# Task-1

**Aim:** Create Simple Hello World Component

**Theoretical Background:**

* Understand the basic structure of a React component.
* Write JSX (JavaScript XML) to define the UI structure.
* Render a React component within an HTML element.
* Display a "Hello World" message using your created component.

## Source Code: index.js

import React from 'react';

import ReactDOM from 'react-dom/client'; import './index.css';

// import App from './App';

import Hello from './mycomponent';

import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root')); root.render(

<React.StrictMode>

{/\* <App /> \*/}

<Hello/>

</React.StrictMode>

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals reportWebVitals();

**mycomponent.js**

// Create Simple Hello World Component function Hello() {

return (

<div>

<h1> Hello World</h1>

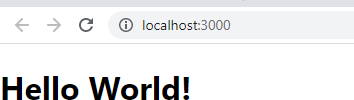
</div>

);

}

export default Hello;

**Output:**



# Task-2

**Aim:** Create Basic/Static Product Page Using nested components & apply styles ( exploring JSX Syntax )

**Theoretical Background:**

1. Setting Up the Project: Start by setting up your project environment with the necessary tools and dependencies. You'll need a code editor, Node.js, and a package manager like npm or yarn.
2. Creating Components: Break down your product page into smaller reusable components. For this example, let's create components for the header, product details, and footer.
3. Applying Styles: You can use CSS or a CSS-in-JS library like styled-components to style your components. For simplicity, let's use inline styles.
4. Rendering the App: Finally, render your App component in the index.js file.

## Source Code: product.js

function Product(){

const ProductCard = ({ product }) => (

<div className="product-card" style={{float:"left",paddingLeft:"20px"}}>

<img src={product.image} alt={product.name} />

<h2>{product.name}</h2>

<p>RS {product.price}</p>

<button> Buy Now

</button>

</div>

);

let data = [

{

product:"Tea:",

price:"100", image:'/images/1.jpg'

},

{

product: "Coffee:",

price:"250", image:'/images/1.jpg'

},

];

return (

<div className="product-page" >

<h1>Products</h1>

{data.map((product) => (

// <li key={product.id}>

<ProductCard product={product} />

// </li>

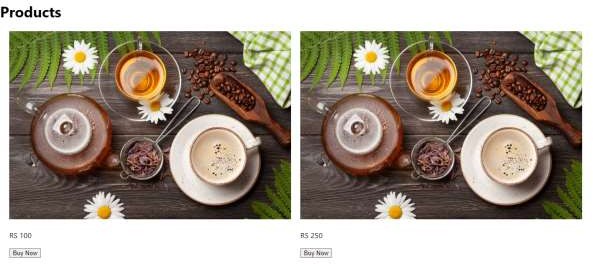
))}

</div>

);

}

export default Product;

**Output:**

# Task-3

**Aim:** Button Enable/Disable based on checkbox click event

**Theoretical Background:**

Enabling or disabling a button based on a checkbox click event is a common interaction pattern in user interfaces. This functionality is often used to ensure that certain actions are only possible when specific conditions are met. Let's discuss the theoretical background of how this can be achieved in the context of web development, specifically using HTML, CSS, and JavaScript.

## Source Code:

**product.js**

import React, { useState } from "react";

function Product() {

const [enableButton, setEnableButton] = useState(false); // State to track button enabling

const data = [

{

product:"Tea:",

price:"100", image:'/images/imagetea.jpg'

},

{

product: "Coffee:", price:"250", image:'/images/imagetea2.jpg'

},

{

product:"Tea:",

price:"100", image:'/images/41.jpg'

},

];

const toggleButton = () => { setEnableButton(!enableButton);

};

return (

<div>

<label>

<input type="checkbox" onChange={toggleButton} /> Enable Button

</label>

{data.map((product, i) => (

<div

className="card"

key={i} style={{

float: "left", paddingLeft: "2rem", paddingRight: "2rem", border: "1px solid black",

}}

>

<img src={product.image} height={"300px"} width={"300px"} alt={product.name} />

<h1>{product.name}</h1>

<p>Rs.{product.price}</p>

<p>{product.model}</p>

<p>{product.comp}</p>

</div>

))}

<button disabled={!enableButton} style={{ marginTop: "1rem" }}

>

Click Me

</button>

</div>

);

}

export default Product;

**Output:**





# Task-4

**Aim:** Create Simple Counter Page Using Use State Hook (share data between components)

**Theoretical Background:**

Creating a simple counter page using the useState hook in React allows you to manage and share state between components. This is a fundamental concept in React and helps in building dynamic and interactive user interfaces. Here's the theoretical background on how you can achieve this:

1. Understanding useState Hook: The useState hook is a built-in React hook that allows you to add state to functional components. It returns a stateful value and a function to update it. When you call useState, it returns an array with two elements: the current state value and a function to update that value.
2. Setting Up the Counter Component: Let's assume you have a simple React application and you want to create a counter page. Here's how you can do it:
3. Explaining the Code:

In the Counter component, the useState hook is used to initialize the count state to 0.

