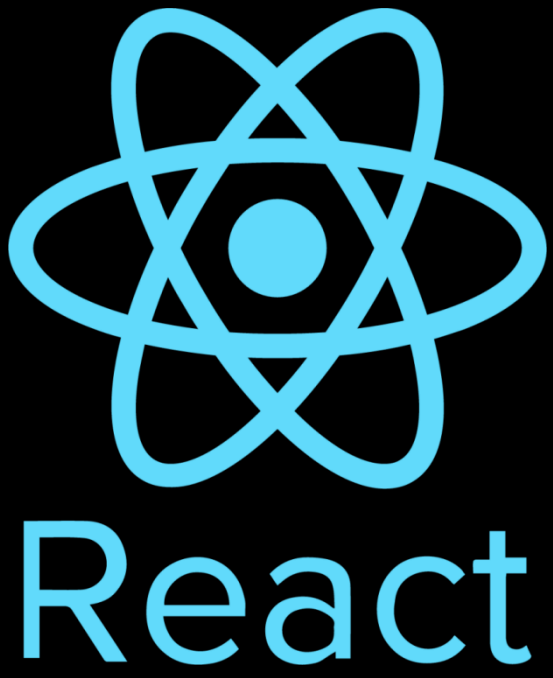
INTRODUCTION TO

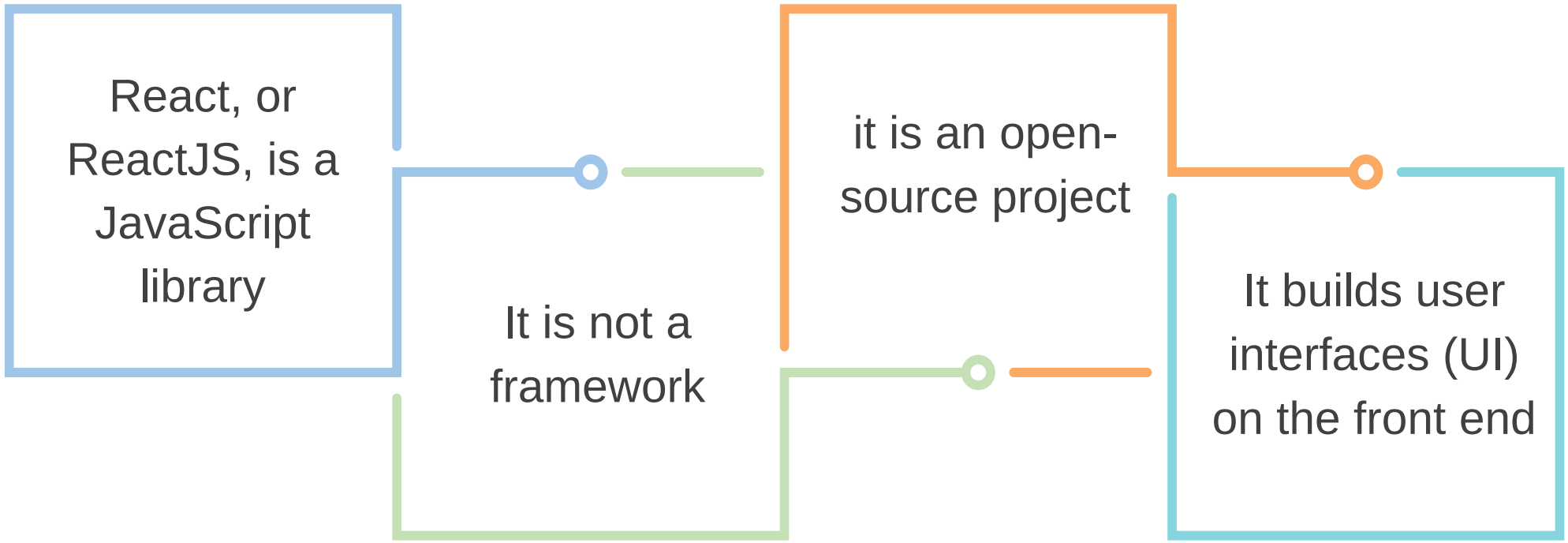


A JavaScript Library

WHAT IS REACT?

- React is a popular JavaScript library for building user interfaces (UIs).
- It was developed by Facebook and is widely used for creating dynamic and interactive web applications.
- React follows a component-based architecture, which means the UI is divided into reusable components that can be composed together to form complex interfaces.
- React introduces a declarative approach to building UIs, where developers describe how the UI should look at any given time, and React automatically updates and renders the components as necessary when the underlying data changes.
- This makes it easier to build and maintain large-scale applications by providing a clear separation between the UI and application logic.

WHY REACT?


React, or ReactJS, is a JavaScript library
It is not a framework
it is an open-source project
It builds user interfaces (UI) on the front end

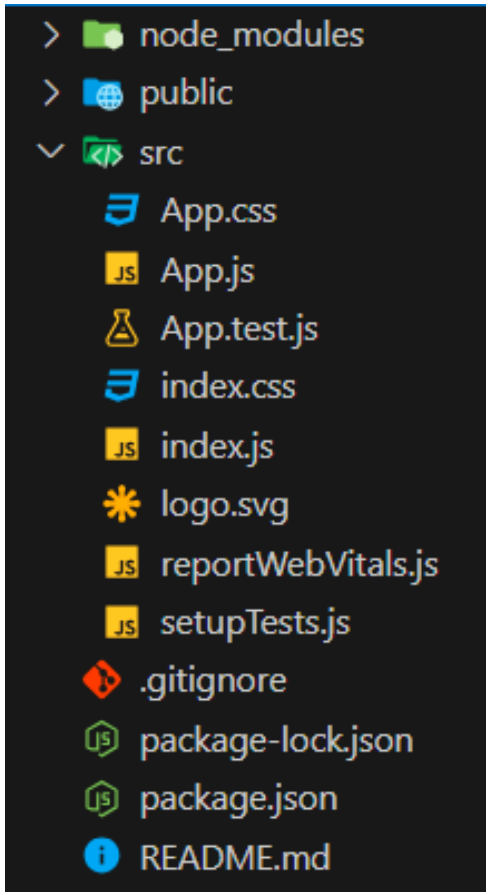
CREATE REACT PROJECT

- Open the folder where you want to create react app.
  - npx create-react-app appname
  - cd appname
  - npm start
  - You can see the default output:
    - http://localhost:3000/

