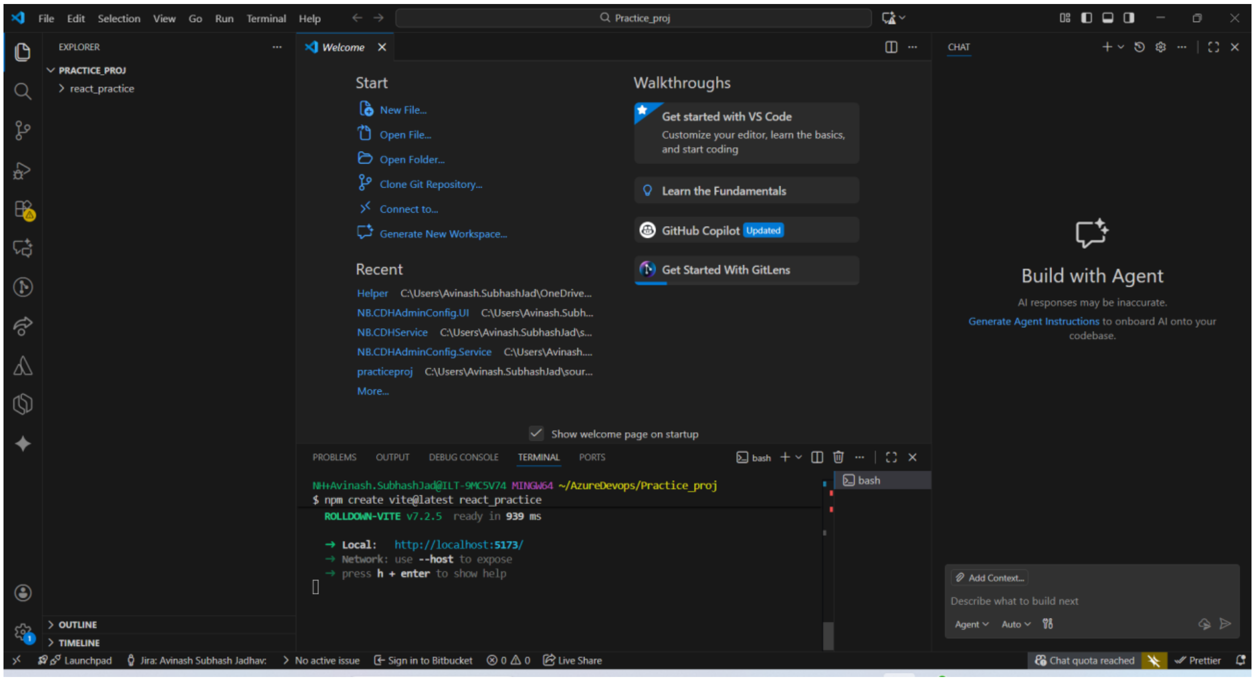
Day1:


File Edit Selection View Go Run Terminal Help
Practice_proj
EXPLORER
PRACTICE_PROJ
react_practice
Welcome
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Connect to...
Generate New Workspace...
Recent
Helper C:\Users\Avinash.SubhashJad\OneDrive...
NB.CDHAdminConfig.UI C:\Users\Avinash.Subh...
NB.CDHService C:\Users\Avinash.SubhashJad\s...
NB.CDHAdminConfig.Service C:\Users\Avinash....
practiceproj C:\Users\Avinash.SubhashJad\sour...
More...
Walkthroughs
Get started with VS Code
Customize your editor, learn the basics, and start coding
Learn the Fundamentals
GitHub Copilot Updated
Get Started With GitLens
Show welcome page on startup
CHAT
Build with Agent
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.
Add Context...
Describe what to build next
Agent
Auto
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash
OUTLINE
TIMELINE
Launchpad
Jira: Avinash Subhash Jadhav:
No active issue
Sign in to Bitbucket
0 0
Live Share
Chat quota reached
Prettier

```
NH+Avinash.SubhashJad@ILT-9MC5V74 MINGW64 ~/AzureDevops/Practice_proj
$ npm create vite@latest react_practice
  ROLLDOWN-VITE v7.2.5  ready in 939 ms

  → Local:   http://localhost:5173/
  → Network: use --host to expose
  → press h + enter to show help
```

Virtual Dom:

In-Memory representation of the Real DOM that react uses to optimize UI updates. Instead of direct UI updates. Instead of directly manipulating the Browser's DOM – which is slow – React updates this virtual copy first, then efficiently syncs only the necessary changes to the real DOM.

