React JS

Hooks

Hooks

What is a Hook?

- A Hook is a special function that lets you “hook into” React features.
  - For example, useState is a Hook that lets you add React state to function components.
- When would I use a Hook?
  - If you write a function component and realize you need to add some state to it, previously you had to convert it to a class. Now you can use a Hook inside the existing function component.

Hooks

useState Hook

useReducer

useEffect Hook

useRef

useLayoutEffect

useState

- The React useState Hook allows us to track state in a function component.
- State generally refers to data or properties that need to be tracking in an application.

Using the useState Hook

- https://reactjs.org/docs/hooks-effect.html

```
import React, { useState } from 'react';
function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

useReducer

- We use useReducer hook when we have many different states being altering with the same action.
- An alternative to useState. useReducer is usually preferable to useState when you have complex state logic that involves multiple sub-values or when the next state depends on the previous one.

Using the Effect Hook

17.ExpAppUseEffect-v16

- https://reactjs.org/docs/hooks-effect.html
- The *Effect Hook* lets you perform side effects in function components.
- The function passed to useEffect will run after the render is committed to the screen.
- Works similar to componentDidMount() or componentDidUpdate()
- The function passed to useEffect will run after the render is committed to the screen.

Using the useEffect Hook

```
import React, { useState, useEffect } from 'react';
function Example() {
  const [count, setCount] = useState(0);
  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

useEffect | axios

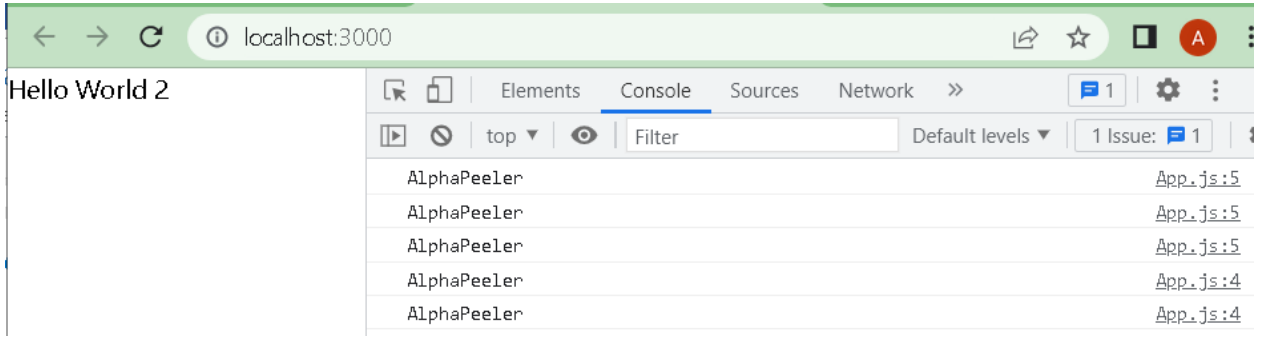
- We will be using axios which is very common library for making api calls. We will be using a public api for dummy data.

- Please install axios first from terminal
- \my-react-app> npm install axios

```
[################...] / reify:form-data: http fetch GET 200 https://registry.
added 2 packages, and audited 1411 packages in 16s
169 packages are looking for funding
  run `npm fund` for details
8 vulnerabilities (6 moderate, 2 high)
To address issues that do not require attention, run:
  npm audit fix
To address all issues (including breaking changes), run:
  npm audit fix --force
```

useEffect

```
import React, { useEffect } from "react";
function EffectTutorial() {
  useEffect( ()=> {
    console.log("AlphaPeeler");
  });
  return <div>Hello World 2</div>
}
export default EffectTutorial;
```



- Note: every time we change the code for example we change value in Hello World and save Ctrl+s then Alphapeeler is logged in console. It means every time state is changed the function we pass in useEffect will be called. useEffect(()=> {}); // highlighted function will be called.

