23CS2049-Web Technology Lab-2025

Exercise 7

React JS Frontend Application Development

Refer this Video Demonstration:

https://youtu.be/oyh_GpR1qYU?si=GZ5cCbHnuFMNk8B4

Objective:

To develop a responsive and interactive frontend application using React JS, integrating modern web development concepts such as functional components, JSX, state management with Hooks, user input handling, and event-driven programming.

Technical Concepts Description

React JS

- A popular JavaScript library for building interactive and dynamic user interfaces.
- Allows developers to create reusable components that manage their own state and render efficiently.

Functional Components

- Components written as JavaScript functions.
- Focus on simplicity and readability.
- Can use Hooks to manage state and lifecycle events.

JSX (JavaScript XML)

- Syntax extension for JavaScript that looks like HTML.
- Enables easy rendering of UI elements in React components.
- Supports embedding JavaScript expressions directly within HTML-like markup.

Bootstrap Integration

- CSS framework used for styling and responsive design.
- Provides pre-built classes for buttons, forms, layouts, and grids.
- Helps in creating professional-looking interfaces quickly.

React Hooks – useState()

- A Hook that allows functional components to manage state (data).
- Provides a way to store and update dynamic values that trigger UI re-rendering.
- State is a JavaScript object that holds dynamic data in a component.
- When state changes, the component re-renders automatically to reflect the new data.
- **Import useState to App.jsx**

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

- **Declare state variables using useState() method**

```
const [name, setName] = useState("")
```

name ⟶ current state value (initially empty string "").
setName ⟶ function to update the state.
useState("") ⟶ initializes the state with an empty string.

- **Update the state from User Input:-** onChange() event needs to be triggered

Example:

```
<input type="text" placeholder="Enter your name"
    value={name}
    onChange={(e)=>setName(e.target.value)}
/>
```

value={name}
⟶ input value is bound to name (state).

onChange={(e) => setName(e.target.value)}
⟶ Updates state whenever the user types.

Event Handling

- React supports events like onClick, onChange, onSubmit, etc.
- Allows developers to define functions that respond to user actions and update the UI dynamically.

Example:
onClick

- Triggered when an element is clicked.
- Commonly used for buttons, links, icons, etc.

```
function App() {

    const handleClick = () => {
        alert("Button clicked!");
    };

    return <button onClick={handleClick}>Click Me</button>;
}
export default App
```

- React uses camelCase (onClick) instead of HTML lowercase (onclick).

- Event handler is a function, handleClick is a **function declared inside the component.** It is called when the button is clicked.

Display Results using conditional Rendering

```
import { useState } from "react";
function App() {
 //state variables
 const [num1, setNum1] = useState(0);
 const [num2, setNum2] = useState(0);
 const [result, setResult] = useState(null);
 //Event handler function
 const handleAddition = () => {
        let sum = parseInt(num1) + parseInt(num2)
        setResult(sum);              //update the result state variable
 };
 return (
  <div>
     <!–Other HTML Elements Goes Here-->
     <div>
       {result !== null && <span>Result: {result}</span>}
     </div>
  </div>
 );
}
export default App;
```

- result !== null ⟶ This checks if the variable sum is not equal to null.
  - If result is not null, the expression after && will be rendered.
  - If result is null, nothing is shown.

Refer this Video Demonstration:

https://youtu.be/oyh_GpR1qYU?si=GZ5cCbHnuFMNk8B4

Select Your Question Number = (Regno%5) + 1

Question 1: EMI Calculator App (React JS Application)

Develop a **React JS** application that calculates the **Equated Monthly Instalment (EMI)** for a loan. The app should allow users to input loan details and compute the EMI amount dynamically.

Requirements:

1. Use a **React functional component** with **useState** hooks to manage form inputs and results.
2. The app should allow users to enter:
   - **Loan Amount** (must be a positive number)
   - **Annual Interest Rate (%)** (must be a positive number)
   - **Loan Tenure (in months)** (must be a positive integer)
3. Include a button labelled **"Calculate EMI"**.
4. On button click:
   - Validate all input fields (generate alert message if the input is empty or negative values)
   - Calculate EMI using the formula:

$$EMI = \frac{P \times R \times (1+R)^N}{(1+R)^N - 1}$$

where

- P = Loan amount
- R = Monthly interest rate = Annual Rate / 12 / 100
- N = Loan tenure in months

5. Display the following details dynamically:
   - Loan Amount
   - EMI
   - Total Interest to be Paid

Question 2: Budget Calculator App (React JS Application)

Create a **Budget Calculator** using React JS to help users manage monthly income and expenses.