How it works:

1. Initial Render: when a React app starts, React builds a Virtual DOM tree from your components.

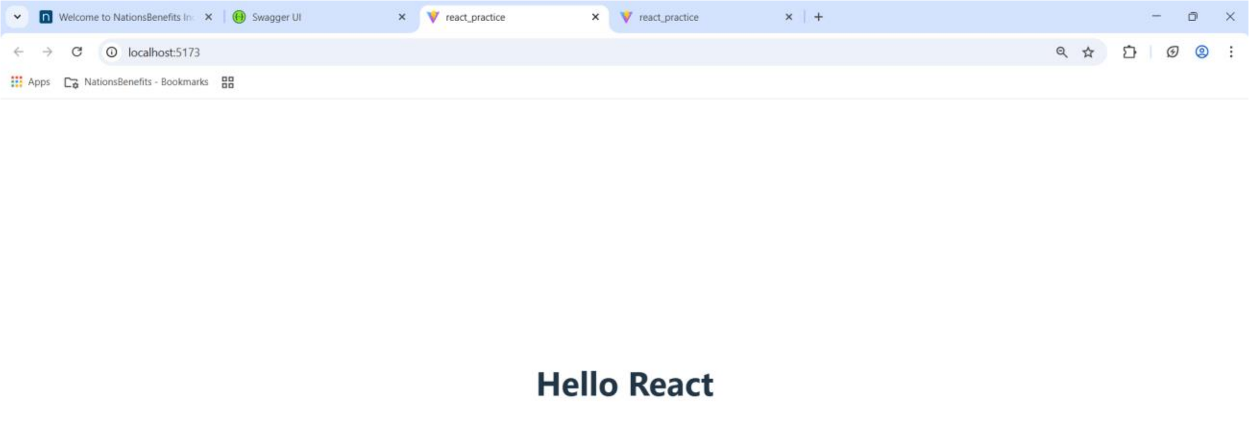
2. State/Props Changes: Any changes in state or props triggers React to create a new Virtual DOM tree.

3. Diffing Algorithm: React Compares the new virtual DOM with the previous one to find the minimal set of changes.

4. Reconciliation: Only the changed nodes are updated in the real DOM, avoiding full re-renders.

5. DOM Update: The Real DOM is patched with these changes, resulting in faster and smoother UI updates.

JSX syntax and Rules:



Hello React

What is JSX: JSX is a syntax extension for JavaScript that allows you to write HTML- Like code inside javascript.

It is not HTML, but it gets compiled into React.createElement() calls.

Example :

Const element = <h1> Hello React <h1>;

1. Return a single Parent Element

JSX must have one root element.

Wrap multiple elements in a parent tag like <div> or <> (React Fragment).

2. Close All tags .
   All elements should be properly closed, including self-closing tags.
3. Use className Instead of class
   In Jsx class is a reversed Javascript keyword, so use className.
4. Javascript Expression in {}
   You can embed Javascript expressions inside {}.
5. Attributes in CamelCase

<button onClick={**handleClick**}> Click Me </button>

6. Inline Style as Objects:
   Inline styles are written as JavaScript objects with camelCase Property names.
   Example:
   const styleObj = {color:”red”, fontSize:”20px”};
   Return <p style={styleObj}/> Styled paragraph </p>;
7. JSX must be wrapped in Parentheses (Optional but Recommended)
8. Conditinal Rendering :
   we can use ternary operators or logical && inside JSX.
   {isLoggedIn ? <p> Welcome! </p>: <p>Please log in.</p>}
   {count >0 && <p> You have {count} messages. </p>}
9. Fragments for Grouping:
   Use <>… </> or <React.Fragment>…</React.Fragment>

Day2 :

Functional Component is A Javascript Function that returns JSX.

```
function Hello(){

return <h1>Hello, world!</h1>;

}
```

This hello function is a functional component that returns the UI.