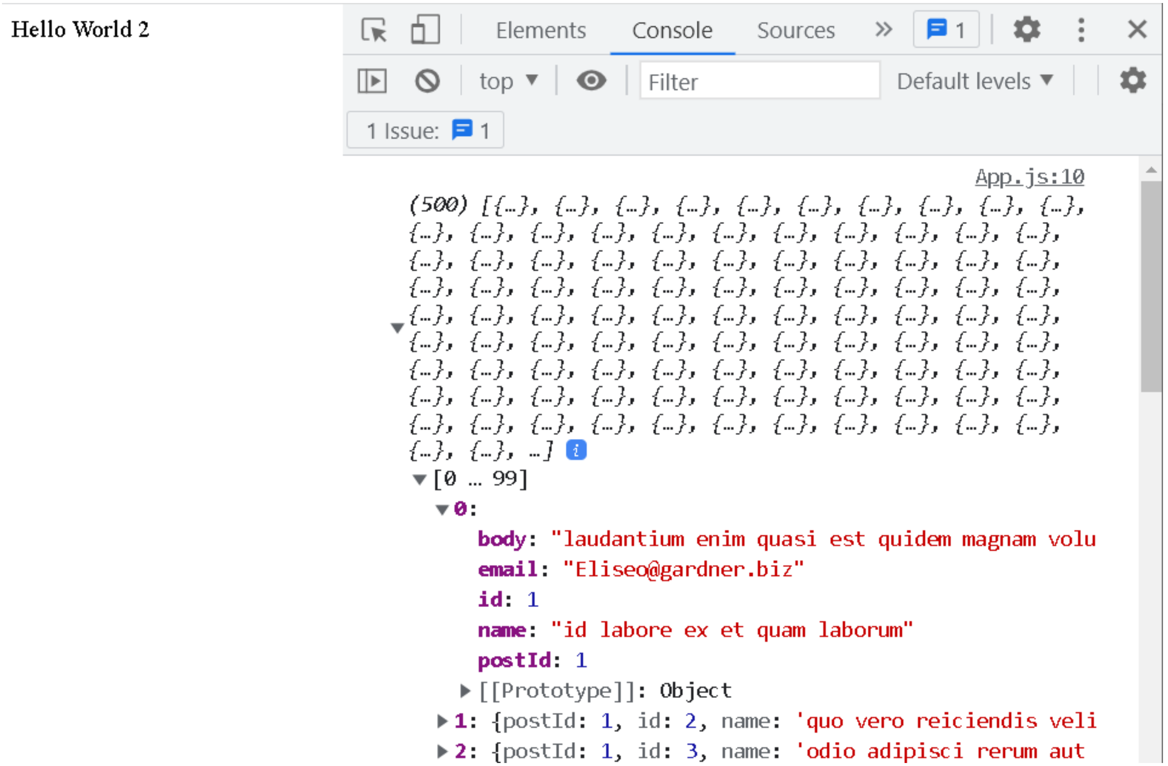
useEffect | axios

- **using useEffect for handling api calls using axios**

```
import React, { useEffect } from "react";
import axios from "axios";
function EffectTutorial() {
  useEffect(() => {
    axios
      .get("https://jsonplaceholder.typicode.com/comments")
      .then((response) => {
        console.log(response.data);
      });
  });
  return <div>Hello World 2</div>
}
export default EffectTutorial;
```

useEffect | axios

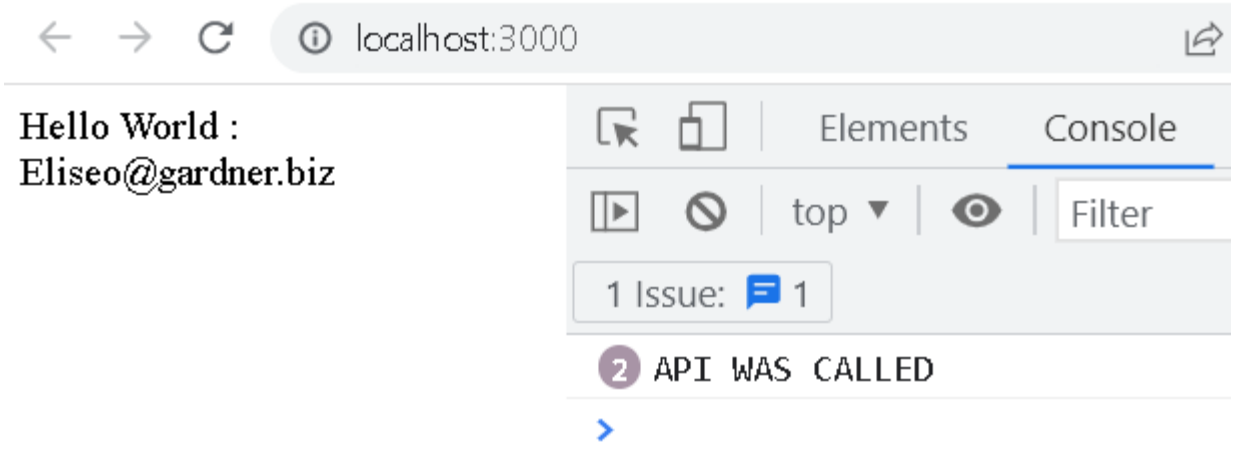
- Displaying data from api to browser screen using useEffect



useEffect | axios

- Now, Assume that we want to access the email address at the first element of array from public data fetched from axios api.

```
import React, { useEffect, useState } from "react";
import axios from "axios";
function EffectTutorial() {
  const [data, setData] = useState("");
  useEffect(() => {
    axios
      .get("https://jsonplaceholder.typicode.com/comments")
      .then((response) => {
        setData(response.data[0].email);
        console.log("API WAS CALLED");
      });
  });
  return <div>Hello World : {data} </div>
}
export default EffectTutorial;
```