The increment and decrement functions update the count state by modifying it using the setCount function.

The return block contains the JSX that displays the current count value and two buttons to increment and decrement the count.

In the main App component, the Counter component is incorporated to display the counter page.

1. State Sharing: In this example, the count state is local to the Counter component. If you want to share state between different components, you can lift the state up to a common parent component and pass it down as props.

## Source Code:

**Product.js**

import React, { useState } from 'react'; import Counter from './counter';

function Product() {

const [sharedCount, setSharedCount] = useState(0);

const ProductCard = ({ product }) => (

<div className="product-card" style={{float:"left",paddingLeft:"20px"}}>

<img src={product.image} alt={product.name} />

<h2>{product.name}</h2>

<p>RS {product.price}</p>

<button> Buy Now

</button>

</div>

);

let data = [

{

product:"Tea:",

price:"100", image:'/images/1.jpg'

},

{

product: "Coffee:", price:"250", image:'/images/1.jpg'

},

];

return (

<div className="product-page">

<h1>Products</h1>

{data.map((product, index) => (

<ProductCard key={index} product={product} />

))}

<Counter sharedCount={sharedCount} setSharedCount={setSharedCount} />

</div>

);

}

## Counter.js

import React, { useState } from 'react';

function Counter({ sharedCount, setSharedCount }) { const [count, setCount] = useState(0);

const increment = () => { setCount(count + 1);

setSharedCount(count + 1);

};

const decrement = () => { setCount(count - 1);

setSharedCount(count - 1);

};

return (

<div>

<h2>Counter</h2>

<p>Local Count: {count}</p>

<button onClick={increment}>Increment</button>

<button onClick={decrement}>Decrement</button>

{/\* <p>Shared Count from Product: {sharedCount}</p> \*/}

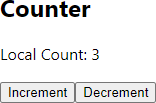
</div>

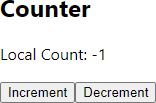
);

}

export default Counter;

**Output:**





# Task-5

**Aim:** Create Task list Page and conditionally render Tasks based on hard-coded variables & Array

**Theoretical Background:**

* 1. Component Structure and Composition: You learn how to break down your user interface into reusable components. In this case, you have a TaskList component that handles the rendering of tasks and an App component that incorporates the TaskList component.
  2. Conditional Rendering: You understand how to conditionally display different content in your UI based on certain conditions. This is crucial for creating dynamic and interactive user interfaces that respond to user input or data changes.
  3. State Management with Arrays: You gain experience in working with arrays to manage data in your application. In this case, the array of tasks holds the data that's used to render the task list. You also see how to map over the array to generate the UI elements.
  4. Hard-Coded Data and Dynamic Rendering: You learn how to use hard-coded data to initially populate your UI, which is essential for testing and development. This experience can be extended to dynamically rendering data from APIs or databases.
  5. CSS Styling and Styling Differentiation: You get practice in applying CSS styles to your components. Styling is crucial for creating visually appealing and user-friendly interfaces. Additionally, you learn how to use CSS classes to differentiate between completed and incomplete tasks.
  6. React Hooks (useState): While the example didn't include complex state management, you were introduced to the concept of using the useState hook to manage state within functional components. This is a fundamental concept in React that's used extensively for more advanced scenarios.
  7. HTML and JSX: You become familiar with writing JSX, which is similar to HTML but used within the context of React components. Understanding how to structure JSX and combine it with JavaScript logic is a foundational skill.
  8. UI Interactivity: You understand how to create an interactive user interface where elements respond to user actions, such as clicking buttons or checkboxes. This is essential for creating engaging and user-friendly applications.

## Source Code: product.js

import React, { useState } from 'react';

import Counter from './counter'; // Make sure to provide the correct path import TaskList from './tasklist'; // Make sure to provide the correct path

function App() {

const [sharedCount, setSharedCount] = useState(0);

const [showCompletedTasks, setShowCompletedTasks] = useState(true);

const ProductCard = ({ product }) => (

<div className="product-card" style={{ float: 'left', margin: '10px', border: '2px solid red', padding: '15px' }}>

{/\* ... your product card content ... \*/}

</div>

);

const handleCheckboxChange = () => { setShowCompletedTasks(!showCompletedTasks);

};

const data = [

{

product: 'Tea:',

price: '100',

image: '/images/1.jpg',

},

{

product: 'Coffee:', price: '250',

image: '/images/1.jpg',

},

{

product: 'Tea:',

price: '100',

image: '/images/1.jpg',

},

];

return (

<div className="product-page">

<h1>Products</h1>

{data.map((product, index) => (

<ProductCard key={index} product={product} />

))}

<Counter sharedCount={sharedCount} setSharedCount={setSharedCount} />

<hr />

<div>

<label>

<input type="checkbox" checked={showCompletedTasks} onChange={handleCheckboxChange} />

Show Completed Tasks

</label>

<TaskList showCompleted={showCompletedTasks} />

</div>

</div>

);

}

export default App;

**tasklist.js**

import React from 'react';

function TaskList({ showCompleted }) { const tasks = [

{ id: 1, title: 'Task 1', completed: true },

{ id: 2, title: 'Task 2', completed: false },

{ id: 3, title: 'Task 3', completed: true },

// ... add more tasks ...