It's a function, so it takes props as input, returns JSX.
Stateless (originally), but since React 16.8, we can use hooks (like useState, useEffect) to manage state and side effects.

Example with props:

```
function Greeting({name}) {

return <p>Hello , {name}</p>

}
```

Usage: <Greeting name=”Avinash”>

Arrow Function expressions:

```
const Greeting =>(props){

return <h1>Hellow, {props.name}!</h1>;

}
```

Arrow function with implicit returns:

If our component just returns JSX (no extra logic), we can make it a one-liner.

```
const Greeting =({props})=><h1>Hello, {props.name}!</h1>;
```

Destructuring props directly in the parameter:

Instead of doing props.name we can unpack props in the signature.

```
function Greeting({name,age}){

return <p> {name} is {age} years old</p>;

}
```

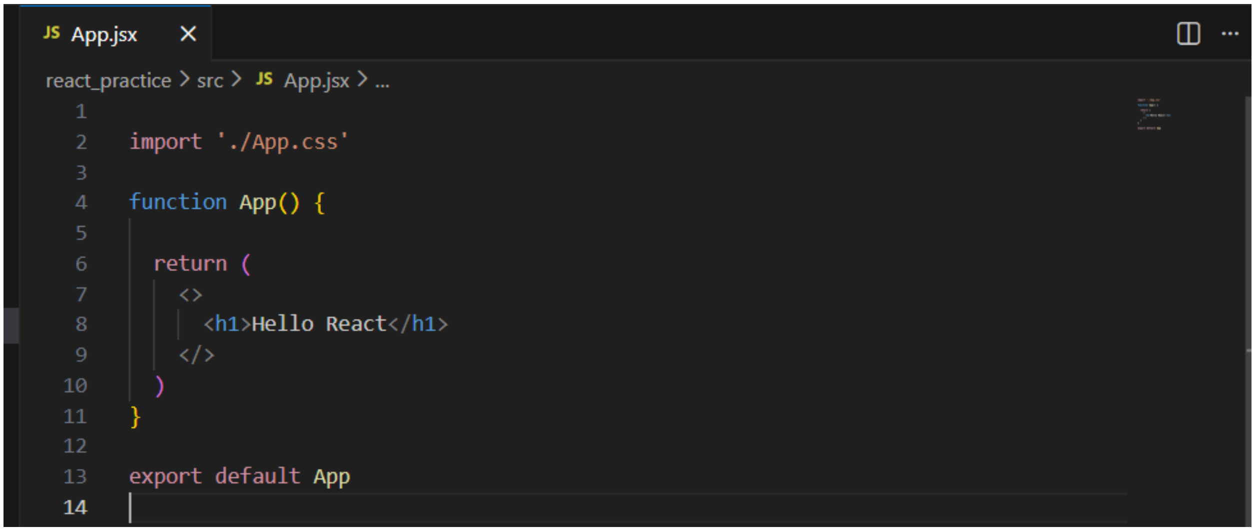
With React.FC (Typescript only):

we can define component types using React.FC (or React.FunctionComponent):

```
import React from 'react';

const Greeting : React.FC<{name:string}>=({name})=>{

return <h1>Hello, {name}!</h1>;

}
```

