Full Stack – II

CT2 Suggestions:

1. Explain the concept of controlled components in React forms.

Ans: Controlled Components in React Forms

Definition:

Controlled components are form elements in React whose values are controlled by the component's state. In other words, the form data is handled by the React component rather than the DOM.

Key Points:

- The input field's value is bound to the component's state.
- Any change in input triggers an event handler to update the state.
- Ensures consistent and predictable form behavior.

Example:

```
function FormExample() {
  const [name, setName] = useState("");

  return (
    <div>
      <input
        type="text"
        value={name}
        onChange={(e) => setName(e.target.value)}
      />
      <p>Your name: {name}</p>
```

```
  </div>
 );
}
```

1. Describe how event handling works in React.

Ans: **Event Handling in React**

In **React**, event handling is similar to handling events in regular JavaScript but follows a few key differences. React uses **synthetic events**, which are wrappers around the browser's native events to ensure **cross-browser compatibility** and consistent behavior.

Key Points:

1. **CamelCase naming:** Event names use camelCase (e.g., onClick, onChange) instead of lowercase (onclick).
2. **Pass functions, not strings:** You pass a function as the event handler, not a string of code.
3. **Synthetic Events:** React's events work the same across all browsers.
4. **Automatic binding (for arrow functions):** Arrow functions help avoid manual binding of this.

Example:

```
import React from "react";

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

 return (
  <button onClick={handleClick}>Click Me</button>
 );
}

export default EventExample;
```

or,

```
function ClickButton() {
 function handleClick() {
  alert("Button Clicked!");
 }
```

return <button onClick={handleClick}>Click Me</button>;

}

1. Discuss the role of arrow functions in handling events.

Ans: Role of Arrow Functions in Handling Events

Definition:

Arrow functions in React are commonly used to handle events because they provide a concise syntax and automatically bind the function to the component's scope.

Key Points:

1. **Automatic binding:** Arrow functions inherit this from their surrounding scope, so no need to manually bind in the constructor.
2. **Passing arguments:** Arrow functions make it easy to pass values or event data to the handler.
3. **Clean syntax:** They make event handler code more concise.

Example:

```
import React from "react";

function ArrowFunctionExample() {
  const handleClick = (name) => {
    alert(`Hello, ${name}!`);
  };

  return (
    <button onClick={() => handleClick("John")}>
      Click Me
    </button>
  );
}
```

Or, function Greeting() {

const handleClick = (name) => {

```
  alert(Hello, ${name}!);
 };

 return <button onClick={() => handleClick("Pravas")}>Greet</button>;
}
```

1. Explain the significance of onChange in form elements.

Ans: Significance of onChange in Form Elements

Definition:

The onChange event in React is used to handle user input changes in form elements and update the component's state accordingly.

Key Points:

- State Synchronization: Keeps the input value and component state in sync.
- Controlled Components: Essential for handling form data dynamically.
- Instant Updates: Triggers whenever the user changes the input value.
- Validation Support: Enables real-time validation or instant feedback.

Example:

```
function FormExample() {
 const [email, setEmail] = useState("");

 return (
  <input
   type="email"
```

```
      value={email}
      onChange={(e) => setEmail(e.target.value)}
      placeholder="Enter your email"
    />
  );
}
```

Or, import React, { useState } from "react";

```
function OnChangeExample() {
  const [email, setEmail] = useState("");

  return (
    <div>
      <input
        type="email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
      />
      <p>Your email: {email}</p>
    </div>
  );
}
```

1. Illustrate how state updates enable real-time form validation.

Ans: How State Updates Enable Real-Time Form Validation

Definition:

In React, state updates help validate user input instantly by checking data as it changes and showing feedback in real time.

Key Points:

- Instant Feedback: Validates input as the user types.
- Error Handling: Stores validation errors in state.
- Live Updates: Re-renders UI based on current input.
- Better UX: Helps users correct errors quickly.

Example:

```
function Form() {
  const [name, setName] = useState("");
  const [error, setError] = useState("");

  const handleChange = (e) => {
    const value = e.target.value;
    setName(value);
    setError(value.length < 3 ? "Too short" : "");
  };

  return (
    <>
      <input value={name} onChange={handleChange} placeholder="Enter name" />
      <p>{error}</p>
    </>
  );
}
```

1. Discuss the purpose of the value attribute in form controls.

Ans: Purpose of the value Attribute in Form Controls

Definition:

In React, the value attribute defines the current value of a form control (like input, textarea, or select) and links it to the component's state, making the element controlled.

Key Points:

- State Binding: Connects the form control's value to the component's state.
- Controlled Input: Ensures React manages the data, not the DOM.
- Real-Time Updates: Reflects state changes instantly in the input field.
- Validation Support: Makes it easier to handle and validate form data.

