Day-1

Wednesday, January 14, 2026 11:59 PM

Node -v install node js
Npm -v

Npm create vite@latest <projectname> -- Industry use -- default host server http://localhost:5173
Npx create-react-app <projectname> -- default host server http://localhost:3000

Cd<projectname>

Npm install (use to install node_module folder-- where all dependencies exists and it takes the configuration of your project from package.json)

Main.jsx / If project is created using vite@latest
If project is created using create-react-app you get index.js -- main program file from your App.jsx component runs through root

Index.css --- global css file the (styling will be applied on all the components)

Create a separate folder with the name as css and create a css file with the component name for which you want to apply css

Npm run dev -- to run your project you can also run via npm start but you need to add a script start in package.json

If project is created using create-react-app then directly you can give the command npm start to run

2) How to install tailwind css(Styling)
   UrL -- www.tailwindcss.com

a) On a Terminal to Install Tailwind CSS below is the command
   npm install tailwindcss @tailwindcss/vite

b) Configure the Vite plugin

   Add the @tailwindcss/vite plugin to your Vite configuration.

   Open vite.config.ts directly add tailwindcss() in plugin

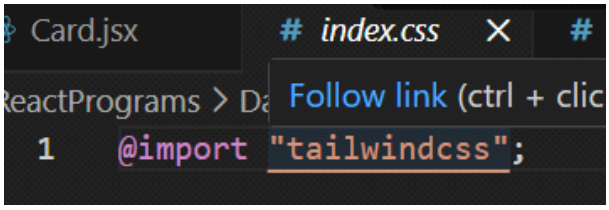
```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import tailwindcss from '@tailwindcss/vite'
// https://vite.dev/config/
export default defineConfig({
  plugins: [react(),tailwindcss()],
})
```

04

c) Import Tailwind CSS in your index.css file

Add an @import to your CSS file that imports Tailwind CSS.

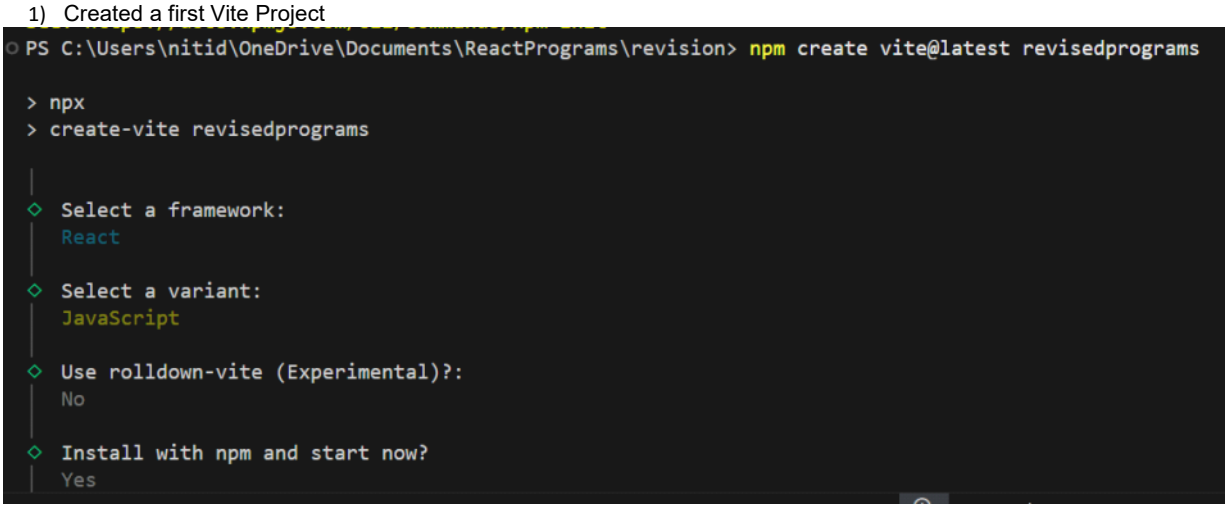
CSS
@import "tailwindcss";



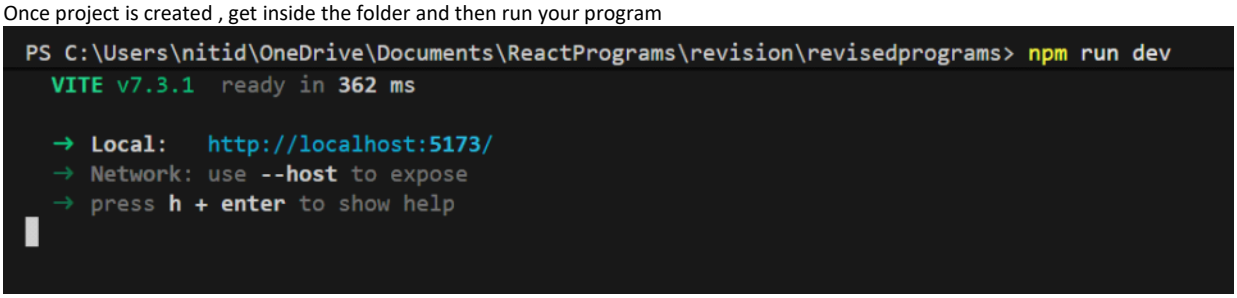
d) Start your build process

Run your build process with npm run dev or whatever command is configured in your package.json file.
npm run dev

1) Created a first Vite Project

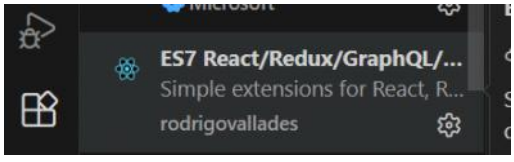


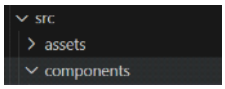
Once project is created , get inside the folder and then run your program



