DAY 3 – Forms, Controlled Inputs, Lists, Keys, CRUD Basics & To-Do App (FULL DAY NOTES)

---

1. What Are Forms in React?

Simple Definition

Forms allow users to enter data:

- Text inputs
- Email fields
- Numbers
- File uploads
- Textarea
- Buttons

Example in HTML

```
<input type="text" placeholder="Enter name" />
```

In React, we control this input using **state**. This is called a **Controlled Component**.

---

2. Controlled Inputs (Very Important for React Apps)

Why controlled inputs?

Because React should **control** the value of input — not the DOM.

Basic Controlled Input Example

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

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

How it works:

1. `value={name}` → input value comes from React state
2. `onChange` → updates the state when user types
3. UI refreshes automatically

---

3. onChange vs onClick vs onSubmit

onChange – input typing

onClick – button press

onSubmit – whole form submission

Example:

```
<form onSubmit={handleSubmit}>
  ...
</form>
```

---

4. Handling Form Submission

React does NOT reload page like HTML forms. We prevent reload using:

```
event.preventDefault();
```

Full Example:

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

  function handleSubmit(e) {
    e.preventDefault();
    alert("Submitted: " + email);
  }

  return (
    <form onSubmit={handleSubmit}>
      <input
```

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

5. Multi-Input Form (Name + Email + Password)

```
const [form, setForm] = useState({
  name: "",
  email: "",
  password: ""
});

function handleChange(e) {
  setForm({
    ...form,
    [e.target.name]: e.target.value
  });
}
```

Inputs:

```
<input name="name" onChange={handleChange} />
<input name="email" onChange={handleChange} />
<input name="password" onChange={handleChange} />
```

Why this method?

✓ Scalable ✓ Cleaner code ✓ Works for large forms

6. Lists in React (Very Important)

Used for tasks, users, products, notes, etc.

Basic example:

```
const animals = ["Dog", "Cat", "Cow"];
```

```
animals.map(a => <p>{a}</p>);
```

---

7. Keys in Lists (Must Understand)

Keys help React track items during:

- Add
- Remove
- Update

Example:

```
animals.map((a, index) => <p key={index}>{a}</p>);
```

Better keys?

Use IDs if available.

```
{users.map(user => (
  <p key={user.id}>{user.name}</p>
))}
```

---

8. CRUD Basics (Used in To-Do App)

CRUD =

- **C**reate (Add)
- **R**ead (List)
- **U**pdate (Edit)
- **D**elete (Remove)

We will build all 4 in the To-Do App.

---

9. To-Do List App (Full Project Explanation)

We will learn: ✓ State ✓ Controlled inputs ✓ Array methods ✓ Adding items ✓ Deleting items ✓ Rendering lists with keys ✓ Styling

---

Step 1: Setup State for Input + Tasks

```
const [task, setTask] = useState("");
const [tasks, setTasks] = useState([]);
```

Step 2: Add a New Task

Function:

```
function addTask() {
  if (task.trim() === "") return;

  setTasks([...tasks, task]);
  setTask("");
}
```

Button:

```
<button onClick={addTask}>Add</button>
```

Step 3: Controlled Input

```
<input
  type="text"
  value={task}
  onChange={(e) => setTask(e.target.value)}
  placeholder="Enter a task"
/>
```

Step 4: Displaying Tasks (mapping)

```
{tasks.map((t, index) => (
  <div key={index}>{t}</div>
))}
```

Step 5: Delete a Task

```
function deleteTask(index) {
  setTasks(tasks.filter((_, i) => i !== index));
}
```

UI:

```
<button onClick={() => deleteTask(index)}>Delete</button>
```

Step 6: Final To-Do Component

```
function TodoApp() {
  const [task, setTask] = useState("");
  const [tasks, setTasks] = useState([]);

  function addTask() {
    if (!task.trim()) return;
    setTasks([...tasks, task]);
    setTask("");
  }

  function deleteTask(index) {
    setTasks(tasks.filter((_, i) => i !== index));
  }

  return (
    <div className="todo-container">
      <h1>To-Do App</h1>

      <input
        value={task}
        onChange={(e) => setTask(e.target.value)}
        placeholder="New task..."
      />

      <button onClick={addTask}>Add</button>

      {tasks.map((t, index) => (
        <div key={index} className="task">
          <span>{t}</span>
          <button onClick={() => deleteTask(index)}>Delete</button>
        </div>
        ))}
      </div>
    );
  }
```

10. Styling Suggestions for Students

```css
.todo-container {
  width: 300px;
  margin: auto;
  padding: 10px;
  background: #f5f5f5;
  border-radius: 10px;
}

.task {
  display: flex;
  justify-content: space-between;
  background: #fff;
  padding: 8px;
  margin-top: 5px;
  border-radius: 5px;
}
```

11. What Students Must Understand on Day 3

By the end of Day 3, students must clearly know:

✓ How to create controlled inputs

✓ How forms work in React

✓ How to handle events

✓ How to store user input in state

✓ How to add data into an array

✓ How to render lists using `.map()`

✓ Why keys are important

✓ How to delete items

✓ Build a complete To-Do App

---

12. Advanced Concepts (Light Introduction Only)

Two-way Binding

```
Input updates state → state updates UI
```

Immutability

Never modify array directly:

❌ Wrong:

```
tasks.push("new");
```

✓ Correct:

```
setTasks([...tasks, "new"]);
```