];

const filteredTasks = showCompleted ? tasks : tasks.filter(task =>

!task.completed);

return (

<div>

<h2>Task List</h2>

{filteredTasks.length === 0 ? (

<p>No tasks to display.</p>

) : (

<ul>

{filteredTasks.map(task => (

<li key={task.id} style={{ textDecoration: task.completed ? 'line- through' : 'none' }}>

{task.title}

</li>

))}

</ul>

)}

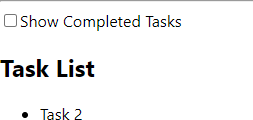
</div>

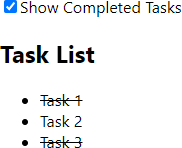
);

}

export default TaskList;

**Output:**





# Task-6

**Aim:** Show(render) List of Students (array of student objects) who are having less marks than ‘35’ using Loop and Conditional Rendering: (filter function, map function)

**Theoretical Background:**

1. Array Filtering: You learn how to use the filter function to select specific elements from an array based on a condition. This is essential when you need to work with a subset of data that meets certain criteria.
2. Array Mapping: You understand how to use the map function to transform each element of an array into something else, such as JSX elements in this case. This is useful for creating UI components from your data.
3. Conditional Rendering: Through the filter and map functions, you see how conditional rendering works. You're rendering only those students who meet the condition (less than 35 marks) and using JSX to present this data in a user-friendly format.
4. Key Prop: You learn about the importance of the key prop when rendering lists of elements in React. The key prop helps React efficiently update and manage lists.
5. JSX Integration: You see how to integrate JavaScript logic (like array filtering and mapping) into JSX. This combination of languages is a key aspect of React development.
6. Data Handling: You gain experience in working with real-world data scenarios, including filtering and transforming data to be presented to users.
7. UI Building Blocks: This exercise helps you build skills in creating UI components using JSX and applying basic styling, which are foundational skills for web development.

## Source Code: product.js

import React from 'react';

import Counter from './counter'; // Make sure to provide the correct path import TaskList from './tasklist'; // Make sure to provide the correct path import StudentList from './studentlist'; // Make sure to provide the correct path

function App() {

const [sharedCount, setSharedCount] = React.useState(0);

const [showCompletedTasks, setShowCompletedTasks] = React.useState(true);

const ProductCard = ({ product }) => (

<div className="product-card" style={{ float: 'left', margin: '10px', border: '2px solid red', padding: '15px' }}>

{/\* ... your product card content ... \*/}

</div>

);

const handleCheckboxChange = () => { setShowCompletedTasks(!showCompletedTasks);

};

const data = [

{

product: 'Tea:',

price: '100',

image: '/images/imagetea.jpg',

},

{

product: 'Coffee:',

price: '250',

image: '/images/imagetea2.jpg',

},

{

product: 'Tea:',

price: '100',

image: '/images/41.jpg',

},

];

const students = [

{ id: 1, name: 'Prachi', marks: 40 },

{ id: 2, name: 'Raj', marks: 30 },

{ id: 3, name: 'Chaku', marks: 25 },

// ... add more students ...

];

return (

<div className="product-page">

<h1>Products</h1>

{data.map((product, index) => (

<ProductCard key={index} product={product} />

))}

<Counter sharedCount={sharedCount} setSharedCount={setSharedCount} />

<hr />

<div>

<label>

<input type="checkbox" checked={showCompletedTasks} onChange={handleCheckboxChange} />

Show Completed Tasks

</label>

<TaskList showCompleted={showCompletedTasks} />

<StudentList students={students} />

</div>

</div>

);

}

export default App;

**studentlist.js**

import React from 'react';

function StudentList({ students }) {

const filteredStudents = students.filter(student => student.marks < 35);

return (

<div>

<h2>Students with Less than 35 Marks</h2>

{filteredStudents.length === 0 ? (

<p>No students with less than 35 marks.</p>

) : (

<ul>

{filteredStudents.map(student => (

<li key={student.id}>

{student.name} - Marks: {student.marks}

</li>

))}

</ul>

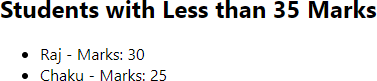
)}

</div>

);

}

export default StudentList;

**Output:**

# Task-7

**Aim:** Create Simple SPA Website using react router package for routing

**Theoretical Background:**

* Routing in React: You'll understand how to set up routing in a React application using the react-router-dom package.
* Creating Components: You'll learn how to create components for different sections/pages of your website.
* Navigation: You'll discover how to create navigation links and use the Link component for seamless navigation between pages.
* Conditional Rendering: You'll see how the Switch component allows you to conditionally render specific components based on the route.
* Single Page Application Concept: You'll become familiar with the concept of Single Page Applications (SPAs), where the content changes without full page reloads, enhancing user experience.