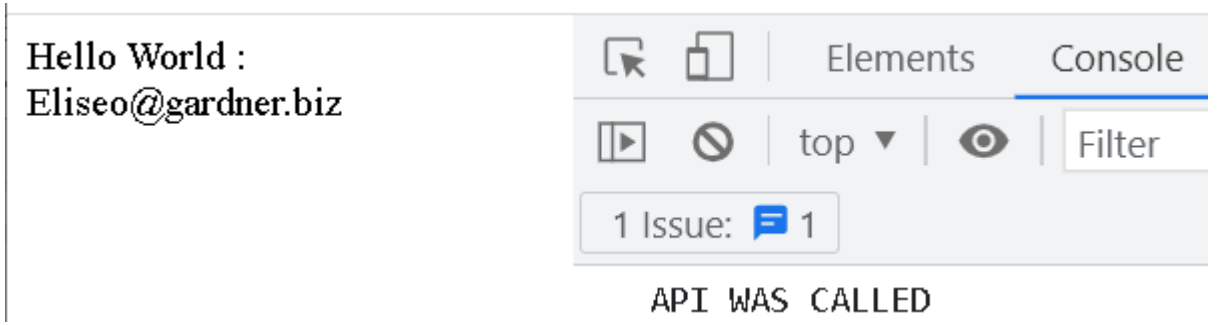


useEffect | axios

- Now, if you look closely, API is called twice. Reason is that what happens with UseEffect, that it will be called every time the page renders. Initially the page renders once and then the page renders again when we make an api call and display the value of data via setData as the state of {data} has changed now.
- **Solution:**
- So how can we just call the useEfect once in the page? UseEffact allows us to pass the array of all states that you wants to listen to in the end (see highlighted text:)

useEffect | axios

```
import React, { useEffect, useState } from "react";
import axios from "axios";
function EffectTutorial() {
  const [data, setData] = useState("");
  useEffect(() => {
    axios
      .get("https://jsonplaceholder.typicode.com/comments")
      .then((response) => {
        setData(response.data[0].email);
        console.log("API WAS CALLED");
      },  [] );
  });
  return <div>Hello World : {data} </div>
}
export default EffectTutorial;
```



useEffect | axios

- Case A- Calling useEffect only once (hence API is loaded once initially), even though counter value changes several times, this is done by passing an empty array [] to useEffect
- Case B- Calling useEffect every time, when counter value changes; (hence API is also loaded every time) this is done by passing an [counter] to useEffect.

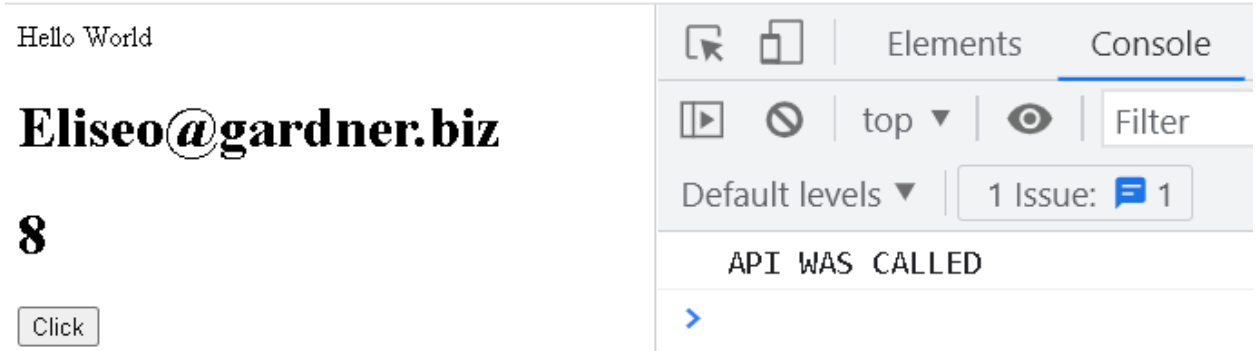
useEffect | axios

```
import React, { useEffect, useState } from "react";
import axios from "axios";
function EffectTutorial() {
  const [data, setData] = useState("");
  const [count, setCount] = useState(0);
  useEffect(() => {
    axios
      .get("https://jsonplaceholder.typicode.com/comments")
      .then((response) => {
        setData(response.data[0].email);
        console.log("API WAS CALLED");
      });
  }, []);
  //}, [count]);
```