Requirements:

1. Use a **React functional component** with state variables for inputs and results.
2. The app should allow users to enter:
    - Monthly Income (must be positive)
    - Rent/EMI (must be positive)
    - Food Expenses (must be positive)
    - Transport Expenses (must be positive)
    - Other Expenses (must be positive)
3. Include a button labelled **"Calculate Balance"**.
4. On button click:
    - Validate all input fields (Generate alert message if the input is empty or negative)
    - Calculate Remaining Balance:

      Balance=Income−(Rent+Food+Transport+Others)
5. Display the calculated Remaining Balance below the button.
6. If the balance is **negative**, display it in **red** with a warning ("You are overspending!").
7. If the balance is **positive**, display it in **green** with a success message ("Good job managing your expenses!").

Question 3: BMI Calculator App (React JS Application)

Design a **BMI (Body Mass Index) Calculator** web application using React JS to help users track their fitness.

Requirements:

1. Use a **React functional component** with state variables for inputs and results.
2. Allow users to enter the following:
    - Height (in centimetres)
    - Weight (in kilograms)
3. Include a button labelled **"Calculate BMI"**.
4. On button click:
    - Validate all input fields (Generate alert message if the input is empty or negative)
    - Calculate BMI using the formula:

$$BMI = \frac{Weight}{(Height/100)^2}$$

5. Display the following results:
    - BMI value
    - BMI Status according to classification:
        - Underweight: BMI < 18.5
        - Normal weight: 18.5 ≤ BMI ≤ 24.9
        - Overweight: 25 ≤ BMI ≤ 29.9
        - Obese: BMI ≥ 30
6. Display BMI status with appropriate **color or message styling**.

Question 4: Loan Eligibility Checker App (React JS Application)

Develop a **Loan Eligibility Checker** using React JS to help customers check if they qualify for a loan.

Requirements:

1. Use a **React functional component** with state variables for inputs and results.
2. Allow users to input:
   - Name
   - Age
   - Monthly Salary
   - Existing EMI/Debts
   - Loan Amount Requested
3. Include a button labelled "**Check Loan Eligibility**".
4. Compute eligibility using the following rules on button click:-
   - **Debt-to-Income Ratio (DTI)** should not exceed 60%

$$DTI = \frac{ExistingEMI + ProposedEMI}{MonthlySalary} \times 100$$

   - Age should be between 21 and 60.
   - Requested loan amount ≤ 10 × Monthly Salary.
5. Display the result:
   - **Eligible:** Show in **green**
   - **Not Eligible:** Show in **red** with reasons.
6. Validate all user input (non-empty and realistic values).

Question 5: Calorie Tracker App (React JS Application)

Develop a React JS application that helps users track their daily calorie intake.

Requirements:

1. Use a **React functional component** with useState hooks to manage form inputs and results.
2. The app should allow users to enter:
    - Name
    - Daily Calorie Goal (must be a positive number)
    - Breakfast Calories (must be a positive number)
    - Lunch Calories (must be a positive number)
    - Dinner Calories (must be a positive number)
    - Snacks Calories (must be a positive number)
3. Include a button labelled **"Calculate Calories"**.
4. On button click:
    - Validate all input fields (show an alert if any field is empty or has negative values).
    - Calculate:
        - **Total Calories Consumed** = Breakfast + Lunch + Dinner + Snacks
        - **Remaining Calories** = Daily Calorie Goal – Total Calories Consumed
5. Display the following details dynamically:
    - Name
    - Daily Calorie Goal
    - Total Calories Consumed
    - Remaining Calories with:
        - **Red warning** if remaining < 0: "You exceeded your daily calorie goal!"
        - **Green message** if remaining ≥ 0: "You are within your daily goal!"