**6. ReactJS-HOL**

Project: TrainersApp

A React-based SPA to display a list of trainers and their details using navigation and dynamic routing

**Step-by-Step Implementation**

**Create React App**

npx create-react-app TrainersApp

cd TrainersApp

code .

**Install React Router**

npm install react-router-dom

**trainer.js**

class Trainer {

constructor(TrainerId, Name, Email, Phone, Technology, Skills) {

this.TrainerId = TrainerId;

this.Name = Name;

this.Email = Email;

this.Phone = Phone;

this.Technology = Technology;

this.Skills = Skills;

}

}

export default Trainer;

**TrainersMock.js**

const trainers = [

{

TrainerId: 1,

Name: "Shashwat Dodamani",

Email: "shashwat@example.com",

Phone: "7483819949",

Technology: "AI/ML",

Skills: "Python, ML, DL"

},

{

TrainerId: 2,

Name: "Ananya Rao",

Email: "ananya@example.com",

Phone: "9845012345",

Technology: "Full Stack",

Skills: "React, Node.js, MongoDB"

}

];

export default trainers;

**Home.js**

import React from 'react';

function Home() {

return (

<div>

<h2>Welcome to Trainers Portal</h2>

<p>View the list of trainers and their expertise in various technologies.</p>

</div>

);

}

export default Home;

**TrainerList.js**

import React from 'react';

import { Link } from 'react-router-dom';

function TrainerList({ trainers }) {

return (

<div>

<h2>Trainers List</h2>

<ul>

{trainers.map(trainer => (

<li key={trainer.TrainerId}>

<Link to={`/trainers/${trainer.TrainerId}`}>

{trainer.Name}

</Link>

</li>

))}

</ul>

</div>

);

}

export default TrainerList;

**TrainerDetails.js**

import React from 'react';

import { useParams } from 'react-router-dom';

import trainers from './TrainersMock';

function TrainerDetails() {

const { id } = useParams();

const trainer = trainers.find(t => t.TrainerId === parseInt(id));

if (!trainer) {

return <p>Trainer not found.</p>;

}

return (

<div>

<h2>{trainer.Name}</h2>

<p><strong>Email:</strong> {trainer.Email}</p>

<p><strong>Phone:</strong> {trainer.Phone}</p>

<p><strong>Technology:</strong> {trainer.Technology}</p>

<p><strong>Skills:</strong> {trainer.Skills}</p>

</div>

);

}

export default TrainerDetails;

**App.js**

import React from 'react';

import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';

import Home from './Home';

import TrainerList from './TrainerList';

import TrainerDetails from './TrainerDetails';

import trainers from './TrainersMock';

function App() {

return (

<Router>

<div>

<nav style={{ marginBottom: "20px" }}>

<Link to="/">Home</Link> | {" "}

<Link to="/trainers">Trainers</Link>

</nav>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/trainers" element={<TrainerList trainers={trainers} />} />

<Route path="/trainers/:id" element={<TrainerDetails />} />

</Routes>

</div>

</Router>

);

}

export default App;

**Run the App**

npm start

Open your browser at:  
<http://localhost:3000>

**7. ReactJS-HOL**

**Props in React**

Props (short for "properties") are a mechanism for passing data from a parent component to a child component in React. They make components reusable by giving each instance its own data.  
**Key points about props:**

* Read-only: Props are immutable inside the child; they cannot be changed by the receiving component.
* Passed from parent to child: Data flows one way, from the parent to the child.
* Accessed via this.props in class components, or directly as an argument in function components.

**Default Props**

**Default props** provide fallback values when no prop is supplied by the parent. This is useful for making components robust and predictable.

**How they work:**

* If the parent doesn’t provide a certain prop, the component receives the default value.
* Set using Component.defaultProps for class or function components.