Example:

```
function NameInput() {
  const [name, setName] = useState("");

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

Or, import React, { useState } from "react";

```
function ValueExample() {
  const [name, setName] = useState("John");

  return (
    <div>
      <input
        type="text"
        value={name}
        onChange={(e) => setName(e.target.value)}
      />
      <p>Hello, {name}!</p>
    </div>
  );
}
```

1. Summarize how to manage multiple input fields in a form.

Ans: Managing Multiple Input Fields in a Form

Definition:

In React, multiple input fields are managed using one state object and a common change handler that updates each field dynamically.

Key Points:

- Single State: Store all inputs in one state object.
- Dynamic Update: Identify fields using the name attribute.
- One Handler: Use a single onChange function for all inputs.

- Clean Code: Reduces repetition and improves readability.

Example:

```
function Form() {
  const [data, setData] = useState({ name: "", age: "" });

  const handleChange = (e) => {
    setData({ ...data, [e.target.name]: e.target.value });
  };

  return (
    <>
      <input name="name" value={data.name} onChange={handleChange} />
      <input name="age" value={data.age} onChange={handleChange} />
    </>
  );
}
```

1. Describe how handleSubmit is implemented in React forms.

Ans: Implementation of handleSubmit in React Forms

Definition:

In React, handleSubmit handles form submission by preventing the default page reload and processing form data using the component's state.

Key Points:

- Prevents Default: Stops browser's automatic reload.

- Custom Handling: Allows validation or API calls.

- Uses State: Submits data stored in state.

- Controlled Flow: Ensures predictable form behavior.

Example:

```
function Form() {
  const [email, setEmail] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    alert(Email: ${email});
  };

  return (
    <form onSubmit={handleSubmit}>
      <input value={email} onChange={(e) => setEmail(e.target.value)} />
      <button type="submit">Submit</button>
    </form>
  );
}
```

Or, import React, { useState } from "react";

```
function HandleSubmitExample() {
  const [name, setName] = useState("");

  const handleSubmit = (event) => {
    event.preventDefault(); // Prevent page reload
```

```
    alert(`Form submitted! Hello, ${name}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        value={name}
        onChange={(e) => setName(e.target.value)}
        placeholder="Enter your name"
      />
      <button type="submit">Submit</button>
    </form>
  );
}
```

1. Explain the purpose of data binding in React.

Ans: Purpose of Data Binding in React

Definition:

Data binding in React refers to the process of connecting the UI (view) with the component's data (state or props), ensuring that changes in data are reflected in the UI automatically.

Key Points:

- One-Way Binding: Data flows from component state/props to the UI.
- State Sync: Keeps UI updated when state changes.
- Event Handling: Updates state when user interacts with inputs.
- Predictable Flow: Ensures data moves in a single, controlled direction.

Example:

```
function DataBinding() {
  const [name, setName] = useState("Pravas");

  return (
    <>
      <input value={name} onChange={(e) => setName(e.target.value)} />
      <p>Hello, {name}!</p>
    </>
  );
}
```

Or, import React, { useState } from "react";

```
function DataBindingExample() {
  const [name, setName] = useState("John");

  return (
    <div>
      <input
        type="text"
        value={name}
        onChange={(e) => setName(e.target.value)} // two-way binding
      />
      <p>Hello, {name}!</p>
    </div>
  );
}
```

1. Describe how React fetches data from an API.

Ans: How React Fetches Data from an API

Definition:

React fetches data from APIs using the fetch() function or libraries like Axios, often inside the useEffect hook to load data when a component mounts.

Key Points:

- useEffect: Runs the API call after component renders.
- fetch(): Sends request and receives data.
- useState: Stores fetched data.
- UI Update: Renders new data automatically.

Example:

```
function Users() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/users")
      .then(res => res.json())
      .then(data => setUsers(data));
  }, []);

  return <p>{users[0]?.name}</p>;
}
```

1. Illustrate how useEffect() supports API calls in React.

Ans: How useEffect() Supports API Calls in React

Definition:

The useEffect() hook allows React components to perform API calls after rendering, making it ideal for fetching data from external sources.

Key Points:

- Runs After Render: Executes once when the component mounts.
- API Calls: Used to fetch data asynchronously.
- State Update: Stores and displays fetched data.
- Dependency Array: Prevents repeated calls.

Example:

```
function User() {
  const [name, setName] = useState("");

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/users/1")
      .then(res => res.json())
      .then(data => setName(data.name));
  }, []);

  return <p>{name}</p>;
}
```

1. Describe how to create a simple API using PHP and MySQL.

Ans: Creating a Simple API Using PHP and MySQL

Definition:

A simple PHP and MySQL API lets a client fetch or send data to a database and returns the result in JSON format.

Key Points:

- Connect Database: Link PHP to MySQL using mysqli.
- Run Query: Fetch data using SQL commands.
- JSON Output: Return data in JSON format.
- API Endpoint: PHP file works as an API URL.

Example:

```
<?php
$conn = new mysqli("localhost", "root", "", "testdb");

$result = $conn->query("SELECT * FROM users");
$data = $result->fetch_all(MYSQLI_ASSOC);

echo json_encode($data);
?>
```

1. Explain the function of JSON in API communication.

Ans: Function of JSON in API Communication

Definition:

JSON (JavaScript Object Notation) is a lightweight data format used in APIs to exchange data between a client and a server in a readable and structured way.

Key Points:

- Data Exchange Format: Transfers data between server and client.
- Readable & Lightweight: Easy for humans to read and machines to parse.
- Language Independent: Works with any programming language.
- Structured Data: Represents data as key–value pairs.

Example:

```
{
  "name": "Chipku",
  "email": "chipku@example.com"
}
```

Or, Example:

JSON Response from an API:

```
{
  "id": 1,
  "name": "Alice",
  "email": "alice@example.com"
}
```

React Fetch Example:

```
fetch("https://example.com/api/user/1")
  .then(response => response.json())
```

.then(data => console.log(data.name)); // Access JSON data

1. Summarize how MVC architecture is applied in AngularJS.

Ans: Application of MVC Architecture in AngularJS

Definition:

AngularJS follows the MVC (Model–View–Controller) architecture to organize application logic, data, and user interface efficiently.

Key Points:

- Model: Holds and manages the application data.
- View: Displays data to the user using HTML templates.
- Controller: Acts as a link between Model and View, handling logic and updates.
- Two-Way Binding: Automatically syncs data between Model and View.

Example:

```
<div ng-app="app" ng-controller="myCtrl">
  <input ng-model="name">
  <p>Hello, {{name}}</p>
</div>

<script>
angular.module("app", [])
.controller("myCtrl", function($scope) {
  $scope.name = "Pravas";
});
```

</script>

**Or, **

<h2>Hello {{name}}</h2>

<input type="text" ng-model="name">

** **

<script>

var app = angular.module("myApp", []);

app.controller("myCtrl", function($scope) {

$scope.name = "John"; // Model

});

</script>

1. Describe how event handling differs between React and AngularJS.

Ans: **Event Handling in React vs AngularJS (4 marks)**

Event handling in **React** and **AngularJS** serves the same purpose — responding to user actions — but the **approach and syntax** differ because of how each framework manages data binding and the DOM.

Key Differences:

Feature	**React**	**AngularJS**
Syntax	Uses **camelCase** event names like onClick, onChange.	Uses **Angular directives** like ng-click, ng-change.
Event Binding	Events are handled via **JSX attributes** and linked to component functions.	Events are handled through **HTML directives** connected to $scope functions.
Data Flow	**One-way binding** — state updates cause UI re-render.	**Two-way binding** — changes in the view update the model automatically.
Event Object	Uses **Synthetic Events**, a React wrapper for native events (cross-browser compatibility).	Uses the **$event** object from Angular's scope.

React Example:

```
function ReactExample() {
  const handleClick = () => alert("Button clicked in React!");
  return <button onClick={handleClick}>Click Me</button>;
}
```

AngularJS Example:

```
<div ng-app="myApp" ng-controller="myCtrl">
  <button ng-click="showAlert()">Click Me</button>
</div>

<script>
  var app = angular.module("myApp", []);
  app.controller("myCtrl", function($scope) {
    $scope.showAlert = function() {
      alert("Button clicked in AngularJS!");
    };
  });
</script>
```

Or, Difference in Event Handling Between React and AngularJS (4 Marks)

Definition:

React and AngularJS both handle user interactions, but React uses JSX event handlers, while AngularJS uses HTML directives like ng-click.

Key Points:

Syntax: React → onClick, AngularJS → ng-click.

Binding: React uses component functions; AngularJS uses $scope methods.

Data Flow: React has one-way binding; AngularJS has two-way binding.

Implementation: React handles events in JS, AngularJS in HTML templates.

Example:

React:

<button onClick={() => alert("React Clicked!")}>Click</button>

AngularJS:

<button ng-click="msg()">Click</button>

<script>

$scope.msg = function() { alert("AngularJS Clicked!"); }

</script>

1. Explain how React handles dynamic data updates.

Ans: How React Handles Dynamic Data Updates

Definition:

React handles dynamic data updates using its state and props system. When data in the state changes, React automatically re-renders the affected parts of the UI.

Key Points:

- State Management: Uses useState to store and update dynamic data.
- Re-rendering: React re-renders components when state or props change.
- Virtual DOM: Efficiently updates only changed elements, not the whole page.

- Real-Time Updates: Reflects data changes instantly in the UI.

Example:

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

  return (
    <>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Add</button>
    </>
  );
}
```