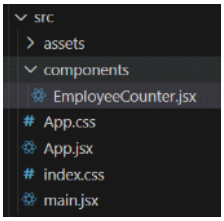
2) Create a component in src/components folder EmployeeCounter.jsx
Note*- Shortcut key to write a function in that component we have rfce (To get this we have to add an extension)

3)





4)



5) Then import that <EmployeeCounter> component in App.jsx to paralley view the changes on a browser

```jsx
import EmployeeCounter from './components/EmployeeCounter'


function App() {
  const [count, setCount] = useState(0)

  return (
    <>

      <div>
        <p className="text-2xl font-bold mb-4">
        Content from App.jsx
      </p>
      <EmployeeCounter></EmployeeCounter>
      </div>

    </>
  )
}

export default App
```

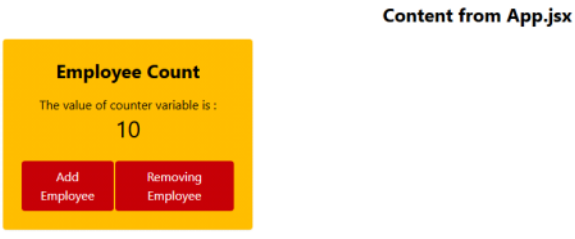
6) Define a initial count as 0 in useState , use a add button to increase count and remove to decrease the count and count never goes below 0 so disable the remove when counter is 0;

EmployeeCounter.jsx

ReactPrograms > revision > revisedprograms > src > components > EmployeeCounter.jsx > EmployeeCounter

```jsx
import React from 'react'
import { useState } from 'react'
function EmployeeCounter() {

    // Here in a state you are setting an initial as 10 after that you can either
    // set or get the value from useState
    const [getCount , setCount] = useState(10);
    //useState : -- It is use to store the content and get the content
    // basically it is state management which is dynamic

    return (
    // It will return or rendered HTML Code and it's a jsx file where we HTML + JavaScript Code
    //  and internally it is converting into javascript

    //To check Tailwindcss we can apply tailwind class names here the class name should be as a className keyword
    // because in javascript there is already a class as a keyword exist
    <div className="bg-amber-400 p-6 rounded shadow w-80 text-center">


      <h1 className='text-2xl font-bold mb-4'>Employee Count </h1>

      {/* getCount is dynamic so we have to specify in {} even styling , obj ,props it is in {} */}
      The value of counter variable is : <p className='text-3xl mb-6'>{getCount}</p>


      {/* // For reusability we can create our own MyButton component also so we need not to
      // define the same tailwind properties on every button */}
      <div className='flex gap-0.5 '>
      <button className="bg-red-700 text-white px-4 py-2 rounded hover:cursor-pointer "   onClick={() => setCount(getCount+1)}> Add Employee</button>
      <button className="bg-red-700 text-white px-4 py-2 rounded hover:cursor-pointer"   onClick={() => setCount(getCount-1)} disabled={getCount == 0}> Removing Employee</button>
    </div>

    </div>
  )
}

export default EmployeeCounter
```

Output:



Activity :

Create a simple inventory screen where

a) Manager sees inventory count
b) Can add stock
c) Can remove stock
d) Remove disabled when stock is zero

Day-2

Wednesday, January 21, 2026 11:48 AM

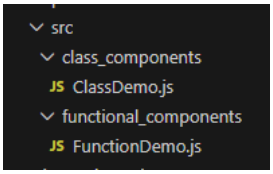
Components -- It's used when break the UI into reusable components and making them talk to each other just like objects in java

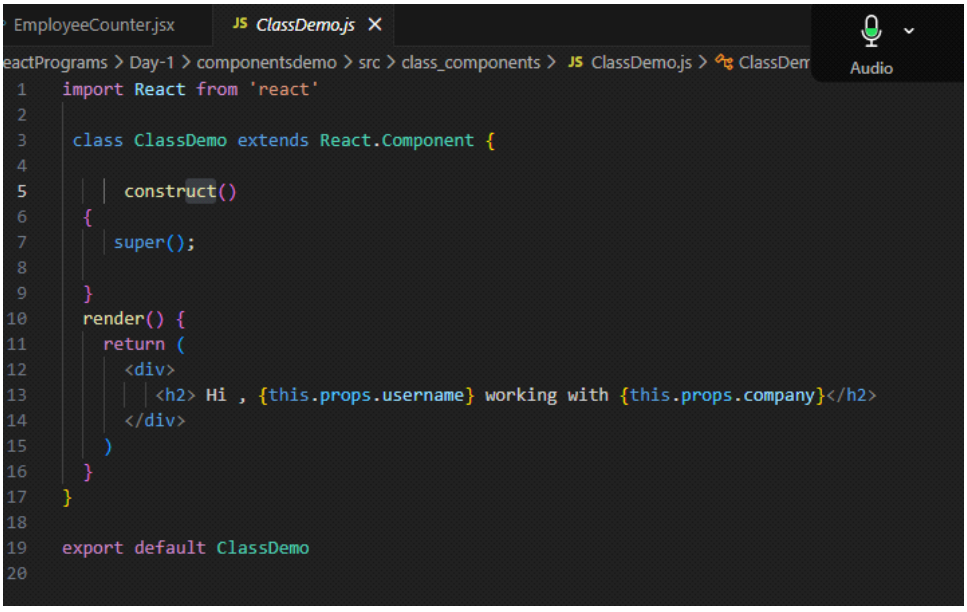
Java Class ---> React Component
Object --> Component instances
Method callings -- Props

