

| Title: Demonstrate React Hooks |
| --- |

**AIM:** To Implement the React Axios

**Problem Definition:**

### Task: User Profile Editor

1. Create a form with fields for the user's name, age, and email.
2. Display the entered information below the form.
3. Add a button to toggle the visibility of the form.
4. Add a button to reset the form fields to their initial values.

### Task: Task Manager

1. Create a task manager with the following features:
   * An input field to add a new task.
   * A list to display added tasks.
   * A button to mark tasks as complete.
   * A button to toggle the visibility of completed tasks.
   * A button to reset the task list.

### Task: User Profile Manager

1. Create a form with fields for the user's name, age, and email.
2. Fetch initial profile data when the component mounts.
3. Display the fetched profile data in the form fields.
4. Allow the user to update their profile information.
5. Display the updated profile information below the form.
6. Log a message to the console whenever the profile data is updated.

**Resources used:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

**CO 2:**. Illustrate the concepts of various front-end, back-end web application development technologies & frameworks using different web development tools.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Shelly Powers Learning Node O’ Reilly 2 nd Edition, 2016.

**Pre Lab/ Prior Concepts:**

**Write details about the following content**

* Functional component , Class Component and React Life cycle

### Functional Component

* **Definition**: A simpler way to create components using JavaScript functions.
* **Characteristics**:
  + Uses React Hooks (useState, useEffect) for state and side effects.
  + No this keyword.

### Class Component

* **Definition**: An older method to create components using ES6 classes.
* **Characteristics**:
  + Manages state with this.state and updates with this.setState().
  + Utilizes lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount.
  + Uses the this keyword.

### React Lifecycle

* **Definition**: Phases a component goes through from creation to removal.
* **Phases**:
  + **Mounting**: Component is created and inserted (componentDidMount, useEffect with empty dependencies).
  + **Updating**: Component is re-rendered (componentDidUpdate, useEffect with dependencies).
  + **Unmounting**: Component is removed (componentWillUnmount, cleanup in useEffect).
  + **Error Handling**: Catches errors during rendering (componentDidCatch, error boundaries).

**Implementation Details:**

### 1] Task: User Profile Editor:

### Form.jsx

import React, { useState } from "react";

import "../styles/form.css";

const Form = () => {

  const [details, setDetails] = useState({

    username: "",

    age: "",

    email: "",

  });

  const [loading, setLoading] = useState(true);

  const handleChange = (e) => {

    const { name, value } = e.target;

    setDetails({

      ...details,

      [name]: value,

    });

  };

  const toggleState = (e) => {

    e.preventDefault();

    setLoading(!loading);

  };

  const resetFields = (e) => {

    e.preventdelfault();

    setDetails({

      username: "",

      age: "",

      email: "",

    });

  };

  return (

    <>

      {loading ? (

        <div className="wrapper">

          <form>

            <h1>JOIN THE CULT!</h1>

            <div className="Form">

              <input

                type="text"

                name="username"

                placeholder="Username"

                value={details.username}

                onChange={handleChange}

                required

              />

              <input

                type="number"

                name="age"

                placeholder="Age"

                value={details.age}

                onChange={handleChange}

                required

              />

              <input

                type="text"

                name="email"

                placeholder="Email"

                value={details.email}

                onChange={handleChange}

                required

              />

            </div>

            <div className="buttons">

              <button className="button" onClick={toggleState} type="submit">

                TOGGLE

              </button>

              <button className="button" onClick={resetFields} type="submit">

                RESET

              </button>

            </div>

          </form>

        </div>

      ) : (

        <>

          <div className="info">

            <h2>User Information</h2>

            <p>

              <strong>Username:</strong> {details.username}

            </p>

            <p>

              <strong>Age:</strong> {details.age}

            </p>

            <p>

              <strong>Email:</strong> {details.email}

            </p>

          </div>

          <div className="buttons">

            <button className="button" onClick={toggleState}>

              Edit Information

            </button>

          </div>

        </>

      )}

    </>

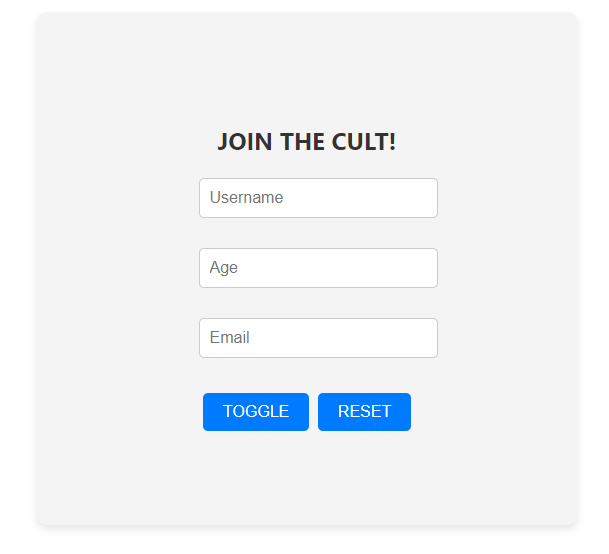
  );

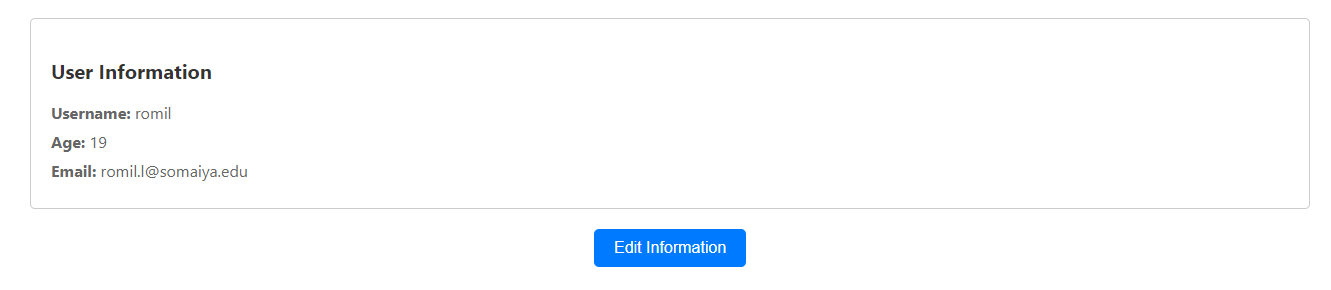
};