Or, Example:

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

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

  return (
    <div>
      <h3>Count: {count}</h3>
      <button onClick={() => setCount(count + 1)}>Increase</button>
    </div>
  );
}
```

1. Implement a React form that captures user name and email.

Ans: React Form to Capture User Name and Email

Definition:

React forms use state and event handling to capture user inputs like name and email dynamically.

Key Points:

- useState: Stores input values.
- onChange: Updates state on typing.
- onSubmit: Handles form submission.
- Controlled Inputs: Keeps UI and state in sync.

Example:

```
function UserForm() {
 const [name, setName] = useState("");
 const [email, setEmail] = useState("");

 const handleSubmit = (e) => {
  e.preventDefault();
  alert(Name: ${name}, Email: ${email});
 };

 return (
  <form onSubmit={handleSubmit}>
   <input value={name} onChange={(e) => setName(e.target.value)} placeholder="Name" />
   <input value={email} onChange={(e) => setEmail(e.target.value)} placeholder="Email" />
```

```
    <button type="submit">Submit</button>
   </form>
  );
}
```

Or, Example Code:

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

function UserForm() {
  const [formData, setFormData] = useState({ name: "", email: "" });

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    alert(`Name: ${formData.name}\nEmail: ${formData.email}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <h3>User Information Form</h3>

      <label>Name:</label>
      <input
        type="text"
        name="name"
        value={formData.name}
        onChange={handleChange}
```

```
        placeholder="Enter your name"
      />
      <br /><br />

      <label>Email:</label>
      <input
        type="email"
        name="email"
        value={formData.email}
        onChange={handleChange}
        placeholder="Enter your email"
      />
      <br /><br />

      <button type="submit">Submit</button>
    </form>
  );
}

export default UserForm;
```

1. Demonstrate the use of arrow functions to handle a button click event in React.

Ans: Using Arrow Functions to Handle Button Click in React

Definition:

Arrow functions in React are used to handle events like button clicks concisely and automatically bind the function to the component's scope.

Key Points:

- Automatic Binding: No need for .bind(this) in functions.

- Short Syntax: Cleaner and easier to write.

- Parameter Support: Can easily pass arguments to event handlers.

- Inline Use: Can be used directly in JSX.

Example:

```
function ClickButton() {
  const handleClick = () => {
    alert("Button Clicked!");
  };

  return <button onClick={() => handleClick()}>Click Me</button>;
}
```

Or, Using Arrow Functions to Handle a Button Click Event in React

Arrow functions are commonly used in React to handle events like button clicks because they provide a clean syntax and automatically bind this, avoiding the need for manual binding.

Example Code:

```
import React from "react";

function ClickExample() {
  const handleClick = () => {
    alert("Button clicked using an arrow function!");
  };

  return (
    <div>
      <h3>Arrow Function Example</h3>
```

```
    <button onClick={handleClick}>Click Me</button>
  </div>
 );
}

export default ClickExample;
```

1. Use fetch API in React to retrieve user data from a JSON API.

Ans: Using Fetch API in React to Retrieve User Data

Definition:

React uses the Fetch API inside the useEffect() hook to retrieve data from external JSON APIs and display it dynamically.

Key Points:

- useEffect: Runs the fetch call after component renders.
- fetch(): Sends request to a JSON API.
- useState: Stores fetched data.
- Dynamic Rendering: Updates UI once data is received.

Example:

```
function UserData() {
 const [user, setUser] = useState({});

 useEffect(() => {
  fetch("https://jsonplaceholder.typicode.com/users/1")
```

```
    .then(res => res.json())
    .then(data => setUser(data));
  }, []);

  return <p>{user.name}</p>;
}
```

Or, Using Fetch API in React to Retrieve User Data (4 marks)

Example:

```
import React, { useEffect, useState } from "react";

function FetchUserData() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/users")
      .then(res => res.json())
      .then(data => setUsers(data));
  }, []);

  return (
    <div>
      <h3>User Data</h3>
      {users.slice(0, 3).map(user => (
        <p key={user.id}>{user.name} - {user.email}</p>
      ))}
    </div>
  );
}

export default FetchUserData;
```

1. Write a React code snippet to send form data to a server using fetch POST.

Ans: Sending Form Data to Server Using Fetch POST

Definition:

React uses the fetch() method with the POST request to send form data to a server.

Key Points:

- useState: Stores user input.
- onSubmit: Handles form submission.
- fetch(): Sends data as JSON.
- POST: Used to add data to the server.