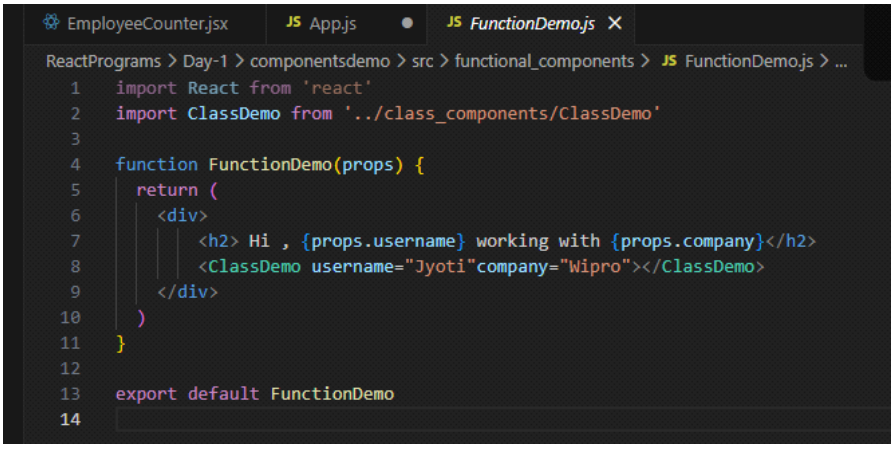
Navbar component
Header component
Product card
Employee card component

Components can be created via two ways either

a) Class component --- Legacy (Outdated / traditional way) -- explicitly we have to define render()
b) Functional component -- Standard --- here render() is not required it's automatically gets rendered that you have defined in return()







```
return (
  <>
  <div >
    <h1> Welcome to Create-React-App </h1>
   <h4> Example of types of Components :</h4>
  {/* props -- means to access the properties of one components or passing the data from one c to another company  */}
    <FunctionDemo username="Niti" company="GreatLearning"/>

  </div>
```

What are the advantages of creating Functional Components

1) Less Boilerplate
2) Hooks Support wherein class doesn't
3) Better performance
4) Easy to test

PROPS -- It is used to maintain unidirectional data flow which is passing the data from parent to child Props are read-only inputs passed from Parent to Child component foreg: app.jsx is the parent for FunctionDemo component so we are passing the data from app to functiondemo component

Parent fetches the data and child display the data

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

  return (
    <>

      <div>
        <p className="text-2xl font-bold mb-4">
        Content from App.jsx
      </p>
       <EmployeeCounter></EmployeeCounter>

       <div>
         {/*From Parent we are passing the data as a props to a product child component  */}
        <Product productname="Laptop" productprice={70000} />
        <Product productname="Mouse" productprice={700} />
        <Product productname="Keyboard" productprice={7000} />

       </div>
      </div>

    </>
  )
}
```

```
import React from 'react'
import { FaRupeeSign } from "react-icons/fa";

function Product(props) {
  return (
    <div>

      <p>Product Name :{props.productname}</p>
      <div className='relative'>
      <p>Price : <FaRupeeSign className='absolute'/><span>{props.productprice}</span></p>
      </div>
    {/* To add react icons i.e. price icon we have to install -  npm install react-icons --save*/}
    {/* This component is the child component of Parent i.e. App.jsx where we are viewing the props  */}

    </div>
  )
}

export default Product
```

Event Handling : is allowing react components to take an action or respond to a user actions like click , input , submit

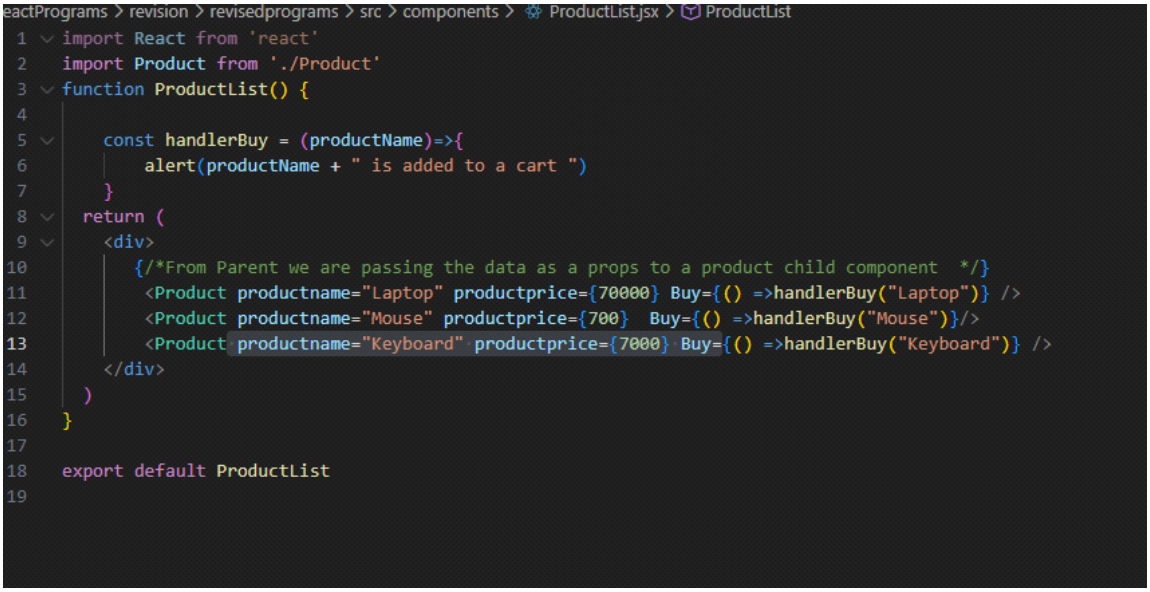
So created a ProductList.jsx to define all the products and import product component in it and then import in app.jsx to

```jsx
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'
import EmployeeCounter from './components/EmployeeCounter'
import ProductList from './components/ProductList'

function App() {
  const [count, setCount] = useState(0)

  return (
    <>

      <div>
        <p className="text-2xl font-bold mb-4">
        Content from App.jsx
      </p>
       <EmployeeCounter></EmployeeCounter>

       <div>

      <h1> Product Listing :</h1>
      <ProductList/>
       </div>
      </div>

    </>
  )
}

export default App
```

In product we can also access the properties in the form of destructuring it instead of accessing all properties in the form of obj or props that we were passing Product(props) we can destructure to a single property or multiple properties we want to access .

ReactPrograms > revision > revisedprograms > src > components > Product.jsx > ...