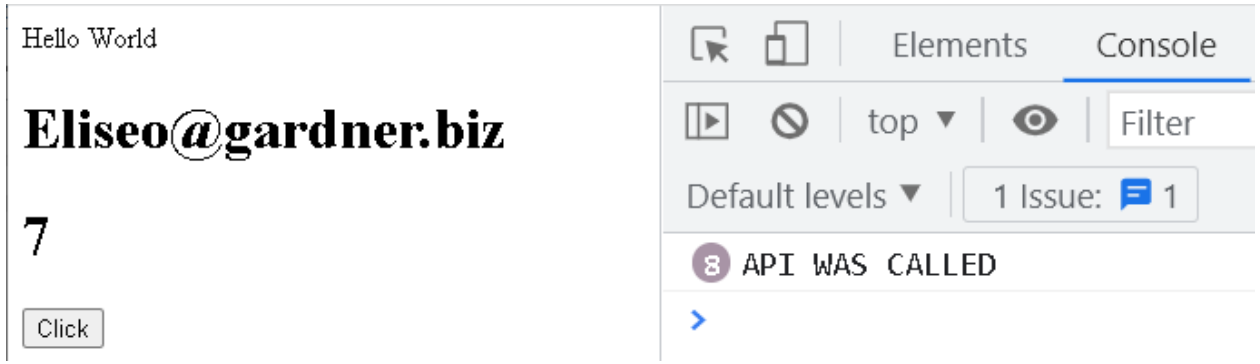
```
return (
    <div>
      Hello World
      <h1>{data}</h1>
      <h1>{count}</h1>
      <button
        onClick={() => {
          setCount(count + 1);
        }}
      >
        Click
      </button>
    </div>
  );
}
export default EffectTutorial;
```

useEffect | axios

- **Case A:** counter incremented by clicking button 8 times, but api loaded once only.



- **Case B:** counter incremented by clicking button 7 times, api also loaded 8 times.

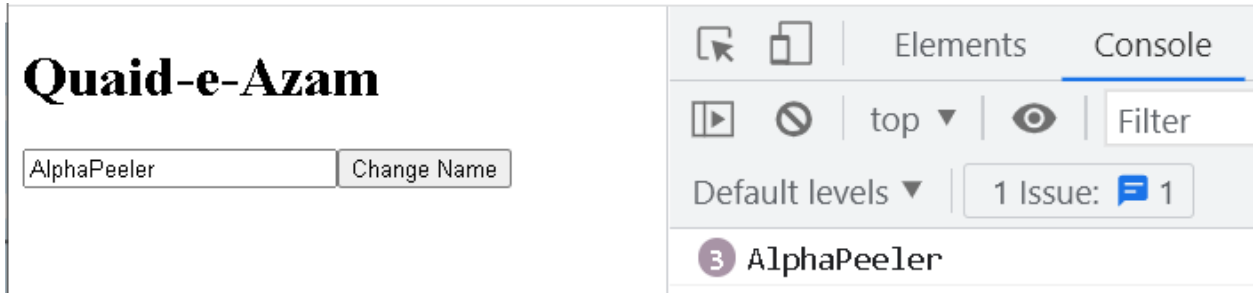


useRef Hook

- useRef hook is the easiest way to access and manipulate DOM elements.
- App.js 01 (When we click on button, it will log the contents input control – using useRef)

useRef Hook

```
import React, { useRef } from "react";
function RefTutorial() {
  const inputRef = useRef(null);
  const onClick = () => {
    console.log(inputRef.current.value);
  };
  return (
    <div>
      <h1>Quaid-e-Azam</h1>
      <input type="text" placeholder="Ex..." ref={inputRef} />
      <button onClick={onClick}>Change Name</button>
    </div>
  );
}
export default RefTutorial;
```



useRef Hook

- App.js 02 (When we click on button, it will focus on input control – using useRef Hook)

```
...
const onClick = () => {
    inputRef.current.focus();
  };
...
```

useRef Hook

- App.js 03 (When we click on button, it will clear the previous input– using useRef Hook)