Example:

```
function PostForm() {
  const [name, setName] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    fetch("https://example.com/api", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ name }),
    });
  };
```

```
  return (
    <form onSubmit={handleSubmit}>
      <input value={name} onChange={(e) => setName(e.target.value)} placeholder="Name" />
      <button type="submit">Submit</button>
    </form>
  );
}
```

Or, Sending Form Data to Server Using Fetch POST

Example:

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

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

  const handleSubmit = (e) => {
    e.preventDefault();
    fetch("https://example.com/api/users", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(user),
    })
      .then((res) => res.json())
      .then(() => alert("Data sent successfully!"));
  };

  return (
    <form onSubmit={handleSubmit}>
      <input name="name" onChange={(e) => setUser({ ...user, name: e.target.value })} placeholder="Name" />
      <input name="email" onChange={(e) => setUser({ ...user, email: e.target.value })} placeholder="Email" />
```

```
    <button type="submit">Send</button>
  </form>
 );
}

export default SendData;
```

1. Implement a dropdown form in React and store selected value in state.

Ans: Dropdown Form in React

Definition:

A dropdown in React lets users pick an option, and the selected value is stored in state.

Key Points:

- useState: Holds selected value.
- onChange: Updates state on selection.
- Controlled Input: Keeps value in sync.

Example:

```
function Dropdown() {
 const [city, setCity] = useState("");

 return (
  <>
   <select value={city} onChange={(e) => setCity(e.target.value)}>
    <option value="">Select City</option>
```

```
        <option value="Delhi">Delhi</option>
        <option value="Mumbai">Mumbai</option>
      </select>
      <p>City: {city}</p>
    </>
  );
}
```

Or,

Dropdown Form in React

Example:

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

function DropdownExample() {
  const [city, setCity] = useState("");

  return (
    <div>
      <select value={city} onChange={(e) => setCity(e.target.value)}>
        <option value="">Select City</option>
        <option value="Delhi">Delhi</option>
        <option value="London">London</option>
        <option value="Paris">Paris</option>
      </select>
      <p>Selected: {city}</p>
    </div>
  );
}

export default DropdownExample;
```

1. Apply basic validation logic to a React form.

Ans: Basic Form Validation in React

Definition:

React uses simple validation logic to ensure correct user input before form submission.

Key Points:

- useState: Stores input and error.
- Check Condition: Validates input.
- Show Error: Displays message if invalid.

Example:

```
function SimpleForm() {
 const [name, setName] = useState("");
 const [error, setError] = useState("");

 const handleSubmit = (e) => {
  e.preventDefault();
  name === "" ? setError("Name is required!") : setError("");
 };

 return (
  <form onSubmit={handleSubmit}>
   <input value={name} onChange={(e) => setName(e.target.value)} placeholder="Name" />
   <button type="submit">Submit</button>
```

```
      <p style={{ color: "red" }}>{error}</p>
    </form>
  );
}
```

Or, Basic Form Validation in React

Example:

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

function FormValidation() {
  const [name, setName] = useState("");
  const [error, setError] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    name === "" ? setError("Name is required") : alert("Form submitted!");
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="Enter Name"
        value={name}
        onChange={(e) => setName(e.target.value)}
      />
      <button type="submit">Submit</button>
      <p style={{ color: "red" }}>{error}</p>
    </form>
  );
}
```

export default FormValidation;

1. Use a button event to toggle a message display in React.

Ans: Toggling Message Display with Button in React

Definition:

React can toggle message visibility using a button click that updates the state to show or hide content.

Key Points:

- useState: Stores visibility status.
- onClick: Toggles the state value.
- Conditional Rendering: Displays message based on state.

Example:

```
function ToggleMessage() {
  const [show, setShow] = useState(false);

  return (
    <>
      <button onClick={() => setShow(!show)}>Toggle Message</button>
      {show && <p>Hello, Welcome to React!</p>}
    </>
  );
}
```

Or, Toggle Message Display Using Button in React

Example:

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


function ToggleMessage() {
  const [show, setShow] = useState(false);


  return (
    <div>
      <button onClick={() => setShow(!show)}>
        {show ? "Hide Message" : "Show Message"}
      </button>
      {show && <p>Hello, welcome to React!</p>}
    </div>
  );
}


export default ToggleMessage;
```

1. lop a contact form that uses local state and alerts the submitted values.

Ans: Contact Form with Local State and Alert in React

Definition:

A contact form in React uses local state to store user input and shows the submitted values using an alert on form submission.

Key Points:

- useState: Stores name and email.
- onChange: Updates input fields.

- onSubmit: Triggers alert with entered data.

- Controlled Form: Keeps input synced with state.

