Internship Report – Frontend Dev

Week 6: React.js Basics

Name: Zainab
Father Name: Assad Qayyum
Date: 30th July, 2025
Internship Domain: Front-end Intern
Task: Lists with .map(), Forms in React, Lifting State Up

Task Overview: (Day3)

Today's task focused on understanding and practicing React.js basics, specifically how to **display lists** dynamically, **handle forms** using controlled components, and manage **shared state between components** using the concept of lifting state up.

Content Covered:

The content covered today is:

- Lists with .map()
- Forms in React
- Lifting State Up

1. Lists with .map()

A list means showing multiple similar items (like products, tasks, names). In React, we often have these items stored in an array. The **.map() method** helps us **loop through the array** and **display each item** on the screen.

Rendering Arrays with .map()

```
const fruits = ["Apple", "Banana", "Mango"];

function FruitList() {
  return (
    <ul>
      {fruits.map((fruit) => <li>{fruit}</li>)}
    </ul>
  );
}
```

- **fruits.map(...)** → goes through each item (Apple → Banana → Mango).
- **
{fruit}
** → creates a list element for each fruit.

Why is "key" important?

React uses keys to know which item changed, added, or removed in a list.

Without keys, React re-renders everything, which can cause bugs.

Add a unique key (like an ID or index):

```
{fruits.map((fruit, index) => <li key={index}>{fruit}</li>)}
```

2. Forms in React

A form lets users **type or select information** (like login forms or search boxes). In React, we usually make forms controlled, meaning the form values live in state.

Controlled Components:

A controlled input means its value is managed by React's state.

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

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

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

- **value={name}** → input shows what's in name.
- **onChange** → updates state when user types.

Handling Different Inputs

Text / Textarea: use onChange same as above.

Checkbox: use e.target.checked instead of value.

Dropdown (select): use e.target.value

For Example, for a checkbox:

```
<input
  type="checkbox"
  checked={isChecked}
  onChange={(e) => setIsChecked(e.target.checked)}
/>
```

Handling Submit:

To handle when the user clicks Submit, use onSubmit. Prevent default refresh with event.preventDefault().

```
<form onSubmit={(e) => { e.preventDefault(); alert(name); }}>
  <button type="submit">Submit</button>
</form>
```

3. Lifting State Up

When **two components need the same data** (for example, a slider and a display),You move the **state to their parent** so both can share and update it.

The Problem:

- If each component keeps its own state → data becomes unsynchronized.
- Example: typing in one input but the other component doesn't update.

The Solution:

- Lift state up → create one state in the parent component,
- Pass it down as props to children.

Example:

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

function Child1({ message }) {
  return <h3>Child 1 says: {message}</h3>;
}
function Child2({ message }) {
  return <h3>Child 2 also says: {message}</h3>;
}
function Parent() {
  const [msg, setMsg] = useState("Hello");

  return (
    <div>
      <input
        type="text"
        value={msg}
        onChange={(e) => setMsg(e.target.value)}
      />
      <Child1 message={msg} />
      <Child2 message={msg} />
    </div>
  );
}
```

Passing State as Props:

This means sending data stored in a parent component's state down to a child component so the child can use and display that data.

Passing Functions (Callbacks):

This means sending a function from the parent component to a child component so the child can call it and send information or trigger changes back in the parent's state.

Practice Code:

App.js

src > App.js > AddItemForm

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

function AddItemForm({ onAdd }) {
  const [inputValue, setInputValue] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    if (inputValue.trim() === "") return;
    onAdd(inputValue);
    setInputValue("");
  };

  return (
    <form onSubmit={handleSubmit} style={styles.form}>
      <input
        type="text"
        value={inputValue}
        onChange={(e) => setInputValue(e.target.value)}
        placeholder="Enter item"
        style={styles.input}
      />
      <button type="submit" style={styles.button}>Add</button>
    </form>
  );
}
function ItemList({ items, onDelete }) {
  return (
    <ul style={styles.list}>
      {items.map((item, index) => (
        <li key={index} style={styles.listItem}>
```

Ln 18, Col 56 Spaces: 2 UTF-8 LF { } JavaScript Go Live

App.js

src > App.js > AddItemForm

```
function ItemList({ items, onDelete }) {
  return (
    <ul style={styles.list}>
      {items.map((item, index) => (
        <li key={index} style={styles.listItem}>
          🛒 {item}
          <button
            onClick={() => onDelete(index)}
            style={styles.deleteButton}
          >
            delete
          </button>
        </li>
      ))}
    </ul>
  );
}

export default function App() {
  const [items, setItems] = useState([]);

  const addItem = (newItem) => {
    setItems([...items, newItem]);
  };

  const deleteItem = (indexToRemove) => {
    const updatedItems = items.filter((_, index) => index !== indexToRemove);
    setItems(updatedItems);
  };
```