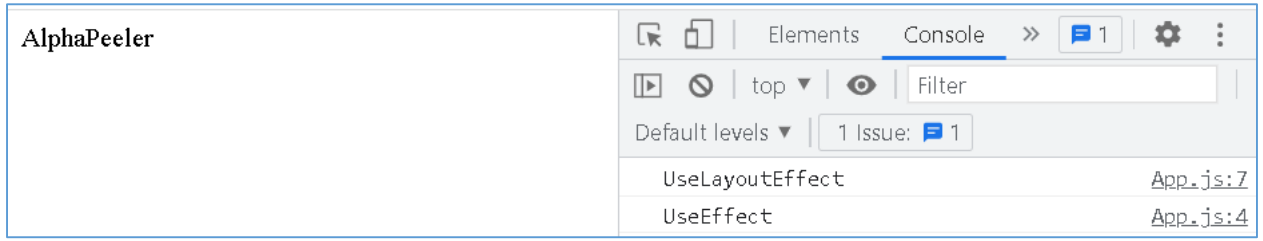
```
...
const onClick = () => {
    inputRef.current.value = "";
  };
...
```

useLayoutEffect Hook

- useLayoutEffect is fundamentally called in an earlier stage of page rendering than the useEffect().
  useLayoutEffect is called before anything is printed to the user.
- So you might use the useLayoutEffect in cases where you want to change the layout of your application before it actually prints out to the user.

useLayoutEffect Hook

```
import { useLayoutEffect, useEffect } from "react";
function LayoutEffectTutorial() {
  useEffect(() => {
    console.log("UseEffect");
  }, []);
  useLayoutEffect(() => {
    console.log("UseLayoutEffect");
  }, []);
  return <div>AlphaPeeler</div>
}
export default LayoutEffectTutorial;
```



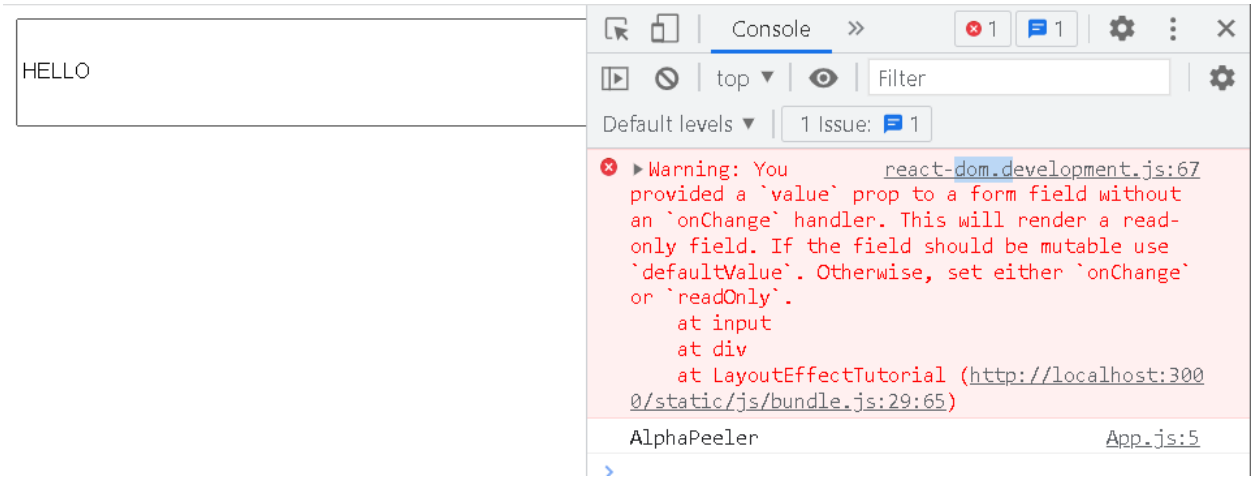
- You can see from the console log that UseLayoutEffect is called earlier then UseEffect even though it was called later in the program.

useLayoutEffect Hook

- App.js 02 (initial value of input was AlphaPeeler which was set via useLayoutEffect. This can be seen in the console log as well, but this value later on get changed to HELLO via useEffect after the page rendering is completed)

useLayoutEffect Hook

```
import { useLayoutEffect, useEffect, useRef } from "react";
function LayoutEffectTutorial() {
  const inputRef = useRef(null);
  useLayoutEffect(() => {
    console.log(inputRef.current.value);
  }, []);
  useEffect(() => {
    inputRef.current.value = "HELLO";
  }, []);
  return (
    <div className="App">
      <input ref={inputRef} value="AlphaPeeler" style={{ width: 400, height: 60 }} />
    </div>
  );
}
export default LayoutEffectTutorial;
```



UseLayoutEffect Hook

- App.js 03 (commenting out the functionality of changing the current value of input to Hello, results in showing the value AlphaPeeler which was set by useLayoutEffect before page rendering starts on browser.)

```
...
useEffect(() => {
    //inputRef.current.value = "HELLO";
  }, []);
...
```