Internship Report – Frontend Dev

Week 6: React.js Basics

Name: Zainab
Father Name: Assad Qayyum
Date: 1st Aug, 2025
Internship Domain: Front-end Intern
Task: Mini-Project: User Info Collector Form

Task Overview: (Day5)

Today's task was to build a **Mini Project in React.js** that incorporates all the concepts learned during Week 6 of the internship. The project chosen was **"User Info Collector App",** where users can enter their details (name, email, gender), add multiple users to a list, and manage the list dynamically.

Objective:

The objective was to:

- Practice building a React application using functional components.
- Manage and update data dynamically using React state and event handling.
- Implement conditional rendering, lists with .map().
- Use the useEffect hook to handle side effects.

Mini-Project: User Info Collector

This project is a simple form based React application that **collects user information (name, email, and gender)**, stores it in a dynamic list, and displays all added users in a clean, horizontal layout. It includes features to **validate input fields, delete individual users,** and **shows popup messages for missing inputs**.

Additionally, the app uses useEffect to log changes in the user list, demonstrating the handling of side effects in React.

Concepts Used:

The concepts used in this mini-project are:

1. **JSX:** Used to structure the UI of the form, buttons, and user list.

2. **Functional Components:** The project is divided into components (Form, DisplayUser, Popup, and App) for modular code.

3. **Props:** Data and functions (addUser, deleteUser, and showError) are passed as props between parent and child components.

4. **State (useState):** Used to manage form inputs and store the list of users dynamically.

5. **Event Handling:** onChange, onSubmit, and onClick events handle input changes, form submission, and delete actions.

6. **Conditional Rendering:** Displays a “No users added yet” message when the user list is empty and conditionally renders the popup.

7. **Lists with .map():** Renders multiple user entries dynamically as they are added.

8. **Forms:** Includes text inputs, email input, and dropdown selection for gender.

9. **Lifting State Up:** The user list state is maintained in the parent (App) and passed down to child components.

10. **useEffect:** Logs a message to the console whenever the user list is updated.

11. **Basic Styling (CSS):** Styled with a clean layout, buttons, popup alerts, and row-based user info display.

Learning Outcomes:

- Learned how to manage dynamic data efficiently using React state (useState) for form inputs and user lists.
- Improved understanding of event handling for form submissions, input changes, and deleting list items.
- Gained practical experience in conditional rendering and dynamically displaying components using .map().
- Understood the useEffect hook for handling side effects and monitoring state changes in React applications.

Code: App.js:

```
JS App.js   # App.css
src > JS App.js > ...
import React, { useState, useEffect } from "react";
import "./App.css";

function Form({ addUser, showError }) {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
  const [gender, setGender] = useState("");

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

    if (!name || !email || !gender) {
      showError("⚠ Please fill out all fields before adding a user.");
      return;
    }

    addUser({ name, email, gender });
    setName("");
    setEmail("");
    setGender("");
  };

  return (
    <form onSubmit={handleSubmit} className="form-container">
      <input
        type="text"
        placeholder="Enter name"
        value={name}
        onChange={(e) => setName(e.target.value)}
      />
Ln 107, Col 1   Spaces: 2   UTF-8   LF   {} JavaScript   Go Live
```

App.js App.css

src > App.js > ...

```javascript
function Form({ addUser, showError }) {
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
        <input
          type="email"
          placeholder="Enter email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
        />
        <select value={gender} onChange={(e) => setGender(e.target.value)}>
          <option value="">Select Gender</option>
          <option value="Male">Male</option>
          <option value="Female">Female</option>
          <option value="Other">Other</option>
        </select>
        <button type="submit">Add User</button>
      </form>
    );
  }

  function DisplayUser({ users, deleteUser }) {
    if (users.length === 0) {
      return <p className="no-user">No users added yet.</p>;
    }

    return (
      <ul>
        {users.map((user, index) => (
```

Ln 107, Col 1 Spaces: 2 UTF-8 LF {} JavaScript Go Live

App.js App.css

src > App.js > ...

```javascript
function DisplayUser({ users, deleteUser }) {
    return (
      <ul>
        {users.map((user, index) => (
          <li key={index} className="user-card">
            <p><strong>Name:</strong> {user.name}</p>
            <p><strong>Email:</strong> {user.email}</p>
            <p><strong>Gender:</strong> {user.gender}</p>
            <button onClick={() => deleteUser(index)} className="delete-btn">
              Delete
            </button>
          </li>
        ))}
      </ul>
    );
  }

  function Popup({ message }) {
    return message ? <div className="popup">{message}</div> : null;
  }

  function App() {
    const [users, setUsers] = useState([]);
    const [popupMessage, setPopupMessage] = useState("");

    const addUser = (user) => {
      setUsers((prevUsers) => [...prevUsers, user]);
    };

    const deleteUser = (index) => {
```