```jsx
import React from 'react'
import { FaRupeeSign } from "react-icons/fa";

function Product({productname,productprice,Buy}) {
  return (
    <div>

        <p>Product Name :{productname}</p>
        <div className='flex justify-center'>
            {/* To add react icons i.e. price icon we have to install -  npm install react-icons --save*/}
    {/* This component is the child component of Parent i.e. App.jsx where we are viewing the props  */}
          <span> Price :</span>  <FaRupeeSign/> <span>{productprice}</span>
       </div>

      <button className=" bg-blue-800  text-white px-3 py-2 mt-2 rounded hover:cursor-pointer" onClick={Buy}> Buy </button>

    </div>
  )
}

export default Product

```



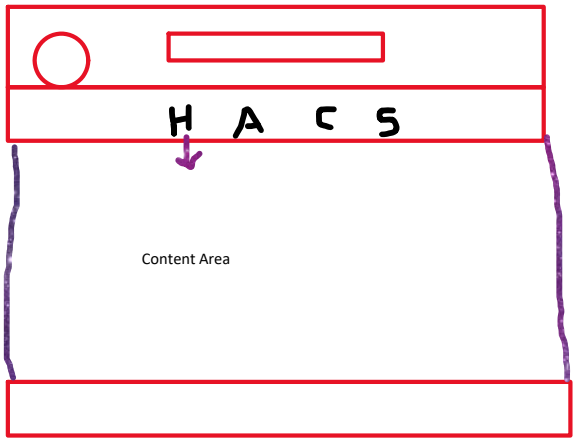
Activity :

Create two components employee and employeelist(name ,role) and display the name and role with the event button which will display the name of employee if he is promoted

Day-3

Wednesday, January 21, 2026 2:36 PM

Routing / Navigation



React is a SPA -- Single Page Application (then how we are going to connect /navigate to multiple pages)

Routing is used for that purpose -- For Handling the Navigation

MVC Controller -- React Route
@RequestMapping --- <Route path="">

Routing in react allows navigation between different UI views without page reload , maintaining the SPA behaviour.

/login , /register , /orders , /profile

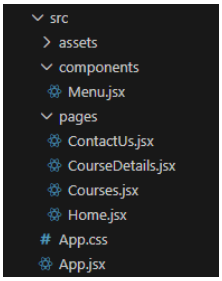
Step1 : npm install react-router-dom

Few Components we get after installing react-router-dom

BrowserRouter
Routes
Route
Link
useParams
useNavigate

Step 2 : In main.jsx change StrictMode to BrowserRouter

```jsx
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'
import { BrowserRouter } from 'react-router-dom'

createRoot(document.getElementById('root')).render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
)
```



Step 3:
using Vite project if it is giving react-router-dom module issue then use the below command
npm install react-router-dom localforage match-sorter sort-by

Tutorial v6.28.0 | React Router

Create a project of having a login screen (add form using bootstrap) create a navbar through routing it will redirect to a login page , about us , contact us(Contact Form)

Menu.jsx

ReactPrograms > revision > routingexample > src > components > Menu.jsx > ...

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

function Menu() {
  return (
    <>
      <div>
        <h3>Link Tag</h3>

        <nav style={{ padding: "10px", background: "#eee" }}>
          <Link to="/">Home</Link> |{" "}
          <Link to="/courses">About Us</Link> |{" "}
          <Link to="/contact">Contact Us</Link>
        </nav>
      </div>
    </>
  );
}

export default Menu;
```

Menu.jsx App.jsx

ReactPrograms > revision > routingexample > src > App.jsx > App

```jsx
import './App.css'
import {Routes,Route} from 'react-router-dom';
import Courses from './pages/Courses'
import Home from './pages/Home'
import CourseDetails from './pages/CourseDetails'
import ContactUs from './pages/ContactUs'
import Menu from './components/Menu'

function App() {


  return (
    <>
      <Menu />
      <Routes>
        <Route path="/" element={<Home/>} />
        <Route path="/courses" element={<Courses/>} />
        <Route path="/courses/:id" element={<CourseDetails/>} />
        <Route path="/contact" element={<ContactUs/>} />
      </Routes>
    </>
  )
}

export default App

```

For Dynamic Query use useParams(); and first define two further navigation links which will route to the path
<Route path="/courses/:id" element={<CourseDetails />} />
And fetch the parameter using useParam(); and store it in id variable and print the id in CourseDetails component

Menu.jsx Courses.jsx CourseDetails.jsx ContactUs.jsx

ReactPrograms > revision > routingexample > src > pages > Courses.jsx > Courses

```jsx
import React from 'react'
import {Link} from 'react-router-dom'
function Courses() {
  return (
    <div>
      <ul>
        <li><Link to='/courses/1'>React Course</Link></li>
        <li><Link to='/courses/2'>Java Course</Link></li>
      </ul>
    </div>
  )
}

export default Courses

```


Menu.jsx
Courses.jsx
CourseDetails.jsx
ContactUs.jsx
eactPrograms > revision > routingexample > src > pages > CourseDetails.jsx > ...
import React from 'react'
import { useParams } from 'react-router-dom'
function CourseDetails() {
const {id} = useParams();
return (

<h3> Course Details :</h3>
 Course ID : {id}
Lorem ipsum dolor sit amet consectetur adipisicing elit.
Harum sit pariatur vel dolorum. Ex aliquam error alias,
eius sapiente maiores eveniet ab et laudantium omnis enim labore debitis consequuntur nam.