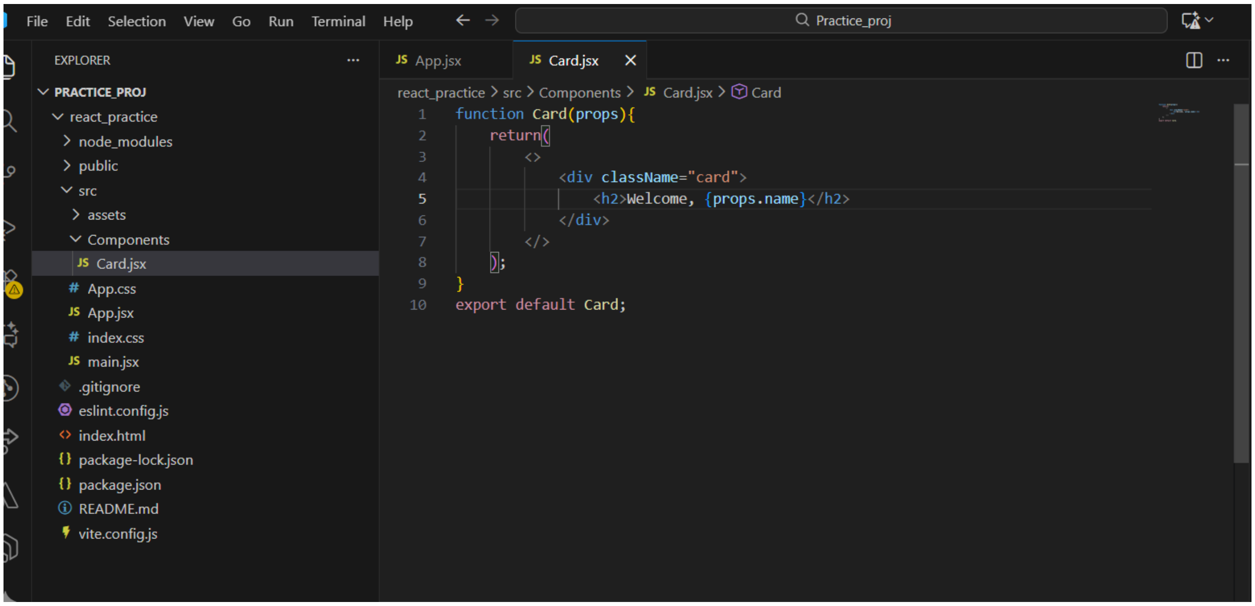
Order Matters:

imports – Function – JSX return – Export

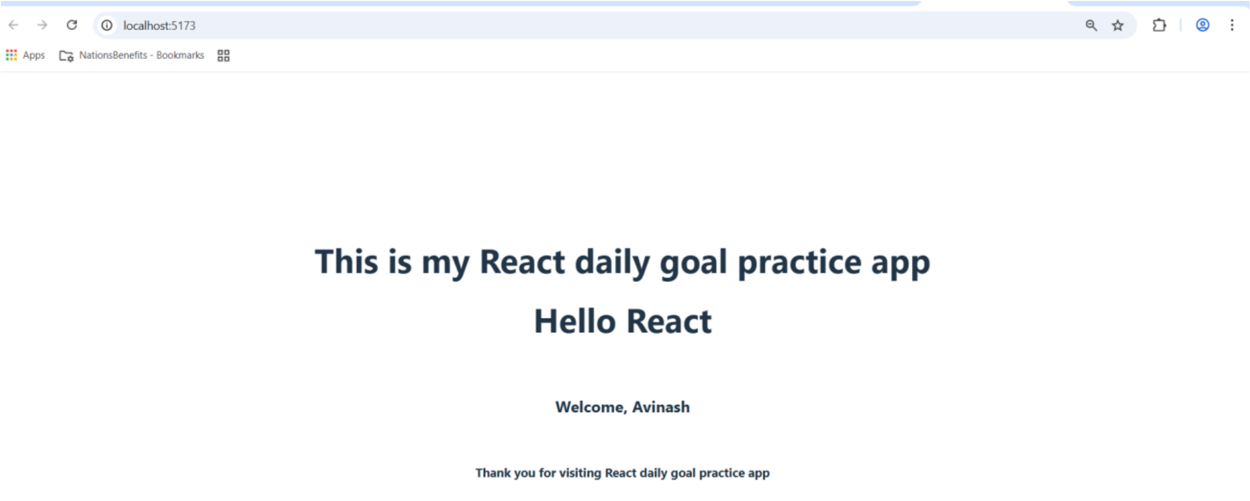


```jsx

import './App.css'

function App() {

  return (
    <>
      <h1>Hello React</h1>
    </>
  )
}

export default App