Ln 107, Col 1 Spaces: 2 UTF-8 LF {} JavaScript Go Live

```javascript
function App() {

  const deleteUser = (index) => {
    setUsers((prevUsers) => prevUsers.filter((_, i) => i !== index));
  };

  const showError = (msg) => {
    setPopupMessage(msg);
    setTimeout(() => setPopupMessage(""), 3000);
  };

  useEffect(() => {
    if (users.length > 0) {
      console.log("Users list updated!", users);
    }
  }, [users]);

  return (
    <div className="app-container">
      <h1>User Info Collector</h1>
      <Form addUser={addUser} showError={showError} />
      <DisplayUser users={users} deleteUser={deleteUser} />
      <Popup message={popupMessage} />
    </div>
  );
}

export default App;
```

App.css:

```css
.app-container {
  text-align: center;
  font-family: Arial, sans-serif;
  padding: 20px;
  background-color: #f0f8ff;
  min-height: 100vh;
}
h1 {
  color: #333;
}

.form-container {
  display: flex;
  gap: 10px;
  justify-content: center;
  margin-bottom: 20px;
}

input, select, button {
  padding: 8px;
  border: 1px solid #ccc;
  border-radius: 4px;
}
button {
  background-color: #007bff;
  color: white;
  cursor: pointer;
}
button:hover {
  background-color: #0056b3;
```

App.js App.css

src > App.css > .error-msg

```css
button:hover {
  background-color: #0056b3;
}

ul {
  list-style-type: none;
  padding: 0;
  max-width: 600px;
  margin: 0 auto;
}

.user-card {
  display: flex;
  justify-content: space-between;
  align-items: center;
  background-color: white;
  padding: 10px;
  margin: 5px 0;
  border: 1px solid #ddd;
  border-radius: 6px;
  box-shadow: 0px 2px 5px rgba(0, 0, 0, 0.1);
}

.user-card p {
  margin: 0 10px;
  flex: 1;
  text-align: left;
}
.no-user {
  color: gray;
```

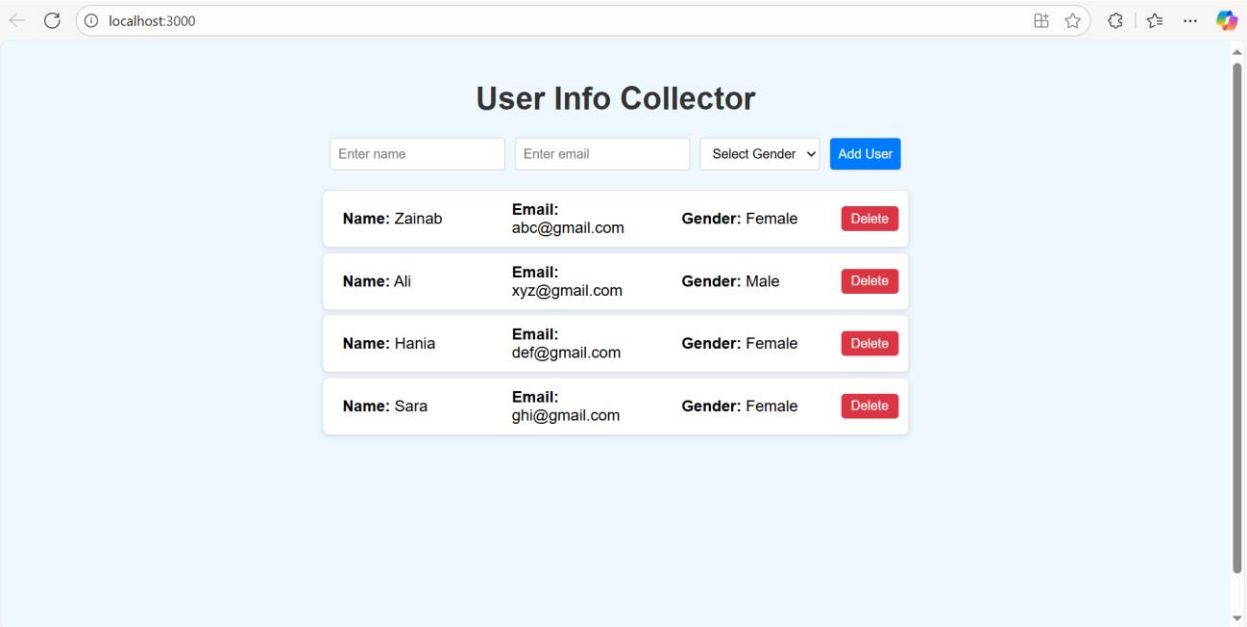
Ln 63, Col 2 Spaces: 2 UTF-8 LF {} CSS Go Live

