Lab Recitation – 8

Name : Sharon Elango Batch : 3 Roll No : 62

Pre-Lab Exercises for Experiment-8

□ Pre-Lab Activity 1: Basic useState Implementation

Create a component that:

1. Displays a message: “Hello Student”

2. Has a button labeled “Change Message”

3. On clicking the button:

o The message should change to “Welcome to React State”

#Activity1.jsx

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

function Activity1() {
  const [message, setMessage] = useState("Hello Student");

  return (
    <div style={{ padding: '20px', border: '2px solid blue' }}>
      <h2>Activity 1</h2>
      <p>{message}</p>
```

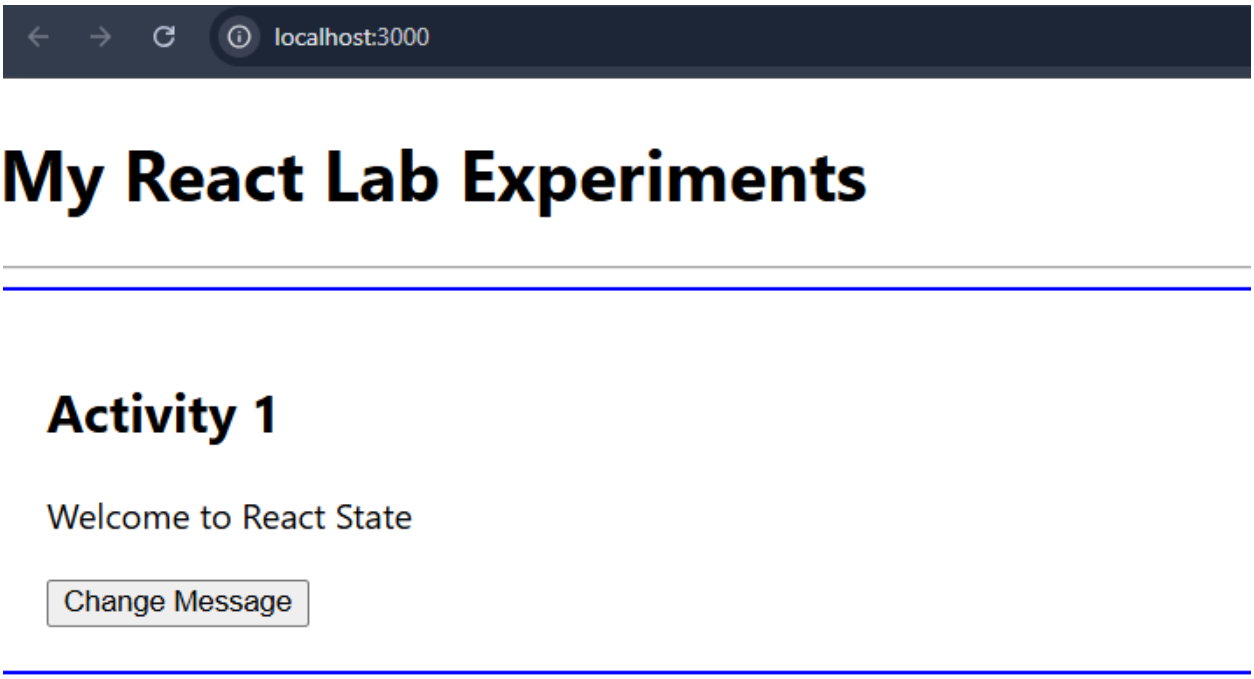
```
    <button onClick={() => setMessage("Welcome to React State")}>
      Change Message
    </button>
   </div>
  );
}

export default Activity1; // This line is crucial!
```

#App.jsx

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

function App() {
  const [message, setMessage] = useState("Hello Student");

  return (
   <div style={{ textAlign: 'center', marginTop: '50px' }}>
    <h1>{message}</h1>
    <button onClick={() => setMessage("Welcome to React State")}>
      Change Message
    </button>
```

```
    </div>
  );
}

export default App;
```

#output :



Pre-Lab Activity 2: Simple Counter

Create a simple counter that:

1. Displays a number (initial value 0)

2. Has two buttons:

o Increase

o Decrease

3. Clicking buttons updates the number accordingly

#Activity2.jsx

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

function Activity2() {
  const [count, setCount] = useState(0);

  return (
    <div style={{ border: '1px solid #ccc', padding: '10px', margin: '10px' }}>
      <h3>Activity 2: Simple Counter</h3>
      <p>Current Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increase</button>
      <button onClick={() => setCount(count - 1)}>Decrease</button>
```

```
    </div>
  );
}

export default Activity2;
```

#App.jsx

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

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

  return (
    <div style={{ textAlign: 'center', marginTop: '50px' }}>
      <h2>Counter: {count}</h2>
      <button onClick={() => setCount(count + 1)}>Increase</button>
      <button onClick={() => setCount(count - 1)} style={{ marginLeft:
'10px' }}>
        Decrease
      </button>
    </div>
```

```
  );
}

export default App;
```

#output :

? Pre-Lab Activity 3: Controlled Text Input

1. Create an input field.

2. Display the entered text below the input field in real-time.

Example:

Input: Rahul

Output: You entered: Rahul

#Activity3.jsx

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

function Activity3() {
  const [text, setText] = useState("");

  return (
    <div style={{ border: '1px solid #ccc', padding: '10px', margin: '10px' }}>
      <h3>Activity 3: Controlled Input</h3>
      <input
        type="text"
        placeholder="Enter text..."
        onChange={(e) => setText(e.target.value)}
```

```
      />
      <p>You entered: {text}</p>
    </div>
  );
}

export default Activity3;
```

#App.jsx

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

function App() {
  const [text, setText] = useState("");

  return (
    <div style={{ padding: '20px' }}>
      <input
        type="text"
        placeholder="Type something..."
        onChange={(e) => setText(e.target.value)}
      />
```

```
      <p>You entered: {text}</p>
    </div>
  );
}

export default App;
```

#output :

SE AIML, Year : 2025-2026

You entered: SE AIML, Year : 2025-2026

Pre-Lab Activity 4: Multiple State Variables

1. Create two input fields:

o Name

o City

2. Display both values dynamically below.

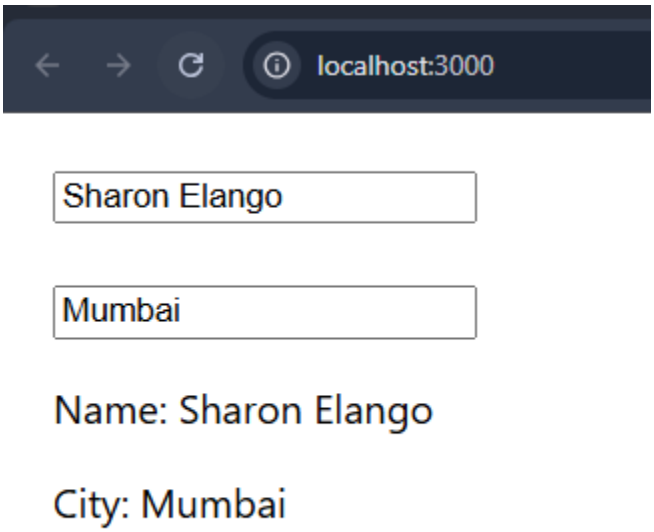
#Activity4.jsx

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

function Activity4() {
  const [name, setName] = useState("");
  const [city, setCity] = useState("");

  return (
    <div style={{ border: '1px solid #ccc', padding: '10px', margin: '10px' }}>
      <h3>Activity 4: Multiple States</h3>
      <input placeholder="Name" onChange={(e) => setName(e.target.value)} />
      <input placeholder="City" onChange={(e) => setCity(e.target.value)} />
      <p>Name: {name} | City: {city}</p>
    </div>
  );
}

export default Activity4;
```

#App.jsx

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

function App() {
  const [name, setName] = useState("");
  const [city, setCity] = useState("");

  return (
    <div style={{ padding: '20px' }}>
      <input placeholder="Name" onChange={(e) =>
setName(e.target.value)} />
      <br /><br />
      <input placeholder="City" onChange={(e) =>
setCity(e.target.value)} />
      <p>Name: {name}</p>
      <p>City: {city}</p>
    </div>
  );
}

export default App;
```