```

Reausable Component (Card):



```
function Card(props){
    return(
        <>
            <div className="card">
                <h2>Welcome, {props.name}</h2>
            </div>
        </>
    );
}
export default Card;
```

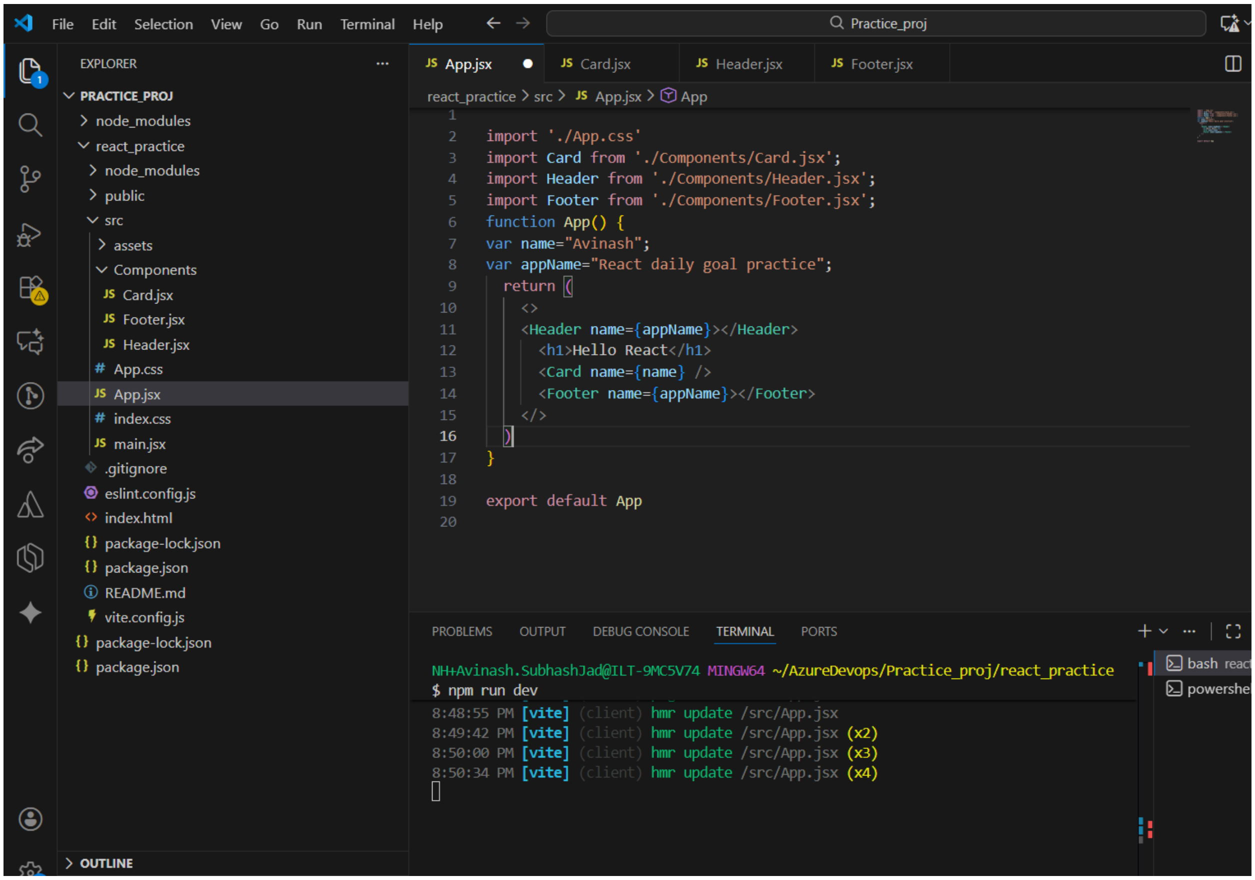


This is my React daily goal practice app

Hello React

Welcome, Avinash

Thank you for visiting React daily goal practice app



```jsx

import './App.css'
import Card from './Components/Card.jsx';
import Header from './Components/Header.jsx';
import Footer from './Components/Footer.jsx';
function App() {
var name="Avinash";
var appName="React daily goal practice";
  return (
    <>
    <Header name={appName}></Header>
      <h1>Hello React</h1>
      <Card name={name} />
      <Footer name={appName}></Footer>
    </>
  )
}

export default App

```

```
NH+Avinash.SubhashJad@ILT-9MC5V74 MINGW64 ~/AzureDevops/Practice_proj/react_practice
$ npm run dev
8:48:55 PM [vite] (client) hmr update /src/App.jsx
8:49:42 PM [vite] (client) hmr update /src/App.jsx (x2)
8:50:00 PM [vite] (client) hmr update /src/App.jsx (x3)
8:50:34 PM [vite] (client) hmr update /src/App.jsx (x4)
```

Default Export vs Named Export:

default:

1. Only one default per file.
2. Name can change during import.

Named Export:

multiple exports allowed.

Name must match.

```
export function Footer () {

return <p>Footer</p>;