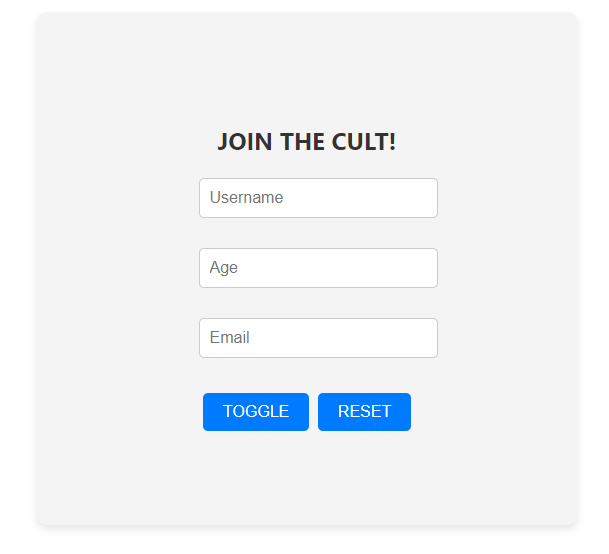
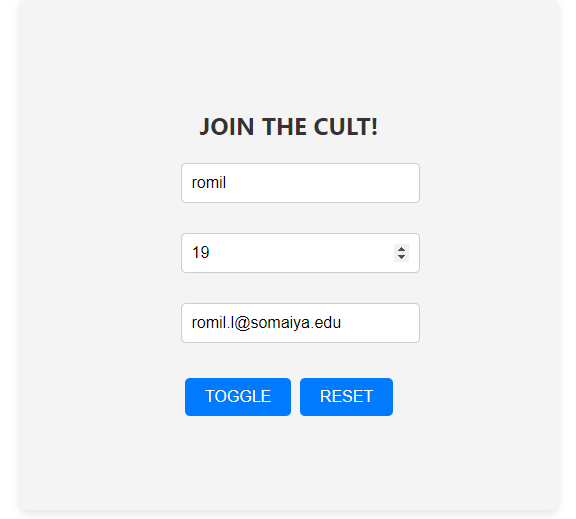
export default Form;

**Output:**

**Initial form:**

****

**Toggle the form : On clicking edit info button: On clicking reset button:**

****

### 2] Task: Task Manager

**Task.jsx**

import React from "react";

const Task = ({ allTasks, setAllTasks, id }) => {

  const handleCheck = (event) => {

    let newTasks = [...allTasks];

    newTasks[id].checked = event.target.checked;

    setAllTasks(newTasks);

  };

  return (

    <div className="task">

      <h3>{allTasks[id].heading}</h3>

      <input

        type="checkbox"

        checked={allTasks[id].checked}

        onChange={handleCheck}

      />

    </div>

  );

};

export default Task;

**TaskList.jsx**

import React, { useState } from "react";

import Task from "./Task";

import "../styles/task.css";

const TaskList = () => {

  const [done, setDone] = useState(false);

  const [newTask, setNewTask] = useState("");

  const [allTasks, setAllTasks] = useState([]);

  const showTasks = (e) => {

    e.preventDefault();

    setDone(!done);

  };

  const resetFields = (e) => {

    e.preventDefault();

    setAllTasks([]);

  };

  const handleSubmit = (e) => {

    e.preventDefault();

    setAllTasks([

      ...allTasks,

      { heading: newTask, checked: false, taskId: allTasks.length },

    ]);

    setNewTask("");

  };

  return (

    <>

      <div className="wrapper">

        <form onSubmit={handleSubmit}>

          <h1>ADD YOUR TASKS</h1>

          <div className="Form">

            <input

              type="text"

              name="Task"

              placeholder="Enter your task"

              value={newTask}

              onChange={(e) => setNewTask(e.target.value)}

              required

            />

          </div>

          <div className="buttons">

            <button className="button" type="submit">

              SUBMIT

            </button>

          </div>

        </form>

      </div>

      <div className="buttons2">

        <button className="button" onClick={resetFields} type="button">

          RESET

        </button>

        <button className="button" onClick={showTasks}>

          {done ? "SHOW ALL TASKS" : "SHOW COMPLETED TASKS"}

        </button>

      </div>

      <div className="taskinfo">

        <h2>Task List</h2>

        {allTasks

          .filter((task) => (done ? task.checked : true))