)
}
export default CourseDetails

Day-4 Controlled & Uncontrolled & JSON API Fetch

Thursday, January 22, 2026 9:23 AM

Controlled component is a form element whose value is controlled by React State.
For useState -- Hook

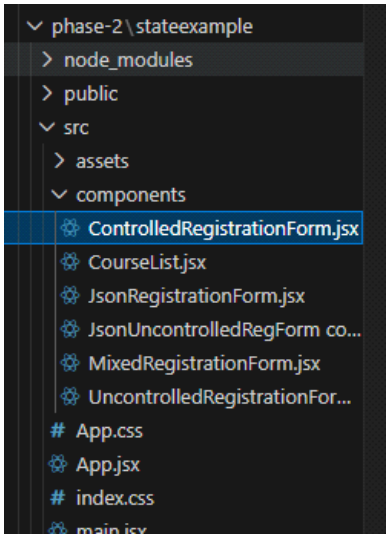
Use cases : Login Form , Payment Form , Registration Form

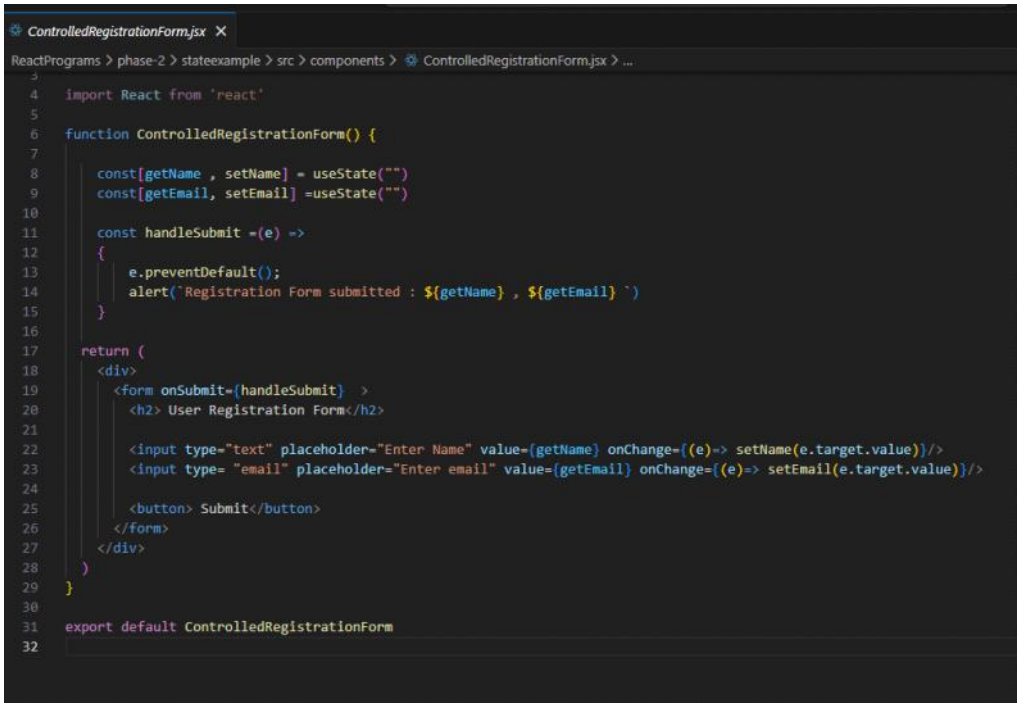
Uncontrolled components stores form data in the DOM , not in React state.

Use Cases : File upload It's like a one time activity

useRef

A ref is explicitly attached to a JSX element and React assigns the corresponding DOM NODE to ref.current during the commit phase.





UncontrolledRegistrationForm.jsx

ReactPrograms > phase-2 > stateexample > src > components > UncontrolledRegistrationForm.jsx > UncontrolledRegistrationForm

```jsx
import React from 'react'

import {useRef} from "react"

function UncontrolledRegistrationForm() {

    const usernameRef = useRef();  // reference variable
    const passwordRef = useRef();

    const handleSubmit =(e) =>
    {
        e.preventDefault();
        alert("Value :"+  usernameRef.current.value)
    }

  return (
   <div>
     <form >
       <h2> User Registration Form</h2>

       <input ref={usernameRef} />

       <input ref={passwordRef} />


       <button onClick= {handleSubmit}> Submit</button>
     </form>
   </div>
  )
}

export default UncontrolledRegistrationForm

```

App.jsx

ReactPrograms > phase-2 > stateexample > src > App.jsx > App

```jsx
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'

import ControlledRegistrationForm from './components/ControlledRegistrationForm'

import MixedRegistrationForm from './components/MixedRegistrationForm'
import JsonRegistrationForm from './components/JsonRegistrationForm'
import CourseList from './components/CourseList'
function App() {
  const [count, setCount] = useState(0)

  return (
    <>
      {/* <ControlledRegistrationForm />

      <br/>

      <UncontrolledRegistrationForm/>

      <br />

      <MixedRegistrationForm /> */}


      <JsonRegistrationForm />

      <CourseList />

    </>
  )
}

export default App