}
```

```
import {Footer} from './Components/Footer.jsx'
```

Day3:

What is state:

State is data owned by a component that can change over time.

When state changes, React re-renders the UI.

Normal variable does not update UI.

state variable does update UI.

React does not watch variables.
React only watches state and props.

UseState :

Hook means:

A special function.
Let's functional components use React features.
Introduced because of functions earlier had no state.

Before Hooks:

Only class components had state.
After hooks:
Functional components do everything.

Why useState:
beacuse: order matters, destructuring is easy, hook internally tracks state by position.

React internally remembers : “First useState ---- this value”

That’s why hooks must be:
at top level
Not inside loops or conditions.

how state update works:
setState(newValue)

React does not immediately change UI.

React does :
1. Schedule update
2. Compares old virual DOM with new Virtual DOM.
3. Finds minimal changes.
4. Updates real DOM.

what is Rerender:

re-render means:
Component function runs again.
JSX is re-evaluated.
UI updates if needed.

re-render does not means:
page reload.
DOM destroyed fully.

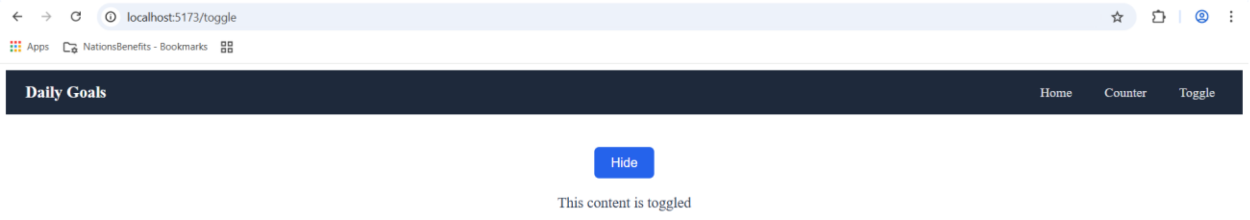
Controlled Components:

in react.js managing form inputs and user interactions is a crucial part of building dynamic web applications.