#output :



Pre-Lab Activity 5: Single State Object

Modify Activity 4 to:

1. Use only ONE state object.

2. Store both name and city inside the same object.

3. Update specific field without overwriting the other.

#Activity5.jsx

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

function Activity5() {
  const [user, setUser] = useState({ name: "", city: "" });
```

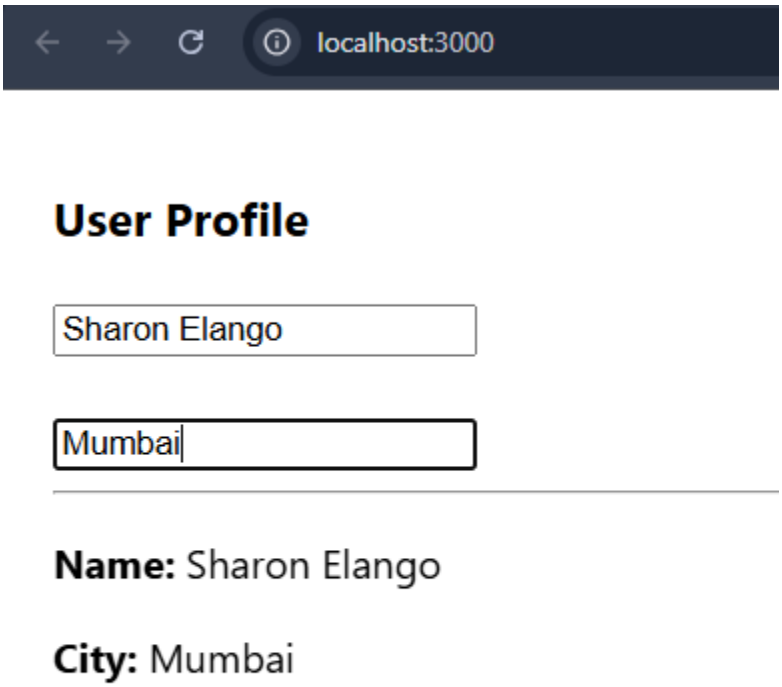
```
  const updateField = (e) => {
    const { name, value } = e.target;
    // ...user copies the old object, [name]: value updates only the changed field
    setUser({ ...user, [name]: value });
  };

  return (
    <div style={{ border: '1px solid #ccc', padding: '10px', margin: '10px' }}>
      <h3>Activity 5: State Object</h3>
      <input name="name" placeholder="Name" onChange={updateField} />
      <input name="city" placeholder="City" onChange={updateField} />
      <p>Output: {user.name} lives in {user.city}</p>
    </div>
  );
}

export default Activity5;
```

#App.jsx

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


function App() {
  const [user, setUser] = useState({ name: "", city: "" });


  const handleChange = (e) => {
    const { name, value } = e.target;
    // ...user preserves the other fields while updating only the target field
    setUser({ ...user, [name]: value });
  };


  return (
    <div style={{ padding: '20px' }}>
      <h3>User Profile</h3>
      <input name="name" placeholder="Name" onChange={handleChange} />
      <br /><br />
      <input name="city" placeholder="City" onChange={handleChange} />
      <hr />
```

```
      <p><strong>Name:</strong> {user.name}</p>
      <p><strong>City:</strong> {user.city}</p>
    </div>
  );
}

export default App;
```

#output :



Pre-Lab Activity 6: Simple Validation Logic

1. Create a single input field for email.

2. When a button is clicked:

o Show error message if field is empty.

o Show error if email does not contain “@”.

o Show success message if valid.