```

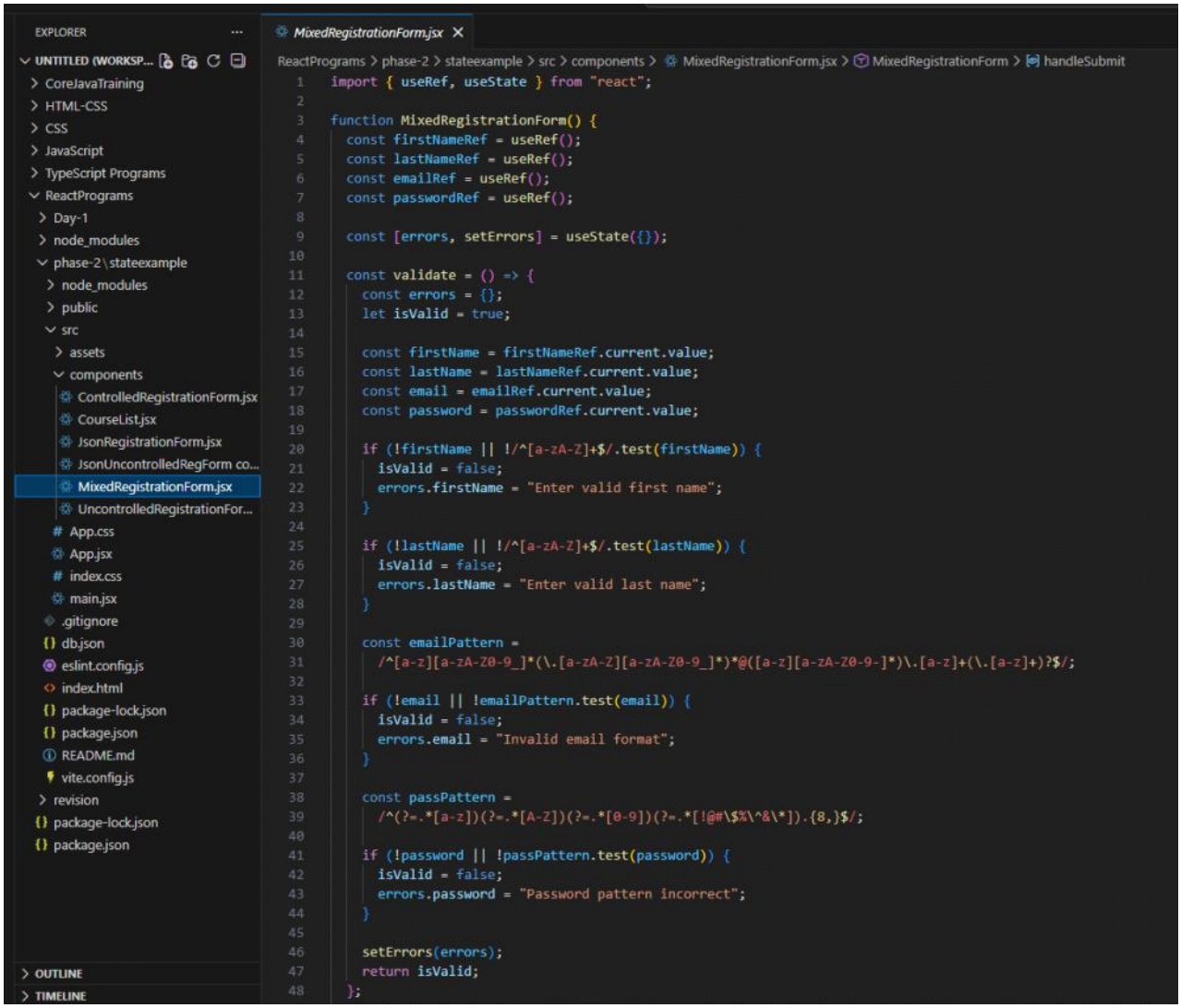
Activity:

Create a registration form with the mandatory constraints

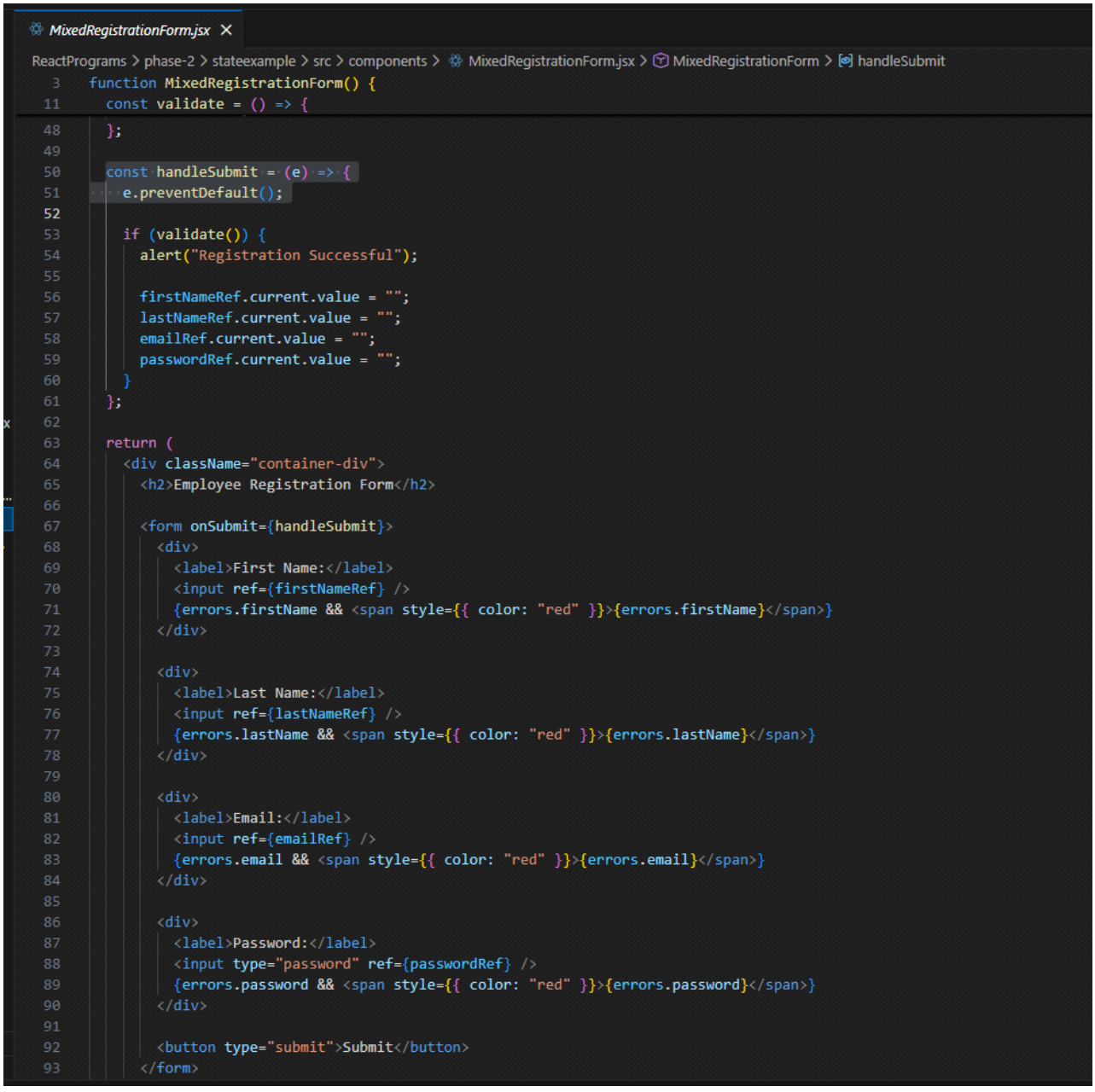
1. Use useRef for
    a. FirstName
    b. LastName
    c. Email
    d. Password
2. Use useState for (because high chances of repeated stroke)
    a. Validation error messages to print
    b. Submission Status