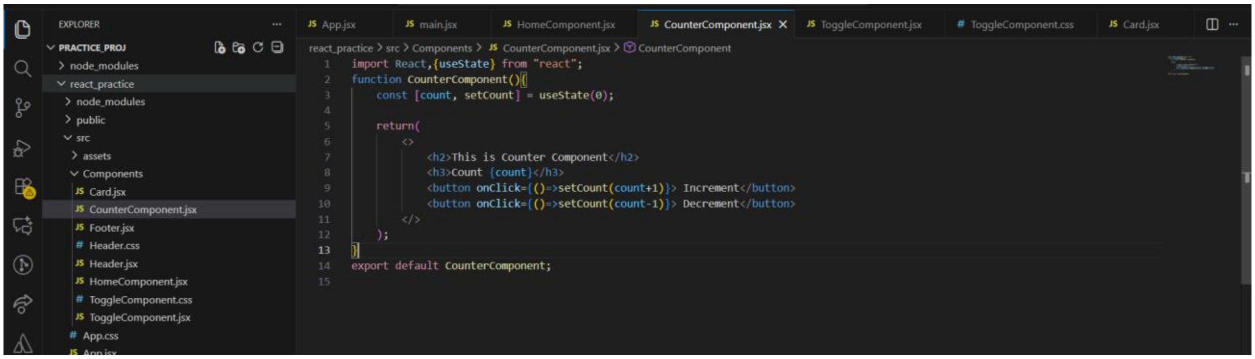
two key concepts that developers need to understand are controlled and uncontrolled components. These concepts define how data is handled withing a react component.

Controlled components are form elements (like input, textarea, or select) that are managed by React state. This means that value of the form element is set and updated through React state,
Making React the "Single Source of truth" for the data.

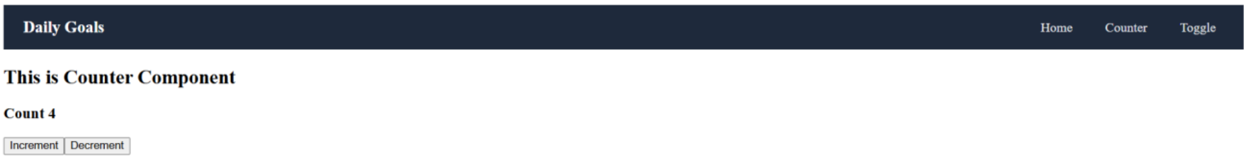
By Controlling form elements via state, you gain more control over user interactions and can easily enforce validation, format data, and respond to changes.



This content is toggled



```
import React,{useState} from "react";
function CounterComponent(){
    const [count, setCount] = useState(0);

    return(
        <>
            <h2>This is Counter Component</h2>
            <h3>Count {count}</h3>
            <button onClick={()=>setCount(count+1)}> Increment</button>
            <button onClick={()=>setCount(count-1)}> Decrement</button>
        </>
    );
}
export default CounterComponent;
```



This is Counter Component

Count 4

```
import React, { useState } from 'react';

function ControlledComponent() {

  const [value, setValue] = useState('');

  const handleChange = (event) => {

    setValue(event.target.value);

  };

  const handleSubmit = (event) => {

    event.preventDefault();

    alert('A name was submitted: ' + value);

  };

  return (

    <form onSubmit={handleSubmit}>

      <label>

        Name:

        <input type="text" value={value} onChange={handleChange} />

      </label>

      <button type="submit">Submit</button>

    </form>
```

In this example:

The value state holds the current value of the input field.

The handleChange function updates the state whenever the user types in the input field.

The handleSubmit function handles the form submission, using the current state value.

day 4:

What is Event Handling?

Event handling lets your React components respond to user actions like clicks, typing, form submission, etc.

React events are:
written in camelCase (onClick, onChange).
Passed as functions, not strings.
Based on Synthetic Events (React wrapper over browser events).

React does not use browser events directly.
Instead it uses Synthetic Event:
A wrapper around the browser events.
Works the same across all browsers.
Improves performance.

Internally React listens to events at the root (event delegation).