#Activity6.jsx

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

function Activity6() {
  const [email, setEmail] = useState("");
  const [message, setMessage] = useState("");

  const handleValidation = () => {
    if (email === "") {
      setMessage("Error: Email cannot be empty.");
    } else if (!email.includes("@")) {
      setMessage("Error: Email must contain '@'.");
    } else {
```

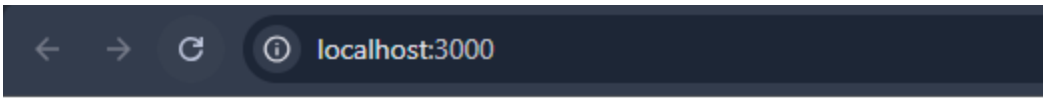
```
    setMessage("Success: Email is valid!");
   }
  };

  return (
   <div style={{ border: '1px solid #ccc', padding: '10px', margin: '10px'
}}>
    <h3>Activity 6: Email Validation</h3>
    <input
     type="text"
     placeholder="Enter email"
     onChange={(e) => setEmail(e.target.value)}
    />
    <button onClick={handleValidation}>Validate</button>
    <p>{message}</p>
   </div>
  );
}

export default Activity6;
```

#App.jsx

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

function App() {
  const [email, setEmail] = useState("");
  const [message, setMessage] = useState("");

  const validate = () => {
    if (email === "") {
      setMessage("Error: Field is empty");
    } else if (!email.includes("@")) {
      setMessage("Error: Email must contain '@'");
    } else {
      setMessage("Success: Valid Email!");
    }
  };

  return (
    <div style={{ padding: '20px' }}>
      <input
        type="text"
```

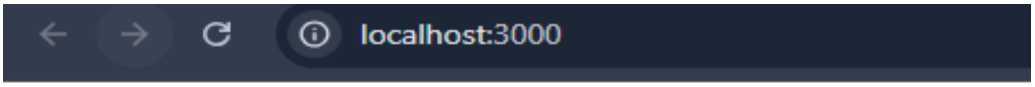
```
      placeholder="Enter Email"
      onChange={(e) => setEmail(e.target.value)}
    />
    <button onClick={validate} style={{ marginLeft: '10px' }}>
      Check
    </button>
    <p style={{ color: message.startsWith("Error") ? "red" : "green" }}>
      {message}
    </p>
  </div>
 );
}

export default App;
```

#output :



sharonelango10@student.s Check

Success: Valid Email!



sharonelango10student.sfit. Check

Error: Email must contain '@'

Pre-Lab Questions

1. What is state in React?
   State is a built-in React object used to store data or information about the component. It represents the "memory" of a component. When the state object changes, React perceives that the UI is now out of date and automatically updates (re-renders) the component to reflect those changes.
2. What is the difference between normal variable and state variable?

   Normal Variables A normal variable (declared using `let` or `const`) is local to the function. Every time React re-renders a component, the function is executed from top to bottom, which means all normal variables are re-initialized to their original values. Furthermore, changing a normal variable does not notify React that something has changed; therefore, the UI remains exactly as it was, and the user will not see the updated data on the screen.

   State Variables A state variable (created via the `useState` hook) is managed by the React engine outside of the component's immediate function scope. This allows the variable to "survive" or persist across multiple re-renders. When you update a state variable using its setter function (e.g., `setCount`), React is explicitly notified that the data has changed. React then automatically triggers a re-render of the component, calculates the difference, and updates the DOM so the user sees the new value instantly.

3. What is a controlled component?
   A **controlled component** is an input element (like <input>, <select>, or <textarea>) whose value is controlled by React state rather than the DOM itself.

- The input's value is bound to a state variable.
- An onChange function updates that state whenever the user types. This ensures the React state is the **"Single Source of Truth"** for the form data.

4. Why does React re-render when state changes?
   React uses a **Declarative** approach. Instead of you manually changing the text on a screen (Imperative), you tell React what the data looks like. When state changes, React runs the component function again to calculate a new **Virtual DOM** tree. It then compares this new tree with the old one (a process called **Diffing**) and updates only the necessary parts of the actual browser DOM to keep the UI in sync with the data.

5. Why do we use the spread operator in object state?
   In React, state is considered **immutable** (it should not be changed directly). When using an object in state, the setter function from useState **replaces** the entire old object with the new one you provide—it does not merge them automatically.
   We use the **spread operator (...)** to:

1. **Copy** all existing properties from the old state object into a new object.
2. **Overwrite** only the specific property we want to change.

   Example:

   ```
   // If state is { name: "Rahul", city: "Mumbai" }
   setUser({ ...user, city: "Pune" });
   // Without '...user', the 'name' property would be lost!
   ```