          .map((task, index) => (

            <Task

              key={task.taskId}

              allTasks={allTasks}

              setAllTasks={setAllTasks}

              id={index}

            />

          ))}

      </div>

    </>

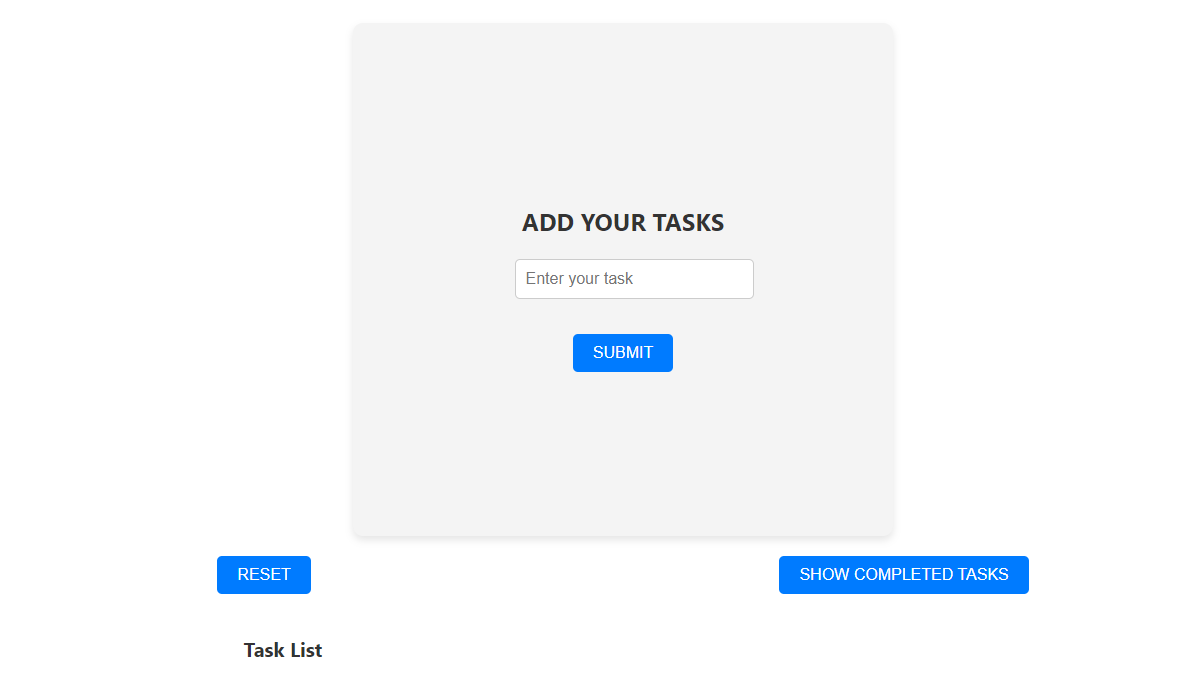
  );

};

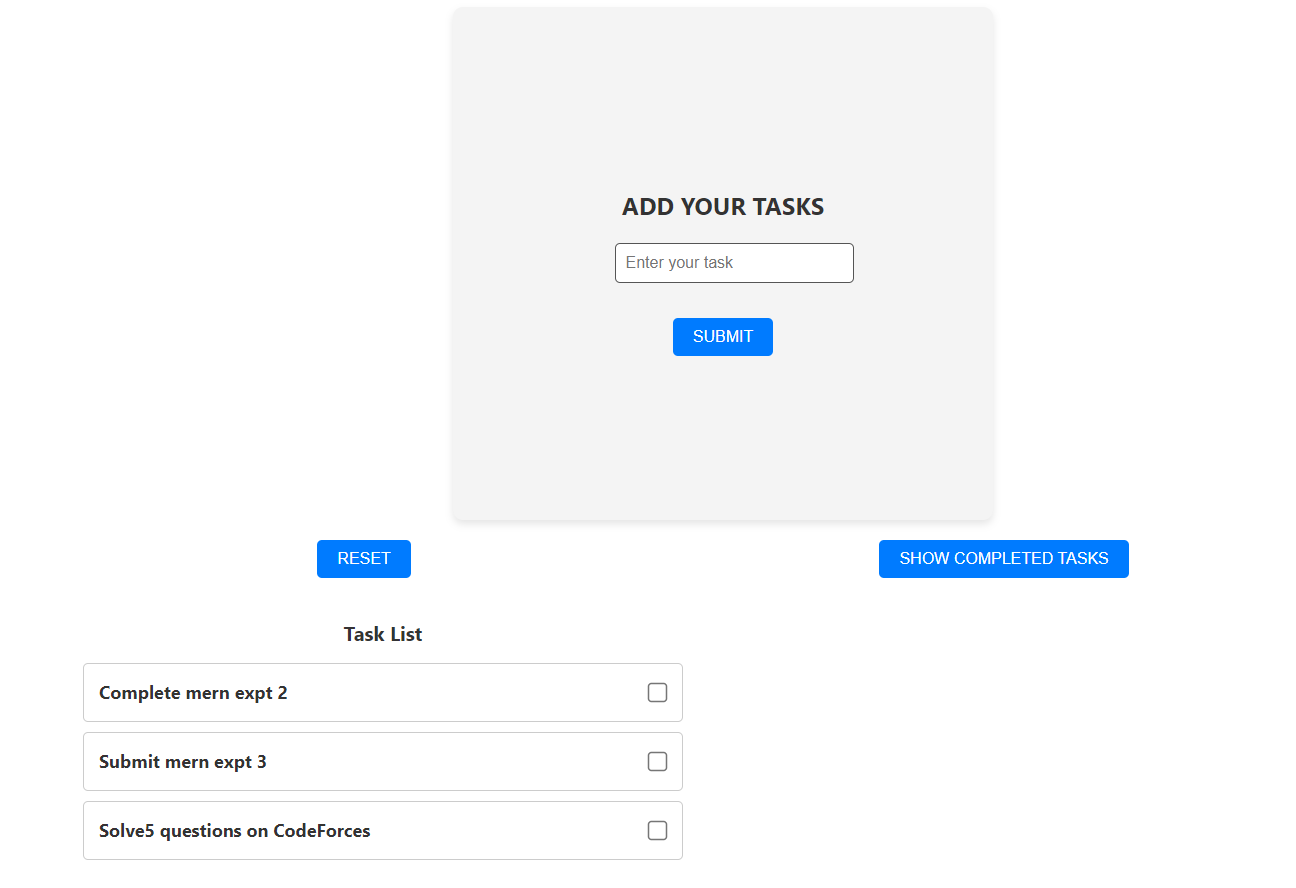
export default TaskList;