## Source Code:

Approuter.js

import React from 'react';

import { BrowserRouter as Router, Route, Link } from 'react-router-dom'; import Home from './home';

import About from './about'; import Contact from './contact';

const AppRouter = () => { return (

<Router>

<nav>

<ul>

<li>

<Link to="/">Home</Link>

</li>

<li>

<Link to="/about">About</Link>

</li>

<li>

<Link to="/contact">Contact</Link>

</li>

</ul>

</nav>

<Route path="/" exact component={Home} />

<Route path="/about" component={About} />

<Route path="/contact" component={Contact} />

</Router>

);

};

export default AppRouter;

App.js

import logo from './logo.svg'; import './App.css';

function App() { return (

<div className="App">

<header className="App-header">

<img src={logo} className="App-logo" alt="logo" />

<p>

Edit <code>src/App.js</code> and save to reload.

</p>

<a

className="App-link"

href="https://developer.mozilla.org/en-US/docs/Glossary/SPA" target="\_blank"

rel="noopener noreferrer"

>

Learn React

</a>

</header>

</div>

);

}

export default App;

**Output:**



# Task-8

**Aim:** Create a Simple Login Page and Validate it.

**Theoretical Background:**

1: Set Up Your Project Create a new React application using create-react-app or your preferred method

Step 2: Create the Login Component Inside the src folder, create a new component named Login.js

Step 3: Use the Login Component Replace the contents of src/App.js with the following code to use the Login component

Step 4: Run the App

## Source Code:

Loginform.js

import React, { useState } from 'react';

const LoginForm = ({ handleLogin }) => {

const [username, setUsername] = useState(''); const [password, setPassword] = useState(''); const [error, setError] = useState('');

const handleSubmit = (e) => { e.preventDefault();

// Basic validation

if (!username || !password) { setError('Please fill in all fields.'); return;

}

// Simulate login validation (you can replace this with actual API calls) if (username === 'user' && password === 'password') {

handleLogin();

} else {

setError('Invalid username or password.');

}

};

return (

<div>

<h2>Login</h2>

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

<form onSubmit={handleSubmit}>

<div>

<label>Username:</label>

<input

type="text" value={username}

onChange={(e) => setUsername(e.target.value)}

/>

</div>

<div>

<label>Password:</label>

<input

type="password" value={password}

onChange={(e) => setPassword(e.target.value)}

/>

</div>

<button type="submit">Login</button>

</form>

</div>

);

};

export default LoginForm;

App.js

import React, { useState } from 'react'; import './App.css';

import LoginForm from './loginform';

function App() {

const [isLoggedIn, setIsLoggedIn] = useState(false);

const handleLogin = () => { setIsLoggedIn(true);

};

const handleLogout = () => { setIsLoggedIn(false);

};

return (

<div className="App">

<header className="App-header">

{isLoggedIn ? (

<div>

<h1>Welcome, User!</h1>

<button onClick={handleLogout}>Logout</button>

</div>

) : (

<LoginForm handleLogin={handleLogin} />

)}

</header>

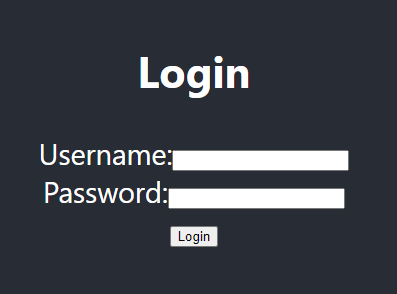
</div>

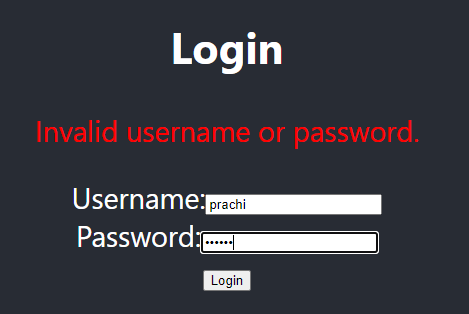
);

}

export default App;

## Output:



**Leaning Outcome:**

CO3: Implement UX-UI using React.js fíamewoík and Responsive web designing foí Web Applications.