PROJECT STRUCTURE

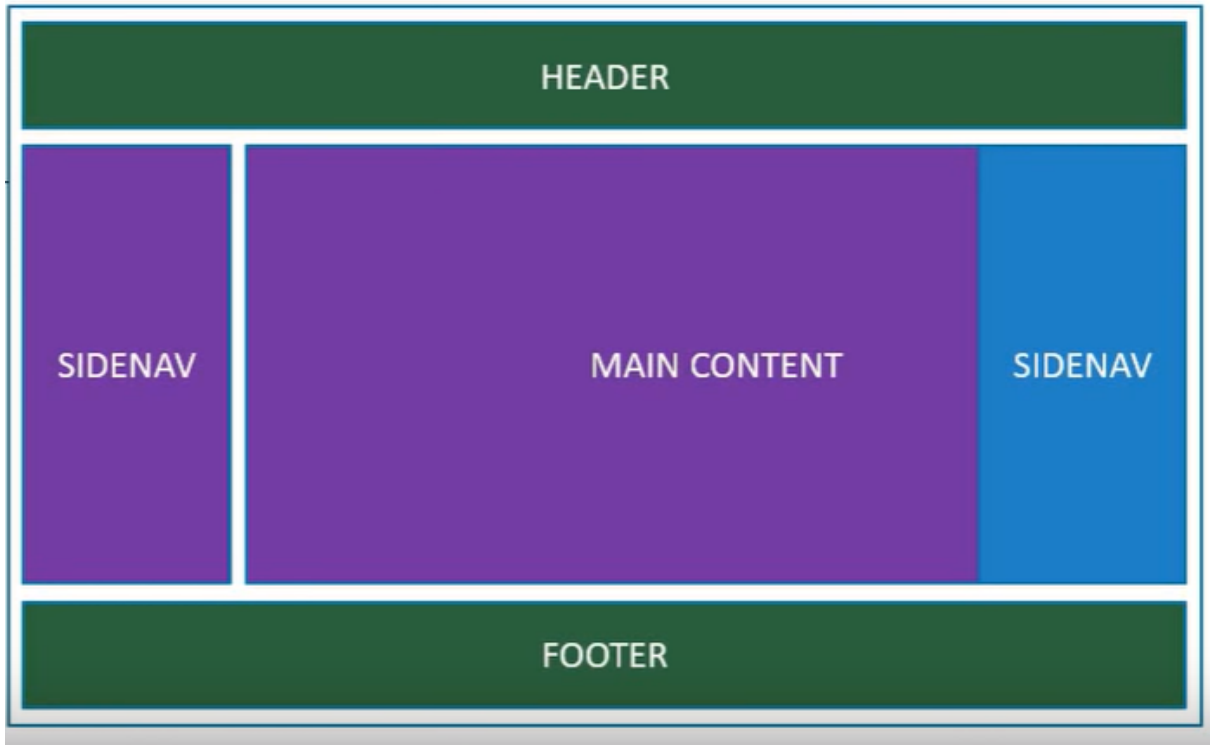
- Let's understand the folder structure:
- In **/public**, the file that is necessary is index.html, which is equivalent to the static index.html file
- **.gitignore** is used for adding the files that git will ignore
- **package.json** includes all the package versions

- **README.md** is the marked-down file used for adding the summary at the bottom of the project
- The **/src** directory includes all the React-related codes.
- **/src/App.js** is the primary component of an application. Often, it is referred to as the App component.
- **/src/App.css** allows the developer to define the visual appearance and layout of components and elements.
- **/src/index.js** has been assigned the task of rendering the root component into the DOM.



REACT COMPONENTS

- As React is a component-based architecture, developers can divide the application into individual units.
- Each component is responsible for:
  - Rendering a specific part of the UI
  - Handling its own logic
  - Creating a hierarchy



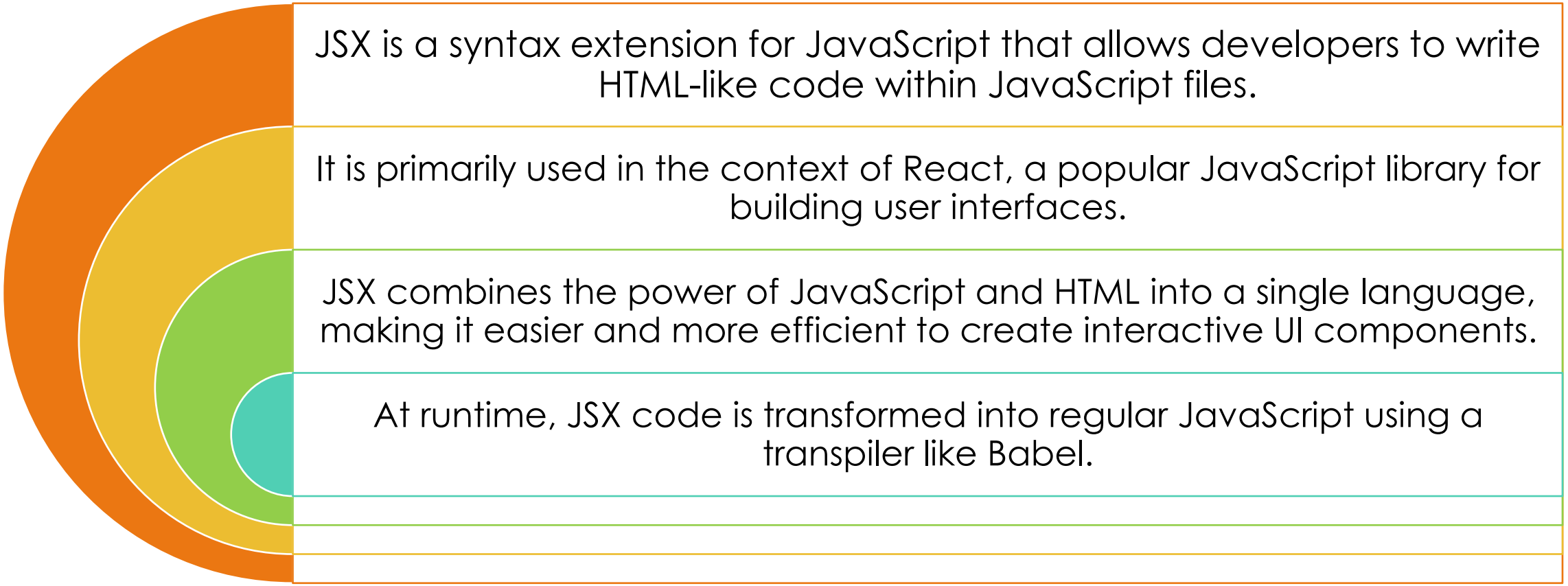
WHY FUNCTIONAL COMPONENTS

- Simplicity & Readability
- Reusability
- Good Performance
- Hooks
- Testing

CREATE COMPONENTS

- Create components for
  - Header
  - Footer
  - Slider
  - Some Featured Section
- Bootstrapping application:
  - 2 ways
  - Install bootstrap: npm I bootstrap
  - Direct copy paste CDN in index.html file inside public folder
  - You can also you React-Bootstrap