App.js App.css

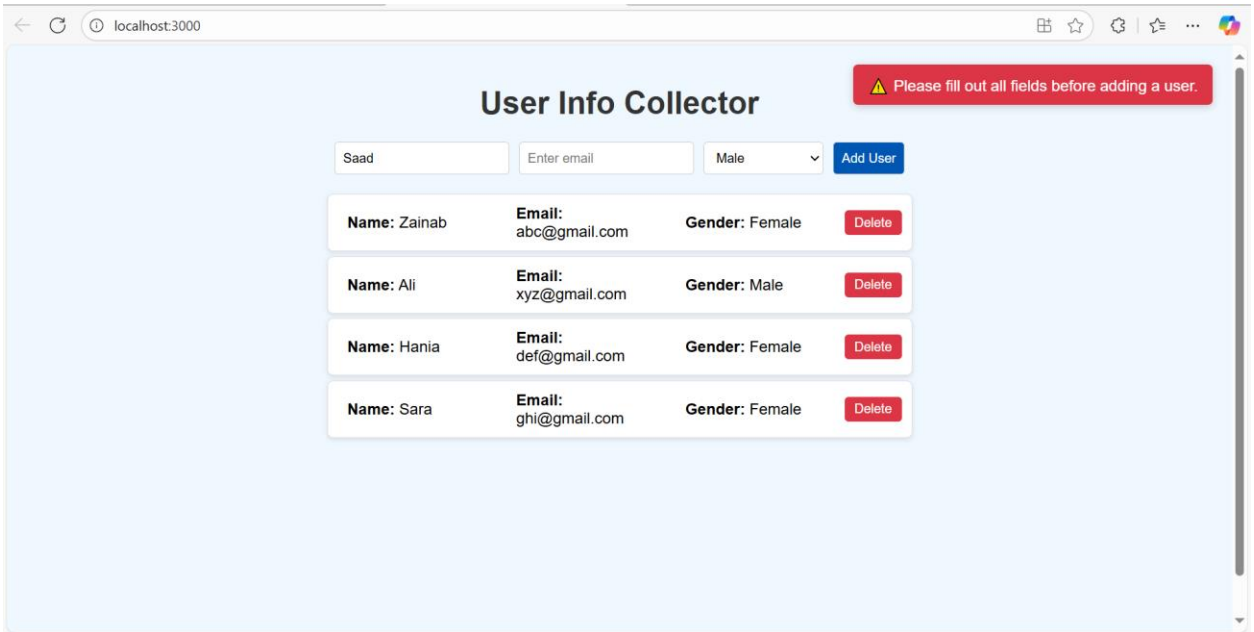
src > App.css > .error-msg

```css
.no-user {
  color: gray;
}
.error-msg {
  color: red;
  margin-top: 2px;
}
.delete-btn {
  background-color: #dc3545;
  color: white;
  border: none;
  padding: 5px 10px;
  border-radius: 4px;
  cursor: pointer;
}
.delete-btn:hover {
  background-color: #c82333;
}
.popup {
  position: fixed;
  top: 20px;
  right: 20px;
  background-color: #dc3545;
  color: white;
  padding: 10px 15px;
  border-radius: 5px;
  box-shadow: 0px 2px 8px rgba(0, 0, 0, 0.2);
  z-index: 1000;
}
```

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



Error msg if all fields are not filled:



Conclusion:

This mini-project demonstrated the key React.js concepts learned in Week 6, including state management, event handling, and useEffect. It provided hands-on experience in building a dynamic, component-based application, strengthening my understanding of front-end development.