Ln 18, Col 56 Spaces: 2 UTF-8 LF { } JavaScript Go Live

App.js

src > App.js > App

```
export default function App() {
  return (
    <div style={styles.container}>
      <h1 style={styles.heading}>📝 Shopping List</h1>
      <AddItemForm onAdd={addItem} />
      <ItemList items={items} onDelete={deleteItem} />
    </div>
  );
}
const styles = {
  container: {
    maxWidth: "400px",
    margin: "40px auto",
    textAlign: "center",
    padding: "20px",
    border: "1px solid #ccc",
    borderRadius: "10px",
    backgroundColor: "#f9f9f9"
  },
  heading: { marginBottom: "20px", color: "#333" },
  form: { marginBottom: "20px" },
  input: {
    padding: "10px",
    width: "65%",
    marginRight: "10px",
    border: "1px solid #ccc",
    borderRadius: "5px"
  },
  button: {
```

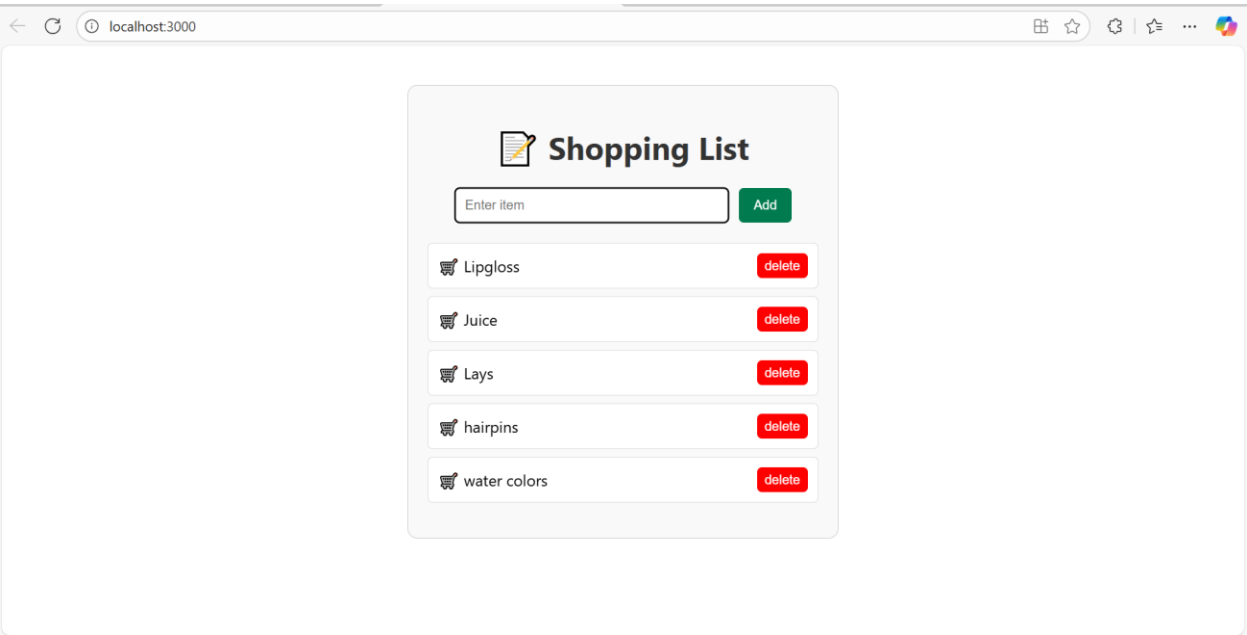
Ln 63, Col 2 Spaces: 2 UTF-8 LF {} JavaScript Go Live

App.js

src > App.js > styles

```
const styles = {
  button: {
    padding: "10px 15px",
    backgroundColor: "#007B4F",
    color: "#fff",
    border: "none",
    borderRadius: "5px",
    cursor: "pointer"
  },
  list: { listStyle: "none", padding: 0 },
  listItem: {
    backgroundColor: "#fff",
    marginBottom: "8px",
    padding: "10px",
    border: "1px solid #ddd",
    borderRadius: "5px",
    textAlign: "left",
    display: "flex",
    justifyContent: "space-between",
    alignItems: "center"
  },
  deleteButton: {
    backgroundColor: "red",
    color: "white",
    border: "none",
    padding: "5px 8px",
    borderRadius: "5px",
    cursor: "pointer"
  }
};
```

Ln 83, Col 3 Spaces: 2 UTF-8 LF {} JavaScript Go Live



Conclusion:

In today's task, I learned how to create dynamic lists, handle user inputs through forms, and effectively share data between components in React. These concepts form the foundation for building interactive and state-managed applications.