JSX (JAVASCRIPT EXTENSION)



ACTIVITY : JSX USAGE IN REACT COMPONENT

```jsx
import React from 'react';

function HelloWorld() {
  return <h1>Hello, World!</h1>;
}

export default HelloWorld;
```

Basic Rendering

```jsx
import React from 'react';

function Greeting() {
  let name="sonam soni";
  return <h1>Hello, {name}!</h1>;
}

export default Greeting;
```

Dynamic Rendering

ACTIVITY : JSX USAGE IN REACT COMPONENT

```jsx
import React from 'react';
function Greeting() {
    const isLoggedIn = false;
    return(
        <div>
            <button>{isLoggedIn?'logout':'Login'}</button>
        </div>
    )
  }

  export default Greeting;
```

Conditional Rendering

ACTIVITY : JSX USAGE IN REACT COMPONENT

```jsx
const divStyle = {
  color: 'blue',
  backgroundColor: 'lightgrey',
  padding: '10px'
};
function StyledComponent() {
  return <div style={divStyle}>
    This is a styled component!
    </div>;
}
export default StyledComponent;
```

Styling with JSX

ACTIVITY : JSX USAGE

```jsx
function FragmentExample() {
  return (
    <>
      <h1>Heading</h1>
      <p>This is a paragraph inside a fragment.</p>
    </>
  );
}

export default FragmentExample;
```

Use of Fragment <></>

ACTIVITY : JSX USAGE

```jsx
function ActionButton() {
  const handleClick = () => {
    alert('Button clicked!');
  };

  return (
    <button onClick={handleClick}>
      Click me
    </button>
  );
}

export default ActionButton;
```

Event handling

```jsx
function NumberList() {
  const numbers = [1,2,3,4,5,6,7];
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}

export default NumberList;
```

Display objects

ACTIVITY: DISPLAY JSON DATA

- Create one JSON files named products.json
- Import it to your component:
  - Import products from ‘products.json’;
  - This products variables work as an array you can directly map it
- Render the data into your component using Table format

STATE USING USESTATE() HOOK

- The useState hook is a fundamental hook in React that allows you to add state to functional components

```
function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Basic state

STATE WITH BOOLEAN

```
function Toggle() {
  const [isToggled, setIsToggled] = useState(false);

  return (
    <div>
      <p>{isToggled ? 'ON' : 'OFF'}</p>
      <button onClick={() => setIsToggled(!isToggled)}>
        Toggle
      </button>
    </div>
  );
}
```

STATE WITH FORM INPUTS

```
function Form() {
  const [name, setName] = useState('');
  const handleChange = (event) => {
    setName(event.target.value);
  };
  const handleSubmit = (event) => {
    alert('A name was submitted: ' + name);
    event.preventDefault();
  };
  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input type="text" value={name} onChange={handleChange} />
      </label>
      <button type="submit">Submit</button>
    </form>
  );
}
```

MANAGING MULTIPLE STATES

```
function MultiState() {
  const [name, setName] = useState('');
  const [age, setAge] = useState(0);

  const handleNameChange = (event) => {
    setName(event.target.value);
  };

  const handleAgeChange = (event) => {
    setAge(event.target.value);
  };
```

USE OBJECTS IN STATE

```
function UserInfo() {
  const [user, setUser] = useState({ name: '', email: '' });

  const handleChange = (event) => {
    const { name, value } = event.target;
    setUser({
      ...user,
      [name]: value
    });
  };
```

```
return (
  <div>
    <input
      type="text"
      name="name"
      placeholder="Name"
      value={user.name}
      onChange={handleChange}
    />
    <input
      type="email"
      name="email"
      placeholder="Email"
      value={user.email}
      onChange={handleChange}
    />
```

ARRAY AS A STATE

```
function ArrayState() {
  const [items, setItems] = useState([]);

  const addItem = () => {
    setItems([...items, {
      id: items.length,
      value: Math.floor(Math.random() * 10) + 1
    }]);
  };
```

```
return (
  <div>
    <button onClick={addItem}>Add a number</button>
    <ul>
      {items.map(item => (
        <li key={item.id}>{item.value}</li>
      ))}
    </ul>
  </div>
);
```

ACTIVITY: TODO APP USING STATE

- Create state todo: takes input from the user
- Create state todos: array type which will handle all the inputs
- Write function to add todo which takes todo input field and store it in todos array
- Display all array values in the list form
- Also implement delete functionality to delete todo.

COMPONENT LEVEL INTERACTION

- Parent Child Communication
- If you want to pass data from parent component to child component then you can use props

```
function Welcome(props) {
    return <h1>Hello, {props.name}</h1>;
}
export default Welcome;
```

```
function App() {
  return (
    <div>
      <Welcome name="Alice" />
      <Welcome name="Bob" />
    </div>
  );
}
```

PASSING MULTIPLE PROPS

```jsx
function UserInfo(props) {
  return (
    <div>
      <h1>{props.name}</h1>
      <p>{props.email}</p>
    </div>
  );
}
```

```jsx
function App() {
  return (
    <div>
      <UserInfo name="Alice" email="alice@example.com" />
      <UserInfo name="Bob" email="bob@example.com" />
    </div>
  );
}
```

PASSING OBJECTS AS PROPS

```jsx
function UserCard(props) {
  return (
    <div>
      <h1>{props.user.name}</h1>
      <p>{props.user.email}</p>
    </div>
  );
}
```

```jsx
function App() {
  const user = {
    name: 'Alice',
    email: 'alice@example.com' };
  return (
    <div>
      <UserCard user={user} />
    </div>
  );
}
```

PASSING FUNCTION AS PROPS