**Output:**

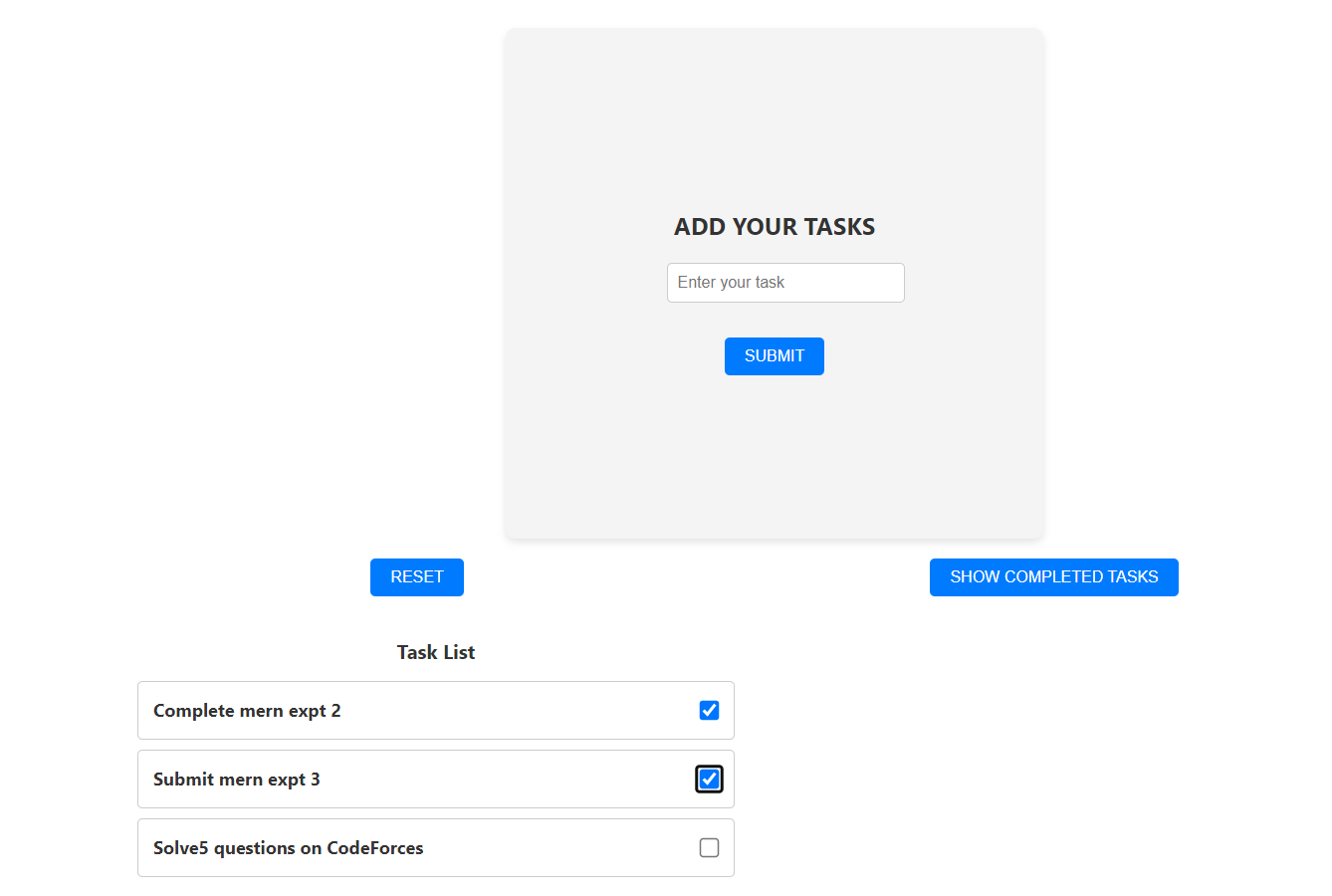
**Initial:**

****

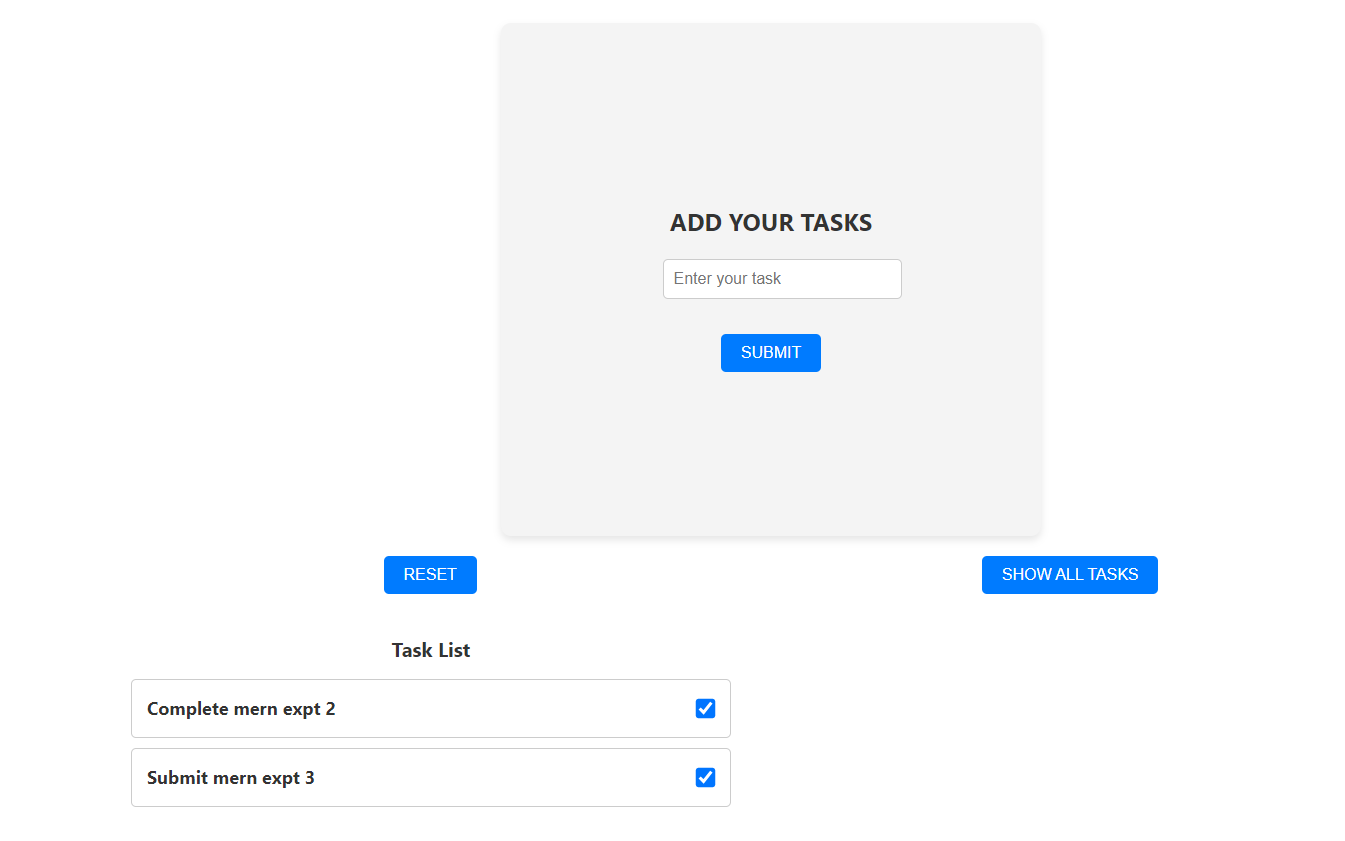
**After adding some task:**

****

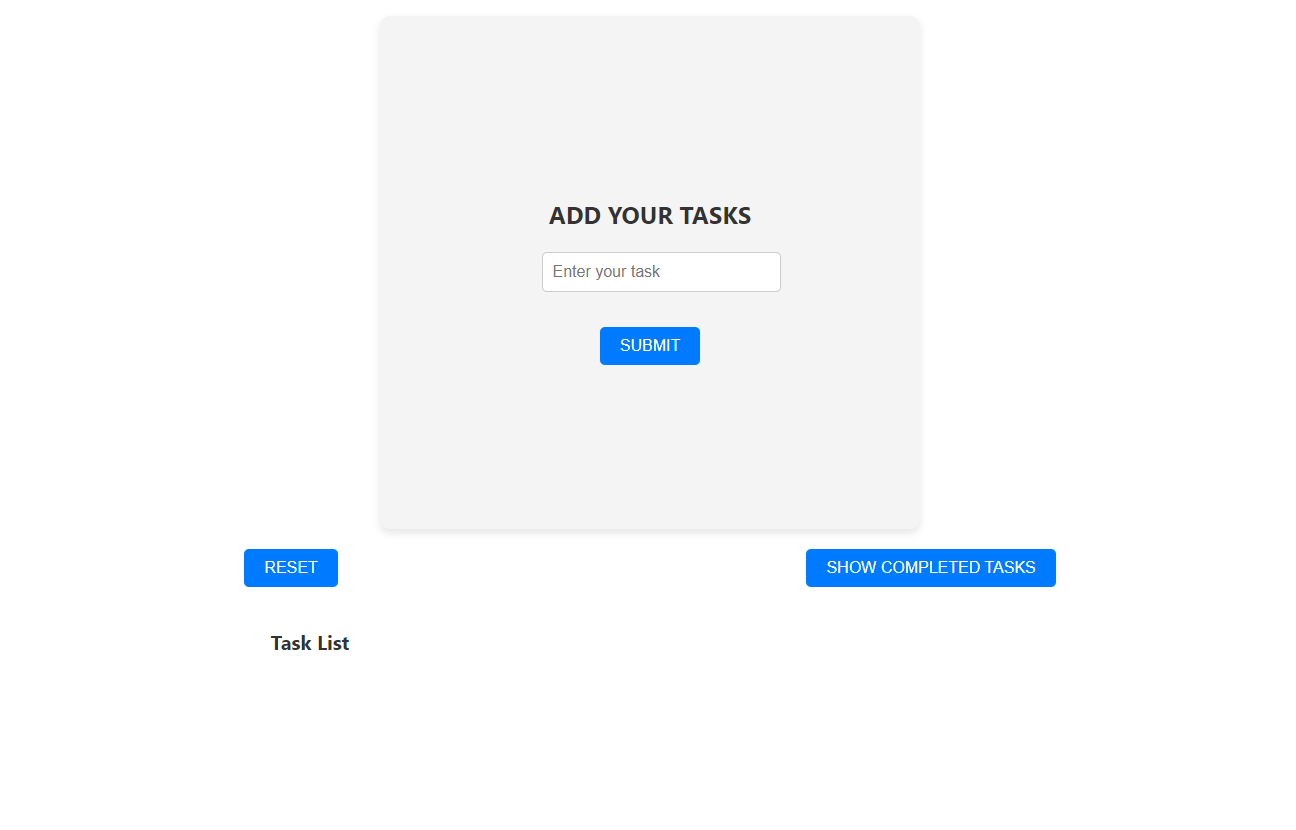
**Checking the check boxes for the task completed:**

****

**On clicking show completed tasks:**



**On clicking reset button:**



### 3] Task: User Profile Manager:

### Profile.jsx

import React from "react";

import { useState, useEffect } from "react";

import '../styles/profile.css';

const Profile = () => {

  const [profileData, setProfileData] = useState({

    username: "",

    age: "",

    email: "",

  });

  const [updatedProfile, setUpdatedProfile] = useState(null);

  useEffect(() => {

    const savedProfile = JSON.parse(localStorage.getItem("userProfile"));

    if (savedProfile) setProfileData(savedProfile);

  }, []);

  const handleChange = (e) => {

    const { name, value } = e.target;

    setProfileData({

      ...profileData,

      [name]: value,

    });

  };

  const handleSubmit = (e) => {

    e.preventDefault();

    setProfileData(profileData);

    localStorage.setItem("userProfile", JSON.stringify(profileData));

    console.log("Profile Updated: ", profileData);

