Internship Report – Week 6, Day 6
Topic: React useEffect Hook – Basic Usage
Name: Hamza Rafique

Work Done Today:
Today I studied the **useEffect Hook** in React, which allows us to perform side effects in functional components. Side effects can include tasks such as fetching data, updating the DOM manually, or setting up subscriptions.

Basic Syntax:

javascript

CopyEdit

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


useEffect(() => {
  // Side effect code here
});
```

Key Concepts Learned:

1. **Purpose of useEffect:**
    - Runs after the component renders.
    - Used for side effects that are not part of the initial rendering logic.
2. **Basic Usage without Dependency Array:**
    - Runs after every render.

javascript

CopyEdit

```
useEffect(() => {
  console.log("Component rendered/updated");
```

});

3. **Usage with Empty Dependency Array []:**
    - Runs only once after the first render (like componentDidMount).

javascript

CopyEdit

```
useEffect(() => {
  console.log("Runs only once after mount");
}, []);
```

4. **Usage with Specific Dependencies:**
    - Runs only when specified variables change.

javascript

CopyEdit

```
useEffect(() => {
  console.log("Count changed");
}, [count]);
```

5. **Cleanup Function:**
    - Returns a function to clean up resources when the component unmounts or before the next effect runs.

javascript

CopyEdit

```
useEffect(() => {
  const timer = setInterval(() => console.log("Tick"), 1000);
  return () => clearInterval(timer); // cleanup
}, []);
```

Key Learnings:

- useEffect replaces lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount in functional components.
- Dependency arrays control when the effect runs.
- Cleanup functions are important for avoiding memory leaks.

Challenges Faced:

- Remembering when to include dependencies in the array to prevent unnecessary re-renders or missed updates.

Next Steps:

- Practice useEffect with API calls.
- Combine useState and useEffect to create dynamic UI updates.