```jsx
function Button(props) {
  return <button onClick={props.onClick}>{props.label}</button>;
}
```

```jsx
function App() {
  const handleClick = () => {
    alert('Button clicked!');
  };
  return (
    <div>
      <Button onClick={handleClick} label="Click Me" />
    </div>
  );
}
```

DEFAULT PROPS

```
function Greeting(props) {
    return <h1>Hello, {props.name}</h1>;
  }

  Greeting.defaultProps = {
    name: 'Guest'
  };
export default Greeting;
```

```
function App() {
  return (
    <div>
      <Greeting />
      <Greeting name="Alice" />
    </div>
  );
}
```

PROPTYPES

```jsx
function Profile(props) {
  return (
    <div>
      <h1>{props.name}</h1>
      <p>Age: {props.age}</p>
    </div>
  );
}
Profile.propTypes = {
  name: PropTypes.string.isRequired,
  age: PropTypes.number.isRequired
};
```

```jsx
function App() {
  return (
    <div>
      <Profile name="Alice" age={25} />
    </div>
  );
}
```

CONDITIONAL RENDERING BASED ON PROPS

```
function Status(props) {
  if (props.isLoggedIn) {
    return <h1>Welcome back!</h1>;
  } else {
    return <h1>Please sign up.</h1>;
  }
}
```

```
function App() {
  return (
    <div>
      <Status isLoggedIn={true} />
      <Status isLoggedIn={false} />
    </div>
  );
}
```

UPDATE PARENT STATE FROM CHILD

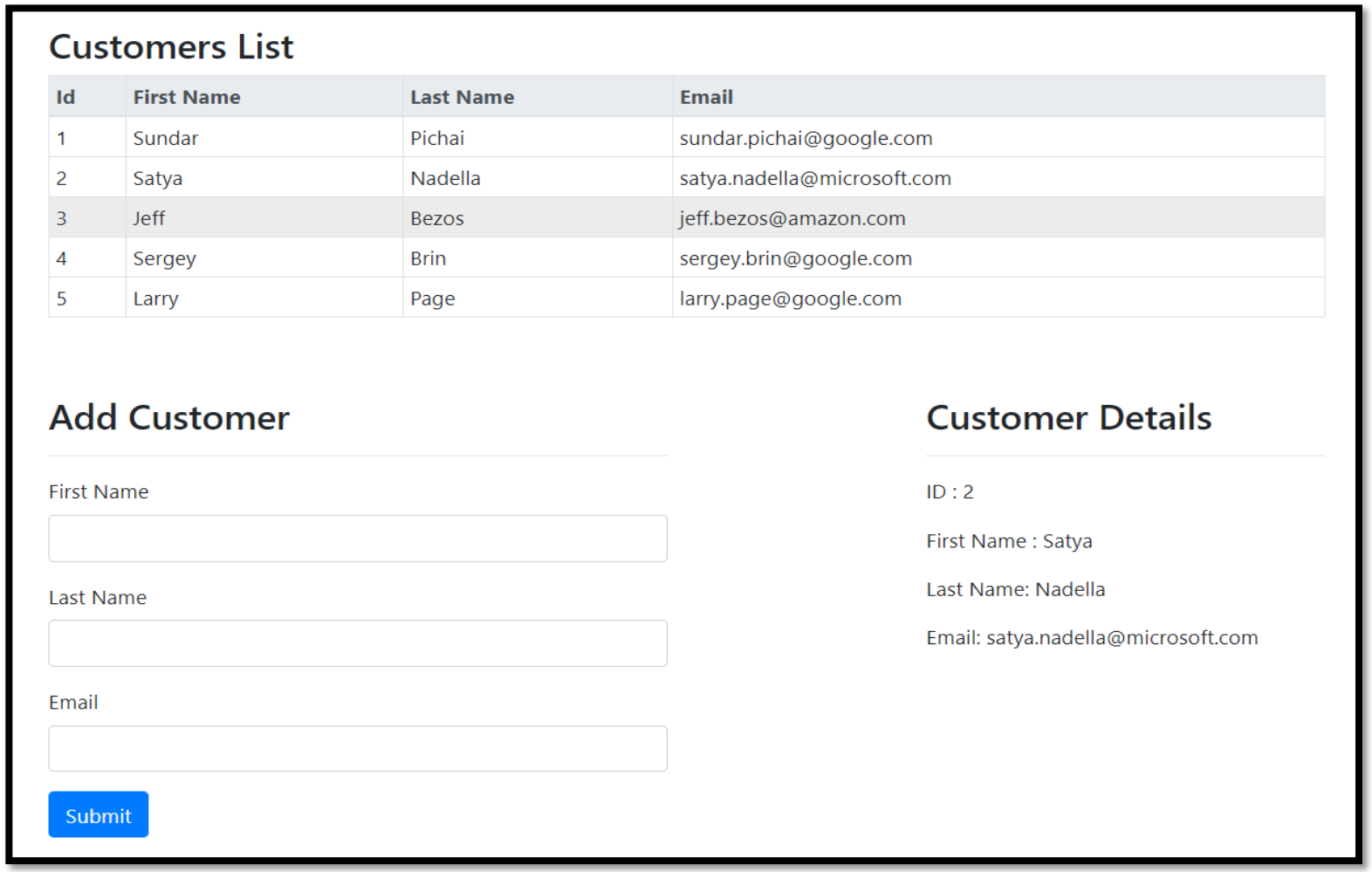
- Create Parent Component
- Declare one state inside the parent component
- Pass the state value as well as function as props to child component
- Now trigger the update set function from props and check the parent state updated or not

```
function App() {
  const [parentState, setParentState] = useState('Initial State');

  const updateParentState = (newState) => {
    setParentState(newState);
  };

  return (
    <div>
      <h1>{parentState}</h1>
      <ChildComponent updateParentState={updateParentState} />
    </div>
  );
}
```

CHILD COMPONENT

```
function ChildComponent(props) {
  return (
    <div>
      <button onClick={() => props.updateParentState('Updated from child')}>
        Update Parent State
      </button>
    </div>
  );
}
```

ACTIVITY: STATE PROPS



DESCRIPTION

- Create 3 components
- CustomerList (parent)
  - addCustomer
  - viewCustomer
- Pass some props from parent to child for interaction
- addCustomer will pass object of customer from child to parent
- Viewcustomer will take customer details to display from parent to child

USE EFFECT HOOK

- The useEffect hook in React allows you to perform side effects in function components.
- It serves as a combination of lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount in class components

