution

Create `src/day4/RegisterForm.jsx`:

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

function RegisterForm() {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [age, setAge] = useState('');
```

```
  const [error, setError] = useState('');

  function handleSubmit(e) {
    e.preventDefault();

    if (!username || !password || !age) {
      setError('All fields are required');
      return;
    }

    if (Number(age) < 10) {
      setError('Age must be 10 or older');
      return;
    }

    setError('');
    alert('Registered: ' + username);

    setUsername('');
    setPassword('');
    setAge('');
  }

  return (
    <form className="card" onSubmit={handleSubmit}>
      <h2>Register</h2>

      <input
        value={username}
        onChange={e => setUsername(e.target.value)}
        placeholder="Username"
      />

      <input
        type="password"
        value={password}
        onChange={e => setPassword(e.target.value)}
        placeholder="Password"
      />

      <input
        type="number"
        value={age}
        onChange={e => setAge(e.target.value)}
        placeholder="Age"
      />

      {error && <p style={{ color: 'red' }}>{error}</p>}
```

```
      <button type="submit">Register</button>
    </form>
  );
}

export default RegisterForm;
```

Final Recap (Very Important)

- Lifecycle = mount → update → unmount
- `useEffect` runs side-effects
- Dependency array controls **when** it runs
- Cleanup prevents bugs
- React apps often fetch data inside `useEffect`

Homework (Module 5)

1. Create `Clock.jsx` that shows current time and updates every second.
2. Fetch posts from: https://jsonplaceholder.typicode.com/posts
3. Show only first 5 posts.

Bonus: Add a loading spinner text.

Instructor Script (Suggested)

- "Rendering is for UI, `useEffect` is for side-effects."
- "Empty dependency array = run once."
- "Cleanup is React being polite before leaving."
- "Every real app uses `useEffect`."

🎉**At this point students are officially past beginner level.**