  };

  return (

    <>

      <div className="wrapper">

        <form onSubmit={handleSubmit}>

          <h1>FILL YOUR DETAILS BELOW!</h1>

          <div className="Form">

            <label>Username:</label>

            <input

              type="text"

              name="username"

              placeholder="Username"

              value={profileData.username}

              onChange={handleChange}

              required

            />

            <label>Age:</label>

            <input

              type="number"

              name="age"

              placeholder="Age"

              value={profileData.age}

              onChange={handleChange}

              required

            />

            <label>Email:</label>

            <input

              type="text"

              name="email"

              placeholder="Email"

              value={profileData.email}

              onChange={handleChange}

              required

            />

          </div>

          <div className="buttons">

            <button className="button" type="submit">

              SUBMIT

            </button>

          </div>

        </form>

      </div>

      {updatedProfile && <p>Profile Updated Successfully!</p>}

      <div className="user-details">

        <h1>Your Profile:</h1>

        <div className="Form">

          <p>

            <strong>Name:</strong> {profileData.username}

          </p>

          <p>

            <strong>Age:</strong> {profileData.age}

          </p>

          <p>

            <strong>Email:</strong> {profileData.email}

          </p>

        </div>

      </div>

      <div className="buttons">

        {/\* <button className="button" onClick={() => setProfileData({

         username: "",

         age: "",

         email: ""

        })}>

          Reset Profile

        </button> \*/}

        <button className="button" onClick={() => localStorage.clear()}>

          Clear Local Storage

        </button>

        <button className="button" onClick={() => window.location.reload()}>

          Refresh Page

        </button>

      </div>

    </>

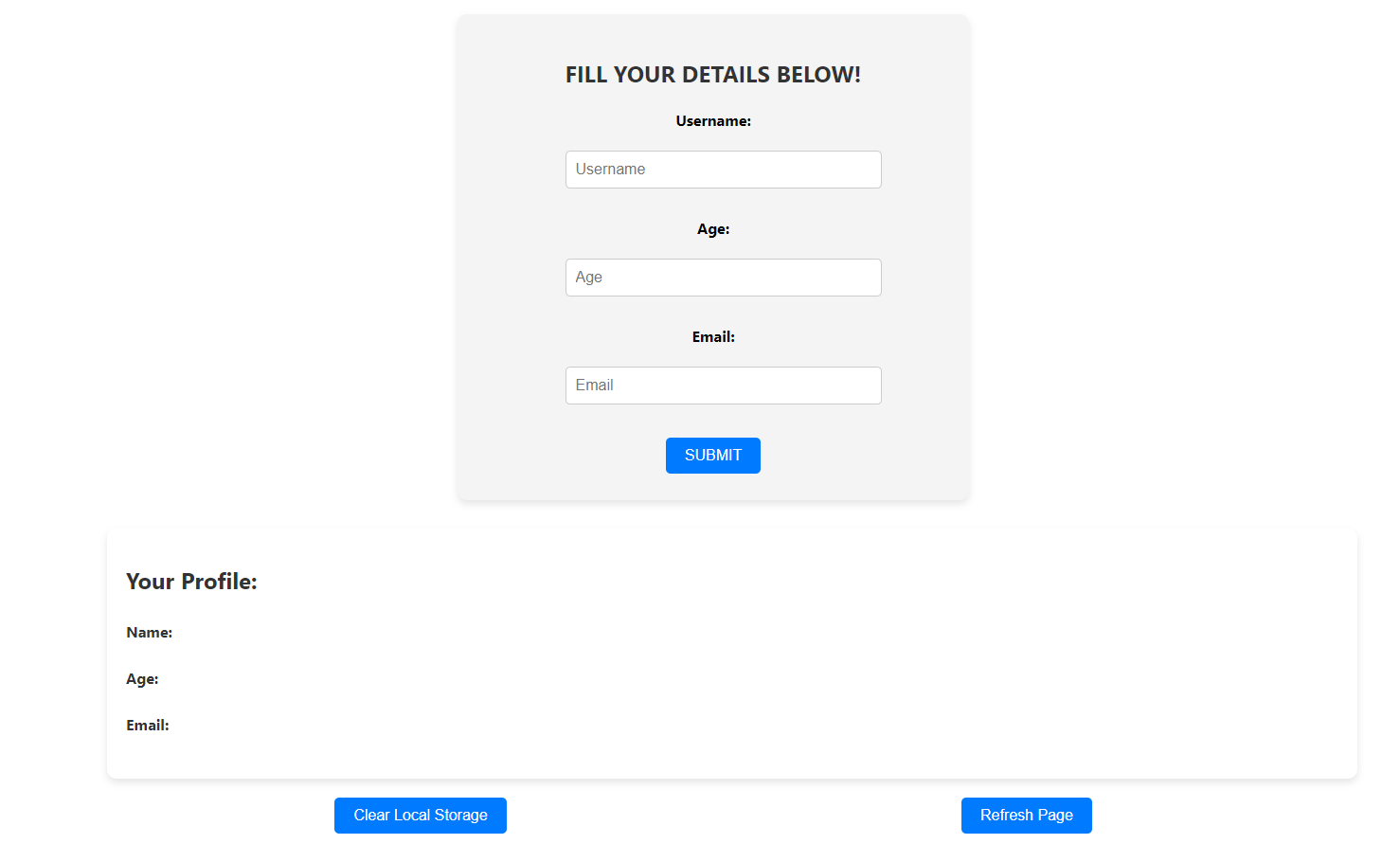
  );

};