Example:

```
function ContactForm() {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    alert(Name: ${name}, Email: ${email});
  };

  return (
    <form onSubmit={handleSubmit}>
      <input value={name} onChange={(e) => setName(e.target.value)} placeholder="Name" />
      <input value={email} onChange={(e) => setEmail(e.target.value)} placeholder="Email" />
      <button type="submit">Submit</button>
    </form>
  );
}
```

Or, Contact Form with Local State and Alert

Example:

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

function ContactForm() {
  const [data, setData] = useState({ name: "", email: "" });
```

```
  const handleSubmit = (e) => {
    e.preventDefault();
    alert(`Name: ${data.name}\nEmail: ${data.email}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        name="name"
        placeholder="Name"
        onChange={(e) => setData({ ...data, name: e.target.value })}
      />
      <input
        name="email"
        placeholder="Email"
        onChange={(e) => setData({ ...data, email: e.target.value })}
      />
      <button type="submit">Submit</button>
    </form>
  );
}

export default ContactForm;
```

1. Justify the use of state in managing form inputs in React.

Ans: **Use of State in Managing Form Inputs in React**

Answer: In React, **state** is used to store and manage the values of form inputs, making them **controlled components**. This allows React to handle user input dynamically and ensures the UI always reflects the latest data.

Key Points:

1. **Real-time updates:** As the user types, state updates instantly and reflects in the input fields.
2. **Validation:** State makes it easy to validate input values before submission.
3. **Single source of truth:** The input values are stored in the component's state instead of the DOM.

4. **Form submission:** Collected state data can be easily sent to APIs or processed.

Example:

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

<input value={name} onChange={(e) => setName(e.target.value)} />
```

Or, Use of State in Managing Form Inputs in React

Definition:

In React, state is used to store and update form input values, ensuring the UI always reflects the current data entered by the user.

Key Points:

Data Control: Keeps input values synchronized with the component.

Real-time Updates: Reflects user input instantly in the UI.

Validation Support: Helps perform live input validation.

Dynamic Behavior: Enables conditional rendering based on input values.

Example:

```
function FormInput() {
 const [name, setName] = useState("");
 return (
  <>
   <input value={name} onChange={(e) => setName(e.target.value)} placeholder="Enter Name" />
   <p>Hello, {name}</p>
  </>
 );
```

}

1. Evaluate the performance of controlled vs uncontrolled components in form handling.

Ans: Controlled vs Uncontrolled Components

Definition:

Controlled components use React state to manage inputs, while uncontrolled components rely on the DOM.

Key Points:

Controlled: More control, supports validation, may re-render often.

Uncontrolled: Faster, uses ref, less flexible.

Controlled = Accuracy, Uncontrolled = Speed.

Example:

<input value={name} onChange={(e) => setName(e.target.value)} /> // Controlled

<input ref={inputRef} /> // Uncontrolled

Or, Controlled vs Uncontrolled Components in React

Answer:Controlled and uncontrolled components differ in how they manage form data, affecting performance and control.

Aspect	Controlled Components	Uncontrolled Components
Data Handling	**Form data is managed by React state.**	**Form data is managed by the DOM.**
Performance	**Slightly slower for large forms (due to frequent re-renders).**	**Faster since React doesn't re-render on every keystroke.**
Validation & Control	**Easier to validate and control inputs dynamically.**	**Harder to validate, requires manual DOM access (ref).**
Use Case	**Preferred for small to medium forms with logic/validation.**	**Useful for simple or large forms where performance is critical.**

Example (Controlled):

<input value={name} onChange={(e) => setName(e.target.value)} />

Example (Uncontrolled):

<input ref={inputRef} />

1. Evaluate how data binding in React differs from AngularJS.

Ans: Data Binding in React vs AngularJS

Definition:

Data binding connects the UI and data model. React and AngularJS handle this process differently.

Key Points:

React: Uses one-way binding — data flows from component state to UI only.

AngularJS: Uses two-way binding — UI and data model update each other automatically.

React: Requires explicit event handlers (onChange) to update state.

AngularJS: Uses directives like ng-model for automatic sync.

Example:

// React (one-way)

<input value={name} onChange={(e) => setName(e.target.value)} />

// AngularJS (two-way)

<input ng-model="name">

Or, Data Binding in React vs AngularJS

Answer:React and AngularJS differ mainly in how they connect data between the UI and application logic.

Feature	**React**	**AngularJS**
Type of Binding	**One-way data binding**	**Two-way data binding**
Data Flow	**Data flows from parent → child (unidirectional)**	**Data flows both ways (model ↔ view)**
Performance	**Faster and more predictable**	**Can be slower with many bindings**
Implementation	**Uses state and props**	**Uses ng-model directive**

Example (React):

<input value={name} onChange={(e) => setName(e.target.value)} />

Example (AngularJS):

<input ng-model="name">

Got it ✅ — you want **each answer** to stay in **long, 5-mark form**, fully descriptive, easy to memorize, written in simple language (not short summaries).
Below is your complete set, rewritten to match the "exam-ready long answer" style while keeping everything clear and readable.

Q1. Discuss the concept of reusable components in React.
Ans:

1. **Meaning**: Reusable components in React are independent UI blocks that can be used multiple times in different parts of an application. They allow developers to write once and use anywhere.
2. **Purpose**: The main goal is to reduce code repetition and make the codebase more organized and modular.