```
function Counter() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

LIFECYCLE USING USEEFFECT()

```
function WelcomeMessage() {
  useEffect(() => {
    console.log('Component did mount');

    return () => {
      console.log('Component will unmount');
    };
  }, []);

  return <h1>Welcome!</h1>;
}

export default WelcomeMessage;
```

FETCH DATA

```
function FetchData() {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/posts/1')
      .then(response => response.json())
      .then(data => setData(data));
  }, []);

  return (
    <div>
      {data ? <p>{data.title}</p> : <p>Loading...</p>}
    </div>
  );
}
```

LISTENING TO EVENTS

```
function WindowWidth() {
  const [width, setWidth] = useState(window.innerWidth);

  useEffect(() => {
    const handleResize = () => setWidth(window.innerWidth);
    window.addEventListener('resize', handleResize);

    return () => {
      window.removeEventListener('resize', handleResize);
    };
  }, []);

  return <p>Window width: {width}px</p>;
}
```

TIMER EVENTS

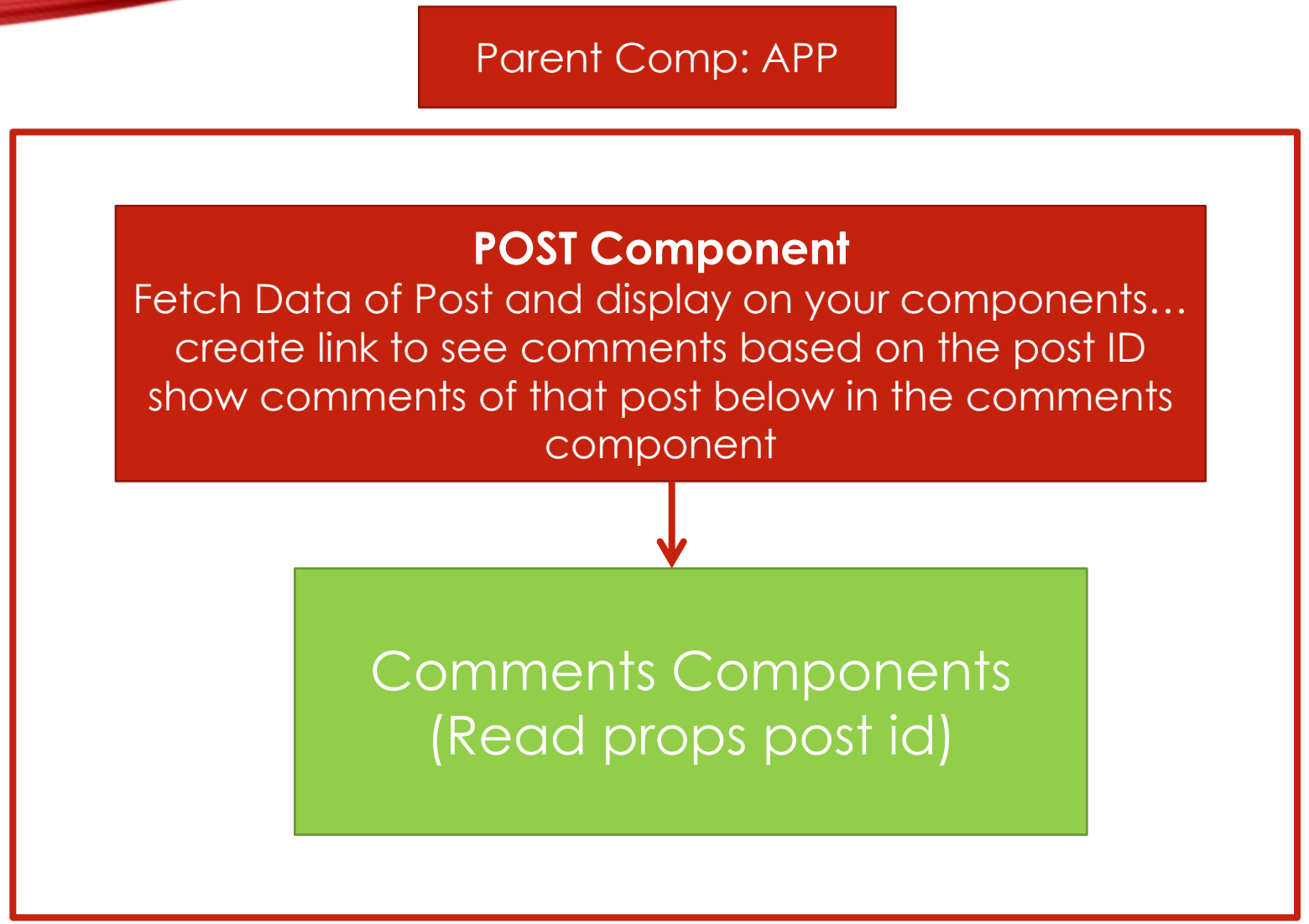
```
function Timer() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds(prevSeconds => prevSeconds + 1);
    }, 1000);

    return () => clearInterval(interval);
  }, []);

  return <p>Seconds: {seconds}</p>;
}
```