**State vs. Props: Key Differences**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Props** | **State** |
| Source | Passed in by parent component | Managed inside the component |
| Mutability | Immutable (read-only in recipient) | Mutable within the component |
| Purpose | External data/configuration | Internal, local data that changes over time |
| Access | this.props (class) or function args | this.state, useState hook (function) |
| Set by | Parent component | The component itself (setState/useState) |
| Lifecycle | Fixed for a render, can be updated by parent | Can change due to user actions, events, etc. |

**ReactDOM.render() Explained**

ReactDOM.render() is a method used to render a React element (or component) into the DOM in web applications. It is how you display your top-level React component(s) inside a specific DOM node.

**Create a React app named shoppingapp with the following structure:**

* Cart component: Accepts Itemname and Price as props
* OnlineShopping component: Holds an array of 5 cart items and passes them as props to the Cart component
* Display each item using a loop

**Step-by-Step Implementation**

**Create React App**

npx create-react-app shoppingapp

cd shoppingapp

code .

**Cart.js (Class Component with Props)**

import React from 'react';

class Cart extends React.Component {

render() {

return (

<div>

<p><strong>Item:</strong> {this.props.Itemname}</p>

<p><strong>Price:</strong> ₹{this.props.Price}</p>

<hr />

</div>

);

}

}

// Default Props

Cart.defaultProps = {

Itemname: 'Unknown Item',

Price: 0

};

export default Cart;

**OnlineShopping.js (Parent Component)**

src/OnlineShopping.js

import React from 'react';

import Cart from './Cart';

class OnlineShopping extends React.Component {

render() {

const cartItems = [

{ Itemname: 'Laptop', Price: 75000 },

{ Itemname: 'Headphones', Price: 2500 },

{ Itemname: 'Keyboard', Price: 1500 },

{ Itemname: 'Mouse', Price: 800 },

{ Itemname: 'Monitor', Price: 12000 }

];

return (

<div>

<h2>Shopping Cart</h2>

{cartItems.map((item, index) => (

<Cart key={index} Itemname={item.Itemname} Price={item.Price} />

))}

</div>

);

}

}

export default OnlineShopping;

**App.js**

import React from 'react';

import './App.css';

import OnlineShopping from './OnlineShopping';

function App() {

return (

<div className="App">

<OnlineShopping />

</div>

);

}

export default App;

**ReactDOM.render()**

src/index.js

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<App />);

**8. ReactJS-HOL**

React app named **counterapp** with a component named CountPeople.  
This component should:

* Use a constructor to initialize state
* Have two methods:
  + UpdateEntry() – increments entry count
  + UpdateExit() – increments exit count
* Render two buttons:
  + **Login** – triggers UpdateEntry()
  + **Exit** – triggers UpdateExit()

**Step-by-Step Implementation**

**Create the React App**

npx create-react-app counterapp

cd counterapp

code .

**Create CountPeople.js**

import React, { Component } from 'react';

class CountPeople extends Component {

constructor(props) {

super(props);

this.state = {

entryCount: 0,

exitCount: 0

};

}

UpdateEntry = () => {

this.setState(prevState => ({

entryCount: prevState.entryCount + 1

}));

};

UpdateExit = () => {

this.setState(prevState => ({

exitCount: prevState.exitCount + 1

}));

};

render() {

return (

<div style={{ textAlign: 'center', marginTop: '50px' }}>

<h2>People Counter</h2>

<p><strong>Number of People Entered:</strong> {this.state.entryCount}</p>

<p><strong>Number of People Exited:</strong> {this.state.exitCount}</p>

<button onClick={this.UpdateEntry} style={{ marginRight: '10px' }}>

Login

</button>

<button onClick={this.UpdateExit}>

Exit

</button>

</div>

);

}

}

export default CountPeople;

**Update App.js**

import React from 'react';

import './App.css';

import CountPeople from './CountPeople';

function App() {

return (

<div className="App">

<CountPeople />

</div>

);

}

export default App;

**Output on Browser**

People Counter

Number of People Entered: 0

Number of People Exited: 0

[Login] [Exit]

Clicking:

* **Login** → Increments entry count
* **Exit** → Increments exit count