3. **Example**: A button, input field, or card component can be designed once and reused in various pages by changing only its props.
4. **Benefits**: Improves consistency, reduces development time, and makes debugging easier.
5. **Conclusion**: Reusable components are the foundation of React's modular design, helping in building scalable and maintainable web applications.

Q2. Describe how to pass props from a parent to a child component.
Ans:

1. **Definition**: Props (short for "properties") are used to pass data from one component (parent) to another (child).
2. **How it works**: In the parent, data is sent as an attribute like <Child name="Subham" />.
3. **Receiving in child**: The child component receives the data through the props object, accessed as props.name.
4. **Unidirectional flow**: Data always flows from parent to child, not the other way.
5. **Example**:
6. function Child(props){ return <h3>Hello, {props.name}</h3>; }
7. This allows flexible and reusable communication between components.

Q3. Explain the significance of useState in functional components.
Ans:

1. **Purpose**: useState is a React Hook that lets functional components manage and store state values.
2. **Syntax**: const [count, setCount] = useState(0); initializes a state variable count and updates it with setCount.
3. **Dynamic behavior**: It enables re-rendering of the UI whenever the state changes.
4. **Use cases**: Used in counters, input fields, forms, and toggles to track changing data.
5. **Conclusion**: useState makes functional components interactive and powerful, replacing the need for class-based components.

Q4. Clarify the difference between state and props in React.
Ans:

Aspect	**State**	**Props**
Definition	Internal data managed inside a component.	External data received from parent component.
Mutability	Mutable – can change using setState or useState.	Immutable – cannot be changed by the child.
Usage	Used for managing dynamic UI behavior.	Used for configuration and data transfer.
Ownership	Owned and controlled by the same component.	Controlled by the parent component.
Example	const [count, setCount] = useState(0)	<Child name="Subham" />

Q5. Discuss the use of prop validation in React.
Ans:

1. **Purpose**: Prop validation ensures that the correct type of data is passed to components, preventing runtime errors.
2. **Library used**: The prop-types package helps define the expected type of each prop.
3. **Syntax**:
4. Component.propTypes = { name: PropTypes.string.isRequired };
5. **Advantages**: Detects bugs early, improves maintainability, and clarifies component requirements.
6. **Conclusion**: Prop validation enforces data integrity and improves code reliability across large applications.

Q6. Describe the rules for using state in React.
Ans:

1. **Declaration**: State in functional components must be declared using the useState hook.
2. **Update rule**: State must never be modified directly; always use the state updater function.
3. **Component scope**: Each component maintains its own state independently.
4. **Initialization**: Provide a default value when initializing, e.g., useState("").
5. **Effect on rendering**: Any state change causes automatic re-rendering of that specific component.

Q7. Analyze how props are used to pass data in React components.
Ans:

1. **Data transfer**: Props pass data from parent to child in a top-down (unidirectional) flow.
2. **Customizability**: They allow child components to behave differently based on data received.
3. **Structure**: Props can hold values, arrays, functions, or even other components.
4. **Example**:
5. <UserCard name="Subham" age={20} />
6. Child can access it as props.name or {name} using destructuring.
7. **Outcome**: Props enable flexible, reusable, and dynamic React component communication.

Q8. Examine how React Router enhances single-page applications.
Ans:

1. **Definition**: React Router allows navigation between components without reloading the entire webpage.
2. **Components used**: <BrowserRouter>, <Routes>, and <Route> handle routing.
3. **Benefit**: Improves speed and performance by updating only parts of the page.
4. **Example**:
5. <Route path="/home" element={<Home />} />
6. **Result**: Provides seamless transitions and better user experience in Single Page Applications (SPAs).

Q9. Explain the concept of controlled components in React forms.
Ans:

1. **Definition**: Controlled components are form elements where React state controls the value.
2. **Mechanism**: The value attribute is bound to the component's state.
3. **onChange handler**: Updates the state every time the user types or selects input.
4. **Example**:
5. <input value={name} onChange={(e)=>setName(e.target.value)} />
6. **Benefit**: Enables live validation, dynamic updates, and better control over user input.

Q10. Describe how event handling works in React.
Ans:

1. **Concept**: React uses synthetic events, which are wrappers around browser events for better performance.
2. **Syntax**: Events are written in camelCase, like onClick instead of onclick.
3. **Function binding**: Handlers are passed as functions, e.g., <button onClick={handleClick}>.
4. **Automatic binding**: Arrow functions help avoid manual binding of this.

5. **Benefit**: Offers consistent event handling across different browsers with virtual DOM optimization.