Do not use useState for every input value : so read the input values only on submit
Perform the validation inside submit handler

Validation rules :
First name & Last name
Required
Alphabets
For email : (It should have email format)
For password : Min 8 characters , 1 uppercase , 1 lowercase , 1 number and 1 special character



In cont…


MixedRegistrationForm.jsx
ReactPrograms > phase-2 > stateexample > src > components > MixedRegistrationForm.jsx > MixedRegistrationForm > handleSubmit
function MixedRegistrationForm() {
const validate = () => {
};
const handleSubmit = (e) => {
e.preventDefault();
if (validate()) {
alert("Registration Successful");
firstNameRef.current.value = "";
lastNameRef.current.value = "";
emailRef.current.value = "";
passwordRef.current.value = "";
}
};
return (

<h2>Employee Registration Form</h2>
<form onSubmit={handleSubmit}>

<label>First Name:</label>
<input ref={firstNameRef} />
{errors.firstName && <span style={{ color: "red" }}>{errors.firstName}}

<label>Last Name:</label>
<input ref={lastNameRef} />
{errors.lastName && <span style={{ color: "red" }}>{errors.lastName}}

<label>Email:</label>
<input ref={emailRef} />
{errors.email && <span style={{ color: "red" }}>{errors.email}}

<label>Password:</label>
<input type="password" ref={passwordRef} />
{errors.password && <span style={{ color: "red" }}>{errors.password}}

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

In cont...

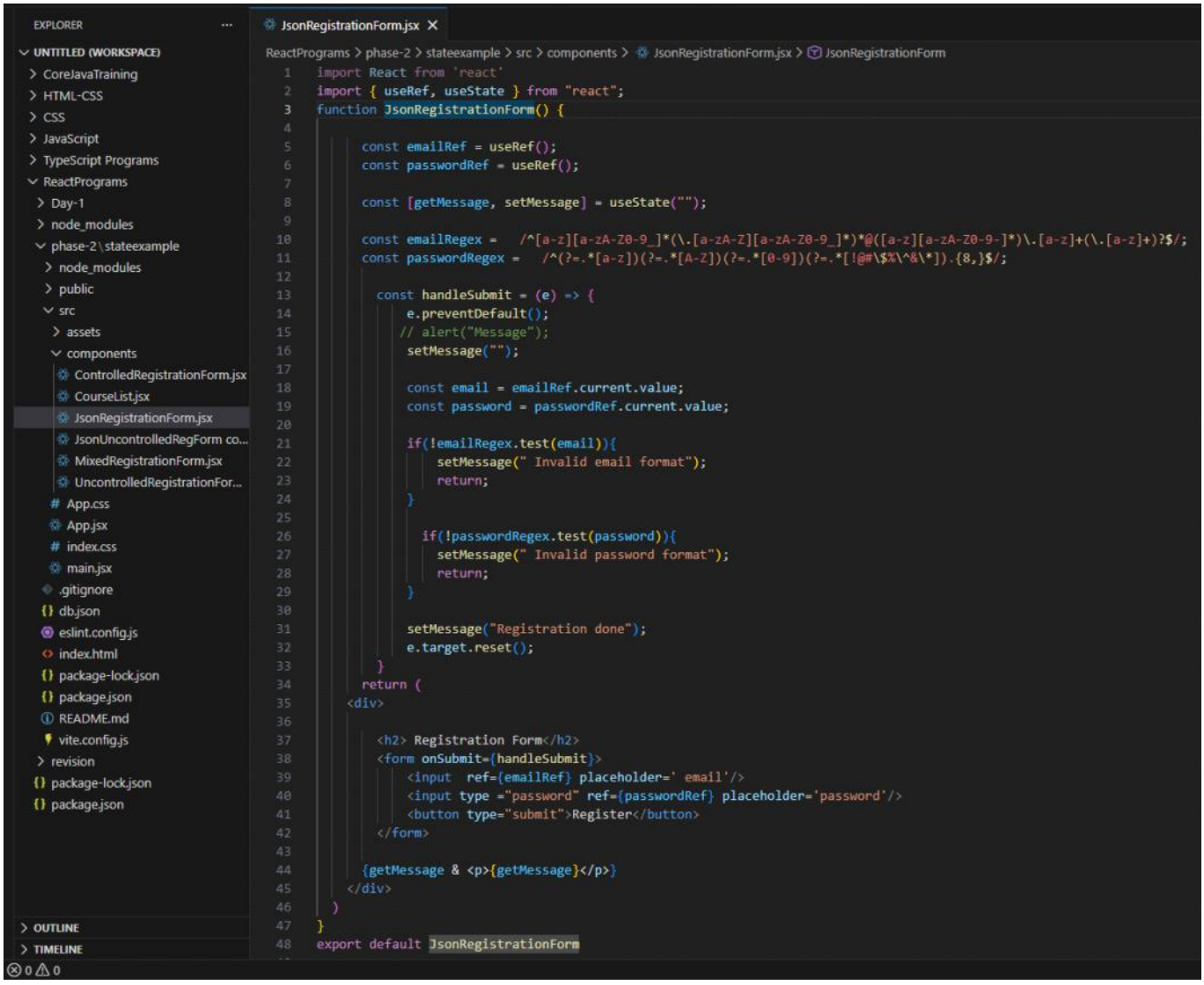
<label>Password:</label>
<input type="password" ref={passwordRef} />
{errors.password && <span style={{ color: "red" }}>{errors.password}}

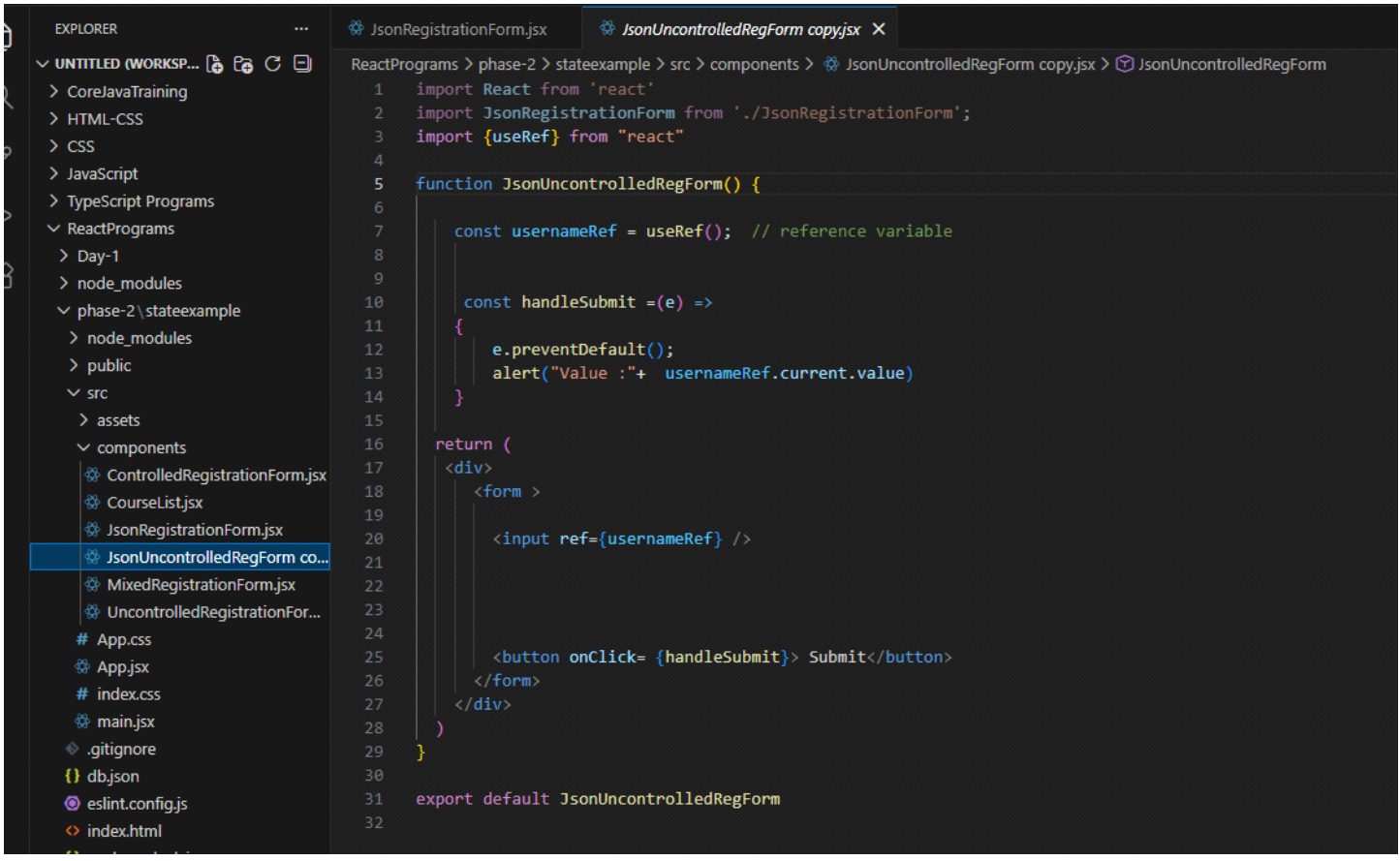
<button type="submit">Submit</button>
</form>

);
}
export default MixedRegistrationForm;