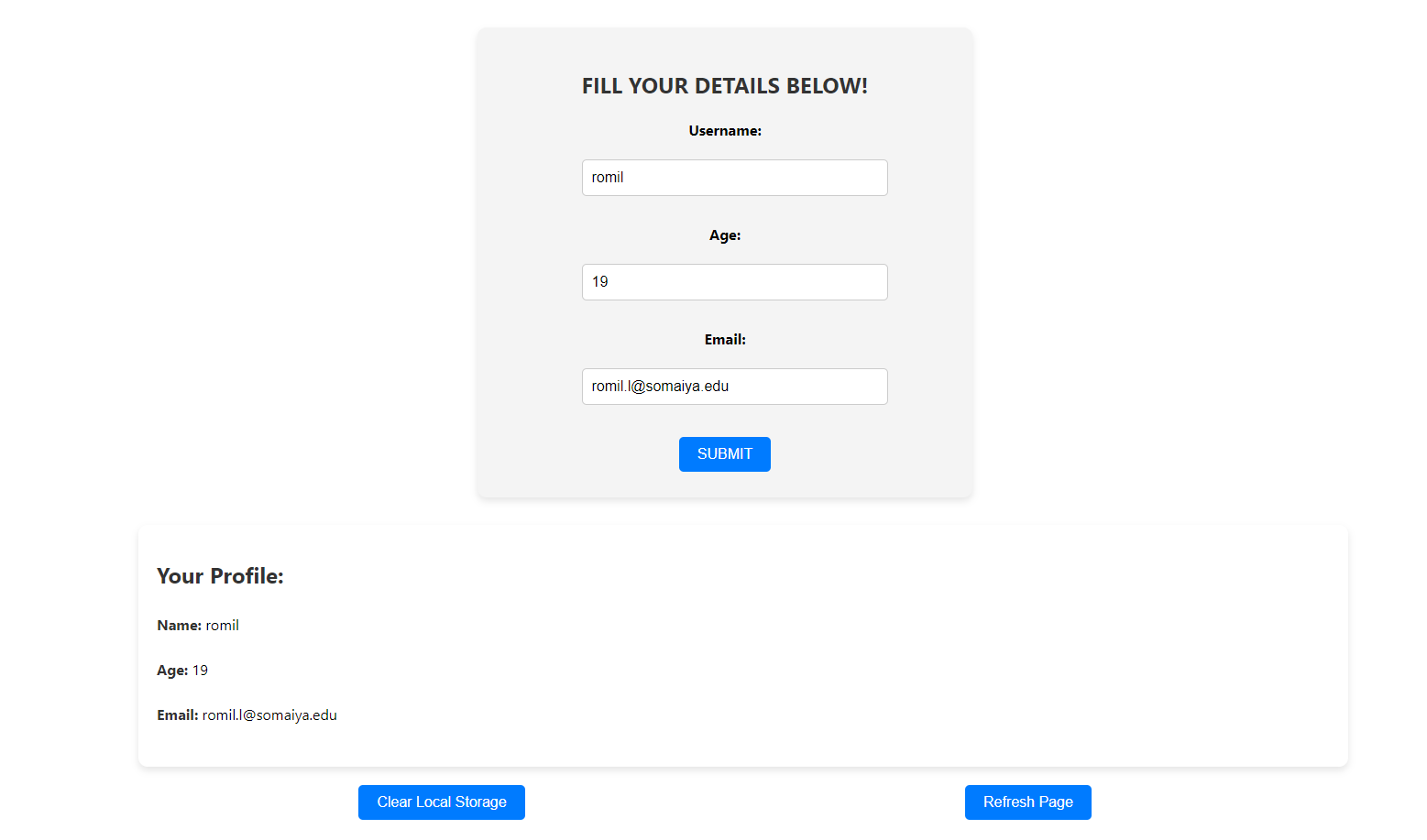
export default Profile;