Q11. Discuss the role of arrow functions in handling events.
Ans:

1. **Definition**: Arrow functions simplify event handling and maintain the correct this context.
2. **Usage**: Commonly used for inline event handlers like
3. <button onClick={() => handleClick()}>Click</button>
4. **Advantage**: No need for .bind(this) when using class components.
5. **Cleaner syntax**: Makes event handler code shorter and more readable.
6. **Outcome**: Ensures predictable behavior and fewer context errors in large projects.

Q12. Explain the significance of onChange in form elements.
Ans:

1. **Purpose**: onChange detects any modification in input fields.
2. **Functionality**: Updates the component's state with the new input value.
3. **Example**:
4. <input onChange={(e)=>setName(e.target.value)} />
5. **Result**: Keeps the component and UI synchronized.
6. **Use case**: Essential for building real-time and interactive form validation systems.

Q13. Illustrate how state updates enable real-time form validation.
Ans:

1. **Process**: Every input change updates the component state using useState.
2. **Validation logic**: Validation functions run on each update to check user input.
3. **Real-time feedback**: Errors are shown instantly while typing.
4. **Example**:
5. setError(name === "" ? "Name required" : "");
6. **Result**: Enhances user experience and ensures correct data submission.

Q14. Discuss the purpose of the value attribute in form controls.
Ans:

1. **Definition**: The value attribute binds the form control directly to the React state.
2. **Control**: Keeps the displayed value in sync with the underlying state.
3. **Usage**:
4. <input value={email} onChange={handleChange} />
5. **Behavior**: Without it, the input becomes uncontrolled by React.
6. **Importance**: Provides complete control for managing data and validation.

Q15. Summarize how to manage multiple input fields in a form.
Ans:

1. **Approach**: Use a single state object to manage all form fields.
2. **Dynamic updates**: Use the name attribute to identify which field changed.
3. **Example**:
4. const handleChange = (e) => setForm({...form, [e.target.name]: e.target.value});
5. **Advantage**: Simplifies code and avoids creating separate states for each input.
6. **Result**: Keeps forms clean, efficient, and scalable for complex UIs.

Q16. Describe how handleSubmit is implemented in React forms.
Ans:

1. **Definition**: The handleSubmit function manages form submission without reloading the page.
2. **Syntax**:
3. const handleSubmit = (e) => { e.preventDefault(); console.log(formData); };
4. **Functionality**: Collects form data, validates it, and sends it to the backend.
5. **Benefit**: Prevents page refresh and gives full control over submission.

6. **Result**: Enables smooth and user-friendly data submission workflows.

Q17. Explain the purpose of data binding in React.
Ans:

1. **Definition**: Data binding is the connection between the UI elements and component state or props.
2. **Type**: React supports one-way binding (state → UI) and two-way binding via value + onChange.
3. **Example**:
4. <input value={name} onChange={(e)=>setName(e.target.value)} />
5. **Purpose**: Keeps the UI synchronized with application data.
6. **Advantage**: Ensures predictable, consistent, and maintainable UIs.

Q18. Describe how React fetches data from an API.
Ans:

1. **Method**: React uses JavaScript functions like fetch() or libraries like axios to request data.
2. **Syntax**:
3. fetch("https://api.example.com/data").then(res => res.json()).then(setData);
4. **Asynchronous**: Uses async/await to handle asynchronous data.
5. **State update**: Stores the fetched data using useState for rendering.
6. **Outcome**: Dynamically displays live data from external sources.

Q19. Illustrate how useEffect() supports API calls in React.
Ans:

1. **Purpose**: The useEffect hook performs side effects like fetching data or updating the DOM.
2. **Usage**:
3. useEffect(()=>{ fetchData(); }, []);
4. **Dependency array**: Controls when the effect runs (empty array = run once).
5. **Advantage**: Prevents repeated API calls during every render.
6. **Outcome**: Makes data loading smooth and efficient when the component mounts.

Q20. Describe how to create a simple API using PHP and MySQL.
Ans:

1. **Database setup**: Create a MySQL table (e.g., users) using SQL queries.
2. **Connection**: Use PHP's mysqli_connect() or PDO to connect to the database.
3. **Query execution**: Fetch or insert data using SELECT, INSERT, or UPDATE.
4. **Output as JSON**: Convert result to JSON using json_encode().
5. **Example**:
6. echo json_encode(["status"=>"success","message"=>"Data fetched"]);

Q21. Explain the function of JSON in API communication.
Ans:

1. **Meaning**: JSON (JavaScript Object Notation) is a lightweight format for data exchange.
2. **Structure**: Represents data as key-value pairs, e.g., { "name": "Subham", "age": 21 }.
3. **Use in APIs**: Enables communication between frontend (React) and backend (PHP, Node.js).
4. **Benefits**: Easy to read, language-independent, and supports nested data.
5. **Conclusion**: JSON is the universal format for modern web applications to exchange structured data efficiently.