**React usage:**

1. React is a JavaScript library for building user interfaces.
2. React is used to build single-page applications.
3. React allows us to create reusable UI components.

**Create react project:**

**npx create-react-app my-react-app**

**Run install:**

**npm start**

**createRoot Function:**

* The createRoot() function takes one argument, an HTML element.
* The purpose of the function is to define the HTML element where a React component should be displayed.

**The render Method:**

The render() method is then called to define the React component that should be rendered.

**What is JSX?**

1. JSX stands for JavaScript XML.
2. JSX allows us to write HTML in React.
3. JSX makes it easier to write and add HTML in React.
4. JSX converts HTML tags into react elements.

**React Components**

Components are independent and reusable bits of code

Props

* Components can be passed as props, which stands for properties.
* Props are like function arguments, and you send them into the component as attributes.
* Props are passed to components via HTML attributes.

**React Events:**

1. React events are written in camelCase syntax:
2. onClick instead of onclick.
3. React event handlers are written inside curly braces:
4. onClick={shoot} instead of onClick="shoot()".

**Passing Arguments:**

To pass an argument to an event handler, use an arrow function.

<button onClick={() => shoot("Goal!")}>Take the shot!</button>

**React Event Object:**

<button onClick={(event) => shoot("Goal!", event)}>Take the shot!</button>

**React Conditional Rendering:**

function Goal(props) {

const isGoal = props.isGoal;

if (isGoal) {

return <MadeGoal/>;

}

return <MissedGoal/>;}

**React Lists**

The JavaScript map() array method is generally the preferred method.

function Garage() {

const cars = ['Ford', 'BMW', 'Audi'];

return (

<>

<h1>Who lives in my garage?</h1>

<ul>

{cars.map((car) => <Car brand={car} />)}

</ul>

</>

);}

**React Memo:**

Using memo will cause React to skip rendering a component if its props have not changed.

LifeCycle hooks:

**useState:**

* The React useState Hook allows us to track state in a function component.
* useState accepts an initial state and returns two values:
* The current state.
* A function that updates the state.

Ex: const [color, setColor] = useState("");

**useEffect:**

Notes: Componentdidmount equal to this.

* The useEffect Hook allows you to perform side effects in your components.
* Some examples of side effects are: fetching data, directly updating the DOM, and timers.
* useEffect accepts two arguments. The second argument is optional.
* useEffect(<function>, <dependency>)

**No dependency passed:**

useEffect(() => {

//Runs on every render

});

An empty array:

useEffect(() => {

//Runs only on the first render

}, []);

Props or state values:

useEffect(() => {

//Runs on the first render

//And any time any dependency value changes}, [prop, state]);

Effect Cleanup:

* Some effects require cleanup to reduce memory leaks.
* Timeouts, subscriptions, event listeners, and other effects that are no longer needed should be disposed.
* We do this by including a return function at the end of the useEffect Hook.
* React’s useEffect cleanup function saves applications from unwanted behaviors like memory leaks by cleaning up effects. In doing so, we can optimize our application’s performance.
* The useEffect Hook is built in a way that we can return a function inside it and this return function is where the cleanup happens. The cleanup function prevents memory leaks and removes some unnecessary and unwanted behaviors.

useEffect(() => {

effect

return () => {

cleanup

}

}, [input])

return () => clearTimeout(timer)

}, []);

**useCallback**:

* The React useCallback Hook returns a memoized callback function.
* This allows us to isolate resource intensive functions so that they will not automatically run on every render.
* The useCallback Hook only runs when one of its dependencies update.
* This can improve performance.

**The useCallback and useMemo Hooks are similar. The main difference is that useMemo returns a memoized value and useCallback returns a memoized function.**

**useMemo:**

* The useMemo Hook only runs when one of its dependencies update.

**Reducer:**

* The reduce() method executes a reducer function for array element.
* The reduce() method returns a single value: the function's accumulated result.
* The reduce() method does not execute the function for empty array elements.
* The reduce() method does not change the original array.

what’s a reducer?

* A reducer is a function that takes multiple things and returns one thing.

Return Value?

The accumulated result from the last call of the callback function.

*Components with many state updates spread across many event handlers can get overwhelming. For these cases, you can consolidate all the state update logic outside your component in a single function, called a reducer.*

**useReducer:**

* The useReducer hook is an alternative to useState.
* It’s preferable to useState when you have more complex.
* State logic involving serveral related values or when the next state depends on the prev state.
* The useReducer Hook returns the current state and a dispatch method.

**const [state, dispatch] = useReducer(reducer, init)**

**React Custom Hooks:**

* Hooks are reusable functions.
* When you have component logic that needs to be used by multiple components, we can extract that logic to a custom Hook.
* Custom Hooks start with "use". Example: useFetch.