Event Delegation (Behind the scenes)
React atteches one event listener at the root (#root), not every element.

why ?
Better Perfomance.
Less memory usage.
Faster event handling.

User Click --> Browser Event --> React root listener --> Synthetic evet-->Your Handler Function.

onClick:
used to handle button clicks or any clickable element.

Important points: Don't call the function directly.
onClick={handleClick()} // this is wrong

Always pass a reference:
onClick={handleClick} // correct way

OnChange:

Used mainly with input, textarea, select to track user input.
React uses controlled components ---> input value is controlled by state.

onChange in React beahaves like onInput in HTML.
It Fires:

- On paste.
- On delete
- on every keystroke.

Handling Form Inputs (Controlled Components):

A controlled input means:
Value comes from state.
Updated using onChange.

```
import {useState} from "react";

function ControlledFormI(){

const [name,setName]=useState("");

return(

<input

type="text"

value={name}

onChange={(e)=>setName(e.target.value)}/>

);

}
```

Benefits:
Easy Evaluation.
Better control
Single Source of truth.

Prevent Default:
By Default, HTML forms refresh the page on submit.
PreventDefault() stops that behavior.

Event Handler function Rules:
must be a function.
Passed as reference.
Can be arrow or normal function.

Passing arguments to Event Handlers.

Never call the function directly

Arrow function(Most common):

<button onClick={handleDelete(id)}> Delete</button>

bind(less used):

<button onClick = {handleDelete.bind(this,id)}> Delete </button>

React topic Form Handling: by vaibhav

State and Hooks:

what is state ?

State is component-level memory used to store data that can change over time (like form input values).

In React, forms are handled using state so React always knows:

what the user typed.
When the value changed.
What to submit or reset.

useState Hook:

useState Hook allows functional components to hold and update state.

```
const [value, setValue]=useState(“”);
```

Never modify the state directly.
Always use the setter function.
Value=”abc” --- wrong
SetValue=("abc”);--correct.

Applying Events on Functions:

events connet uset actions to state updates.

common forms event:
OnChange: capture user input.
OnSubmit: Handle form submit.
OnClick: Button actions.

```
const handleChanges = (e)=>{
setName(e.target.value);
}
```

event FLow:

user types --> onChange fires--> state updates – UI re-renders.

Form Elements in React:

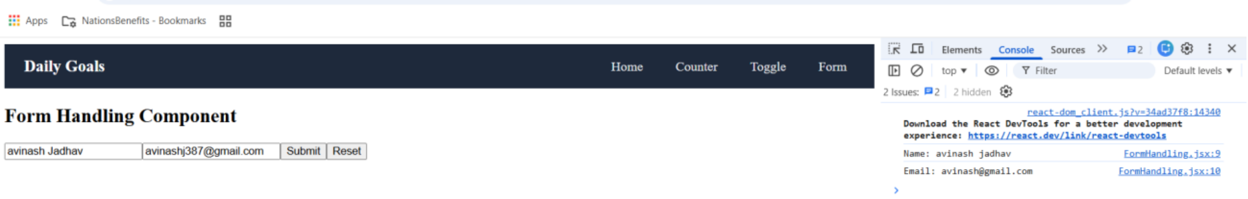
React supports all HTML form elements, but they are usually controlled by state.

```
<input
type=”text”
value={name}
onChange={(e)=>setName(e.target.value)}
/>

<textarea
value={message}
OnChange={(e)=>setMessage(e.target.value)} />
```

Select:

```
<select value={role} onChange={(e)=> setRole(e.target.value)>
<options vlaue =””> Select </options>
<options value = “admin”>Admin</options>
</select>
```

reset form :



Form Handling Component

avinash Jadhav | avinashj387@gmail.com | Submit | Reset

```
react-dom_client.js?v=34ad37f8:14340
Download the React DevTools for a better development
experience: https://react.dev/link/react-devtools
Name: avinash jadhav                    FormHandling.jsx:9
Email: avinash@gmail.com                FormHandling.jsx:10
```