DEPENDENCY ARRAY IN USEEFFECT

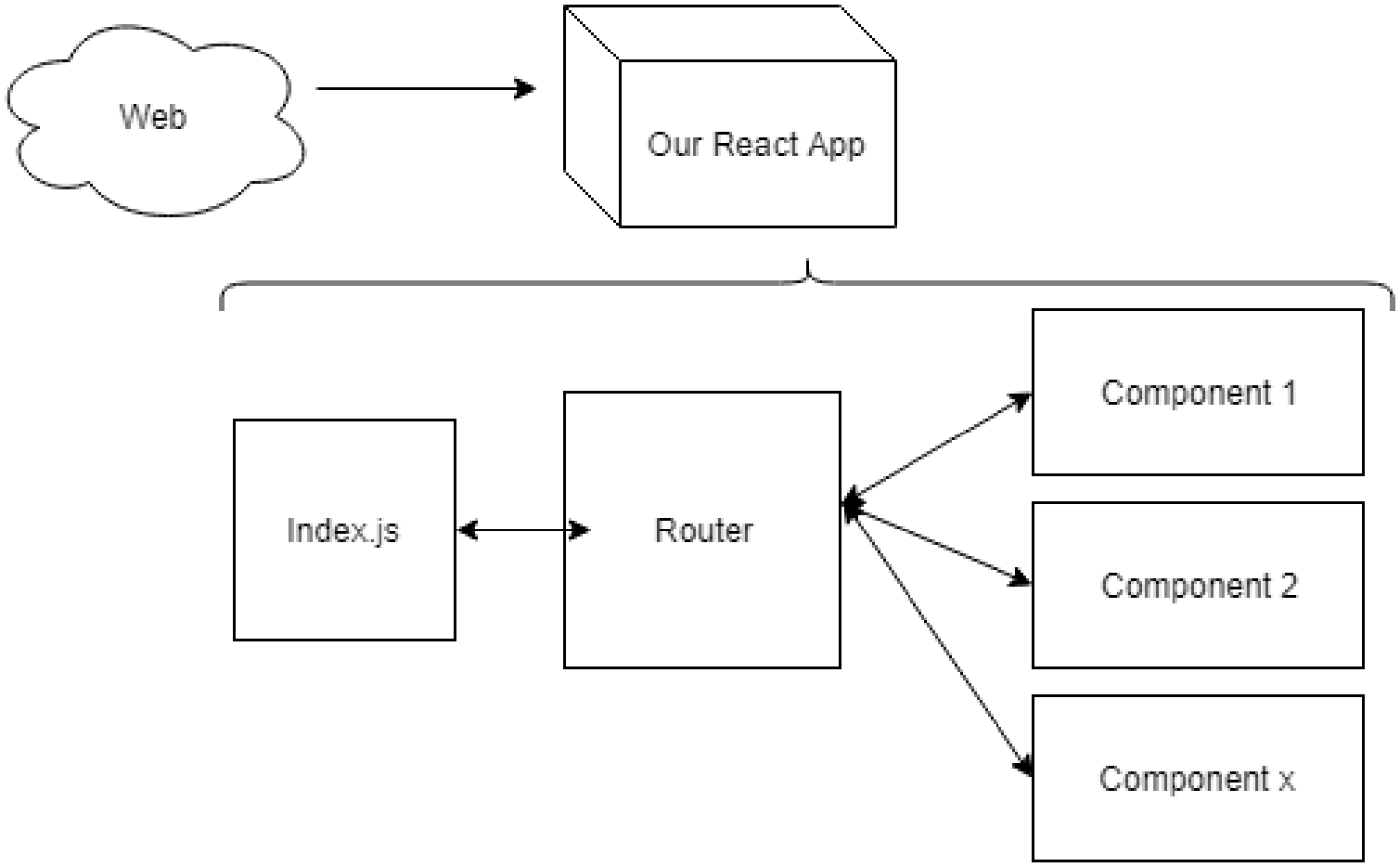
- By Default useEffect hooks run everytime when any state changes of the component.
- If you want to run use effect to run only once then use [] (empty array) as dependency.
- If you want to execute useeffect incase of a perticular state chnages of the component then pass that state name to the dependency array.

```
function Incrementer() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log(`Count is: ${count}`);
  }, [count]);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>
        Increment
      </button>
    </div>
  );
}
```

ACTIVITY


Parent Comp: APP
POST Component
Fetch Data of Post and display on your components…
create link to see comments based on the post ID
show comments of that post below in the comments component
Comments Components
(Read props post id)

REACT ROUTER



HOW TO SETUP?

- Install dependency: npm install react-router-dom

```
function App() {
  return (
    <Router>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </Router>
  );
}
```

Browser Router

Root tag Routes→ add Route

NESTED ROUTES

```
function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="about" element={<About />} />
        <Route path="dashboard/*" element={<Dashboard />} />
      </Routes>
    </Router>
  );
}
```

NESTED ROUTES

```
function Dashboard() {
  return (
    <div>
      <h2>Dashboard</h2>
      <nav>
        <Link to="profile">Profile</Link>
        <Link to="settings">Settings</Link>
      </nav>
      <Routes>
        <Route path="profile" element={<Profile />} />
        <Route path="settings" element={<Settings />} />
      </Routes>
    </div>
  );
}
```

DYNAMIC ROUTES

```
function App() {
  return (
    <Router>
      <nav>
        <Link to="/user/1">User 1</Link>
        <Link to="/user/2">User 2</Link>
      </nav>
      <Routes>
        <Route path="user/:userId" element={<User />} />
      </Routes>
    </Router>
  );
}
```

```
function User() {
  let { userId } = useParams();
  return <h2>User ID: {userId}</h2>;
}
```

PROTECTED ROUTE

```
function App() {
  const isAuthenticated = false;
  return (
    <Router>
      <Routes>
        <Route path="/login" element={<Login />} />
        <Route
          path="/dashboard"
          element={
            <ProtectedRoute isAuthenticated={isAuthenticated}>
              <Dashboard />
            </ProtectedRoute>
          }
        />
      </Routes>
    </Router>
  );
}
```

PROTECTED ROUTE

```
function ProtectedRoute({ isAuthenticated, children }) {
  return isAuthenticated ? children : <Navigate to="/login" />;
}

function Login() {
  return <h2>Login Page</h2>;
}

function Dashboard() {
  return <h2>Dashboard</h2>;
}
```

PROGRAMMATIC NAVIGATION

```
function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </Router>
  );
}
```

```
function Home() {
  const navigate = useNavigate();
  const goToAbout = () => {
    navigate('/about');
  };
  return (
    <div>
      <h2>Home</h2>
      <button onClick={goToAbout}>
        Go to About</button>
    </div>
  );
}
function About() {
  return <h2>About</h2>;
}
```