**Output:**

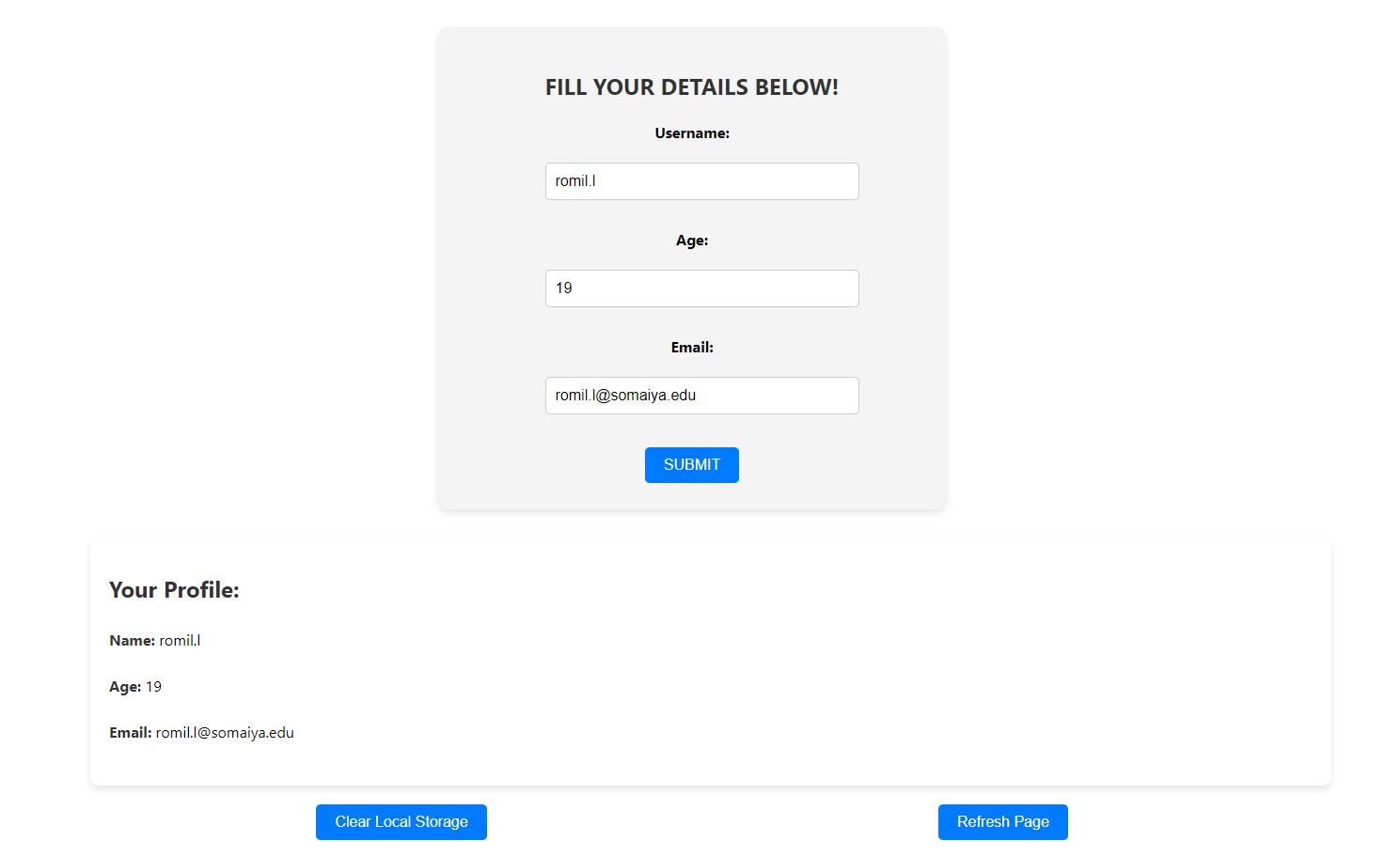
**Initially:**



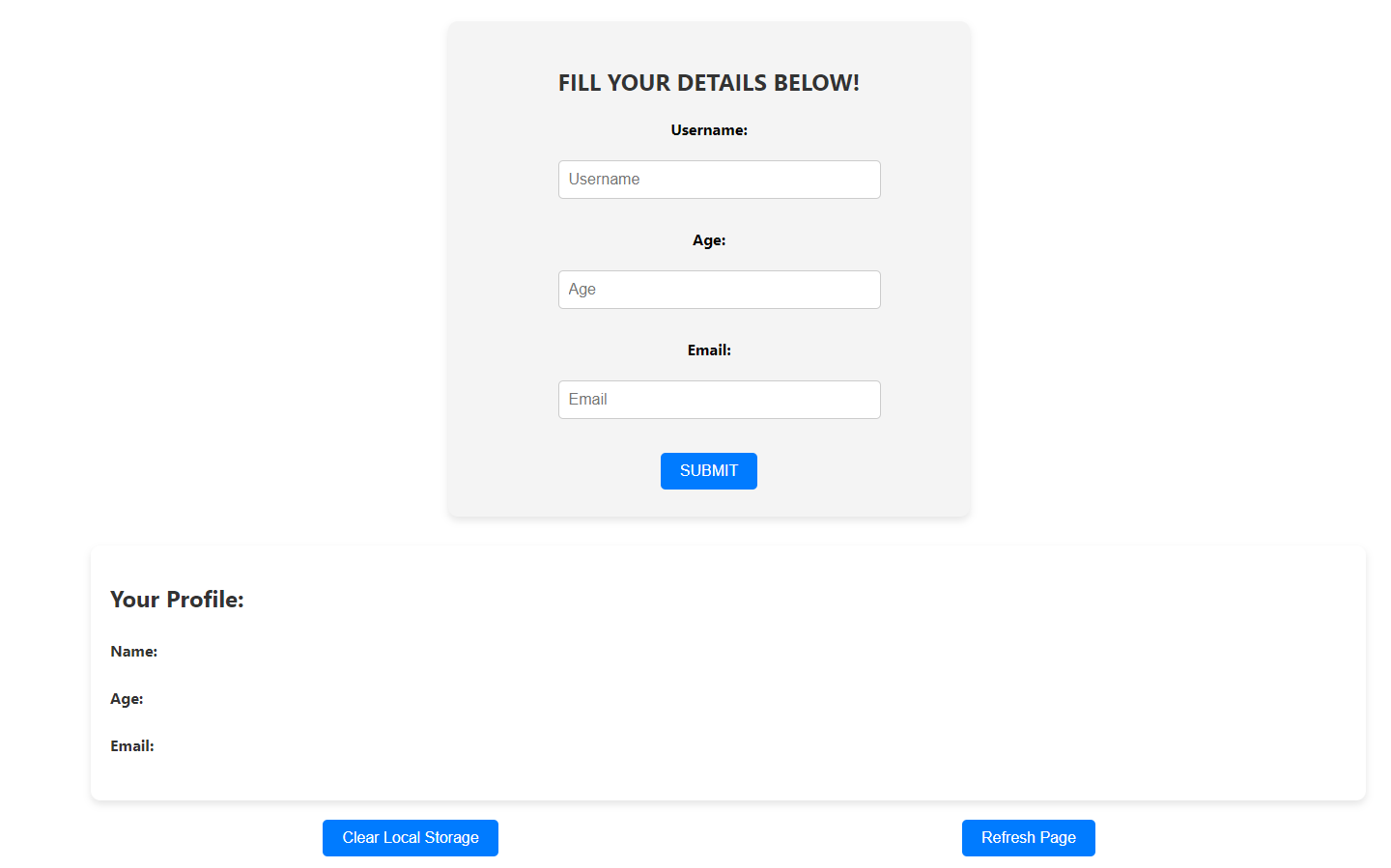
**On entering data:**



**On refreshing the page via refresh button:**

****

**After clearing the local storage and refreshing:**

****

**Conclusion:** Implementing React Hooks with Axios enables efficient management of form data and state, allowing dynamic user profile editing, task management, and real-time updates with streamlined code and improved user experience.

**Postlab questions:**

1) Different type of hooks

1. **Basic Hooks**:
   * **useState**: Manages local component state.
   * **useEffect**: Handles side effects such as data fetching and subscriptions.
   * **useContext**: Accesses context values without nesting.
2. **Additional Hooks**:
   * **useReducer**: Manages more complex state logic using reducers.
   * **useCallback**: Memoizes callback functions to prevent unnecessary re-renders.
   * **useMemo**: Memoizes expensive calculations to optimize performance.
3. **Custom Hooks**:
   * **Custom hooks**: User-defined hooks to encapsulate and reuse logic across components.
4. **Imperative Hooks**:
   * **useRef**: Accesses and interacts with DOM elements or stores mutable values.

These hooks help manage state, handle side effects, optimize performance, and more in functional components.