HANDLING 404 ERROR

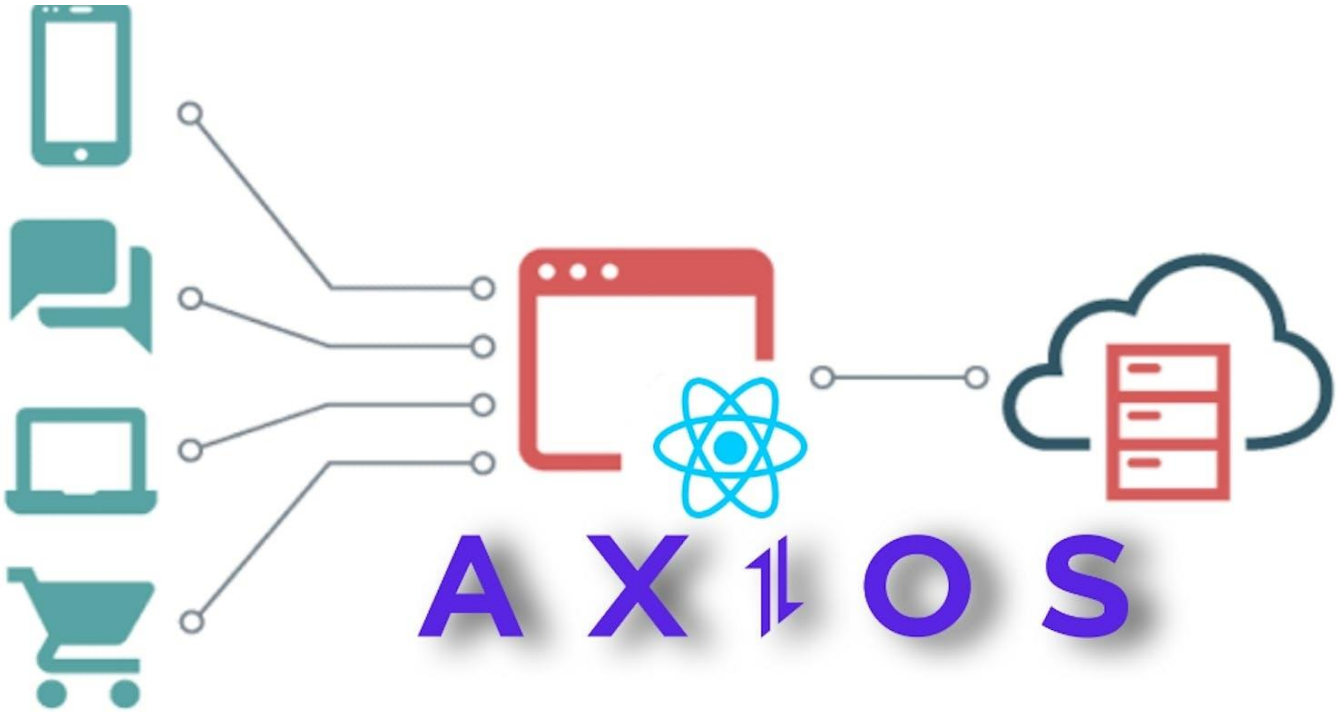
```
function App() {
  return (
    <Router>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
        <Link to="/non-existent-page">Non-existent Page</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="*" element={<NotFound />} />
      </Routes>
    </Router>
  );
}
```

AXIOS

Axios is a popular JavaScript library for making HTTP requests.

It can be used with both JavaScript and TypeScript, and it works in both browser and Node.js environments.

Using axios we can trigger post, get, put & delete method directly.



GET DATA USING AXIOS

```
function FetchData() {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/posts/1')
      .then(response => {
        setData(response.data);
        setLoading(false);
      })
      .catch(error => {
        setError(error);
        setLoading(false);
      });
  }, []);
```

```
  if (loading) return
  <p>Loading...</p>;
  if (error) return
  <p>Error: {error.message}</p>;

  return (
    <div>
      <h2>{data.title}</h2>
      <p>{data.body}</p>
    </div>
  );
}
```

POST DATA

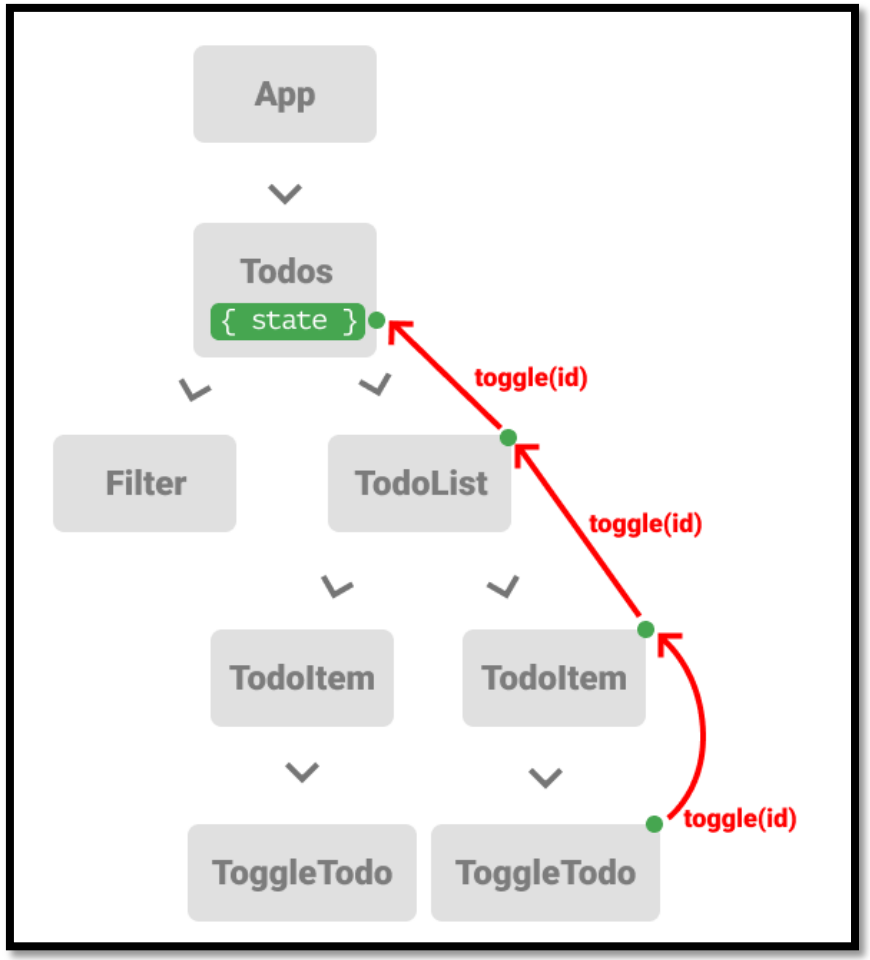
```
const handleSubmit = async (event) => {
  event.preventDefault();
  try {
    const result = await axios.post('BASE_URL', {
      title,
      body
    });
    setResponse(result.data);
  } catch (error) {
    console.error('Error creating post:', error);
  }
};
```

ACTIVITY: CRUD OPERATION

- Let’s create one dummy JSON server
- Create React application
- Trigger methods
  - Get
  - Post
  - Put
  - delete

STATE MANAGEMENT

- Usually when we declare a state in a component it becomes local state
- So if you want to use it to another component then you have to pass as a props.
- If there are many children then also we have to pass using props which is called propdrilling.



ACTIVITY

- Implement propdrilling using props
- Create 3 components
  - GrandParent
  - Parent
  - Child
- Access Grand Parent property to Child component

CONTEXT API

```
const CountContext = createContext();
function CounterProvider({ children }) {
  const [count, setCount] = useState(0);
  return (
    <CountContext.Provider value={{ count, setCount }}>
      {children}
    </CountContext.Provider>
  );
}
```

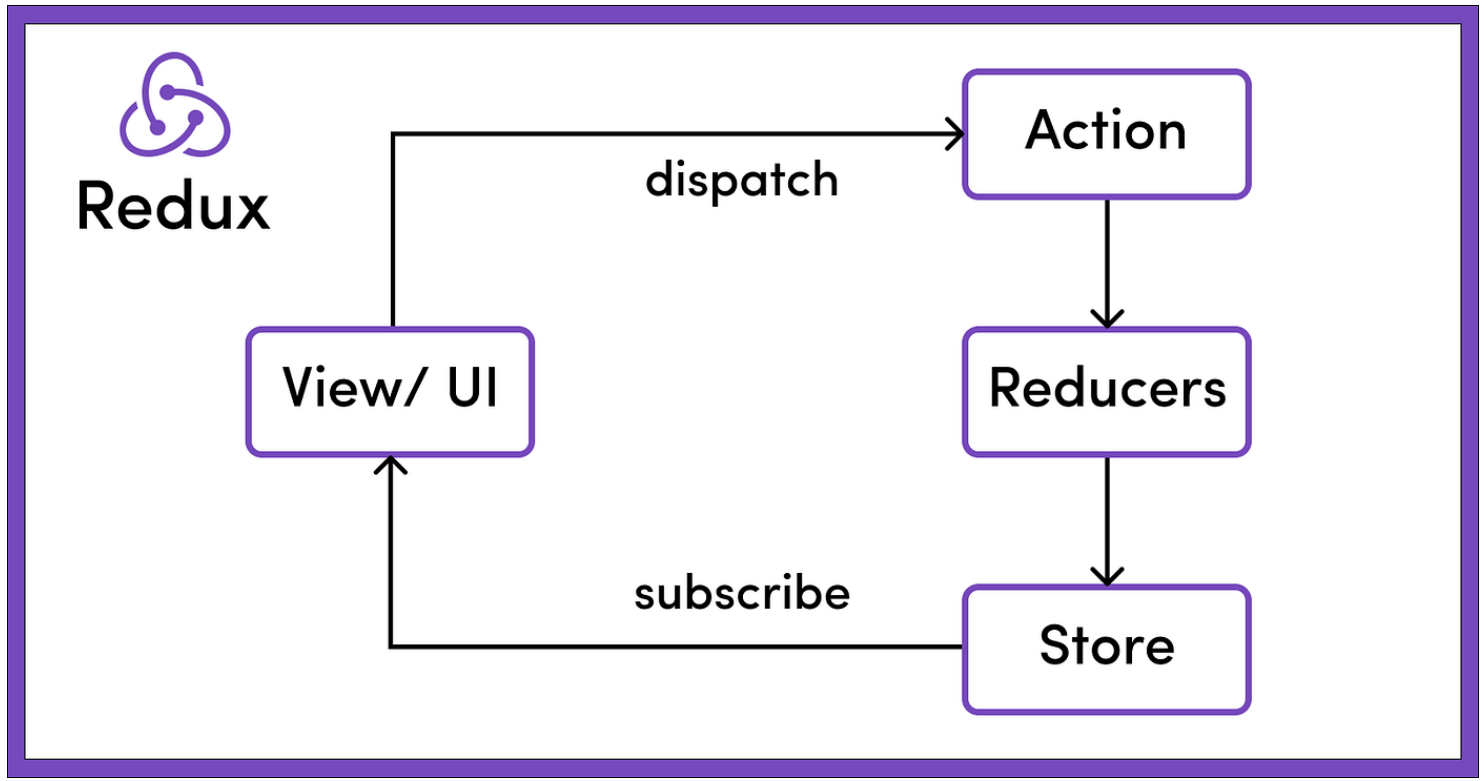
CONTEXT API

```
function CounterDisplay() {
  const { count } = useContext(CountContext);
  return <p>Count: {count}</p>;
}
function CounterControls() {
  const { setCount } = useContext(CountContext);
  return (
    <div>
      <button onClick={() => setCount(prevCount => prevCount + 1)}>
        Increment</button>
      <button onClick={() => setCount(prevCount => prevCount - 1)}>
        Decrement</button>
    </div>
  );
}
```

```
function App() {
  return (
    <CounterProvider>
      <CounterDisplay />
      <CounterControls />
    </CounterProvider>
  );
}
```

REDUX

- Third party library for implement using redux
- Install: npm install react-redux redux



REDUX IMPLEMENTATION

- Create Store: store.js
- Register in index.js: using <provider> tag attach store here
- Create reducer: its functions which is used to update the global state.

```
import { createStore } from 'redux';

const initialState = { count: 0 };

function counterReducer(state = initialState, action) {
  switch (action.type) {
    case 'INCREMENT':
      return { count: state.count + 1 };
    case 'DECREMENT':
      return { count: state.count - 1 };
    default:
      return state;
  }
}

const store = createStore(counterReducer);

export default store;
```

REDUX

```
function Counter() {
  const count = useSelector(state => state.count);
  const dispatch = useDispatch();
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => dispatch({ type: 'INCREMENT' })}>
        Increment</button>
      <button onClick={() => dispatch({ type: 'DECREMENT' })}>
        Decrement</button>
    </div>
  );
}
```

```
function App() {
  return (
    <Provider store={store}>
      <Counter />
    </Provider>
  );
}
```

IMPLEMENT REDUX BY SEPARATING FILES

- Create store.js
- Create reducer.js
- Register reducer in root reducer
- Include store in index.js
- Read state in Component
- Update State in component

ACTIVITY: CART MANAGEMENT

- Implement frontend state management for cart implementation
- Write cart reducer:
- Add item to cart
- Remove item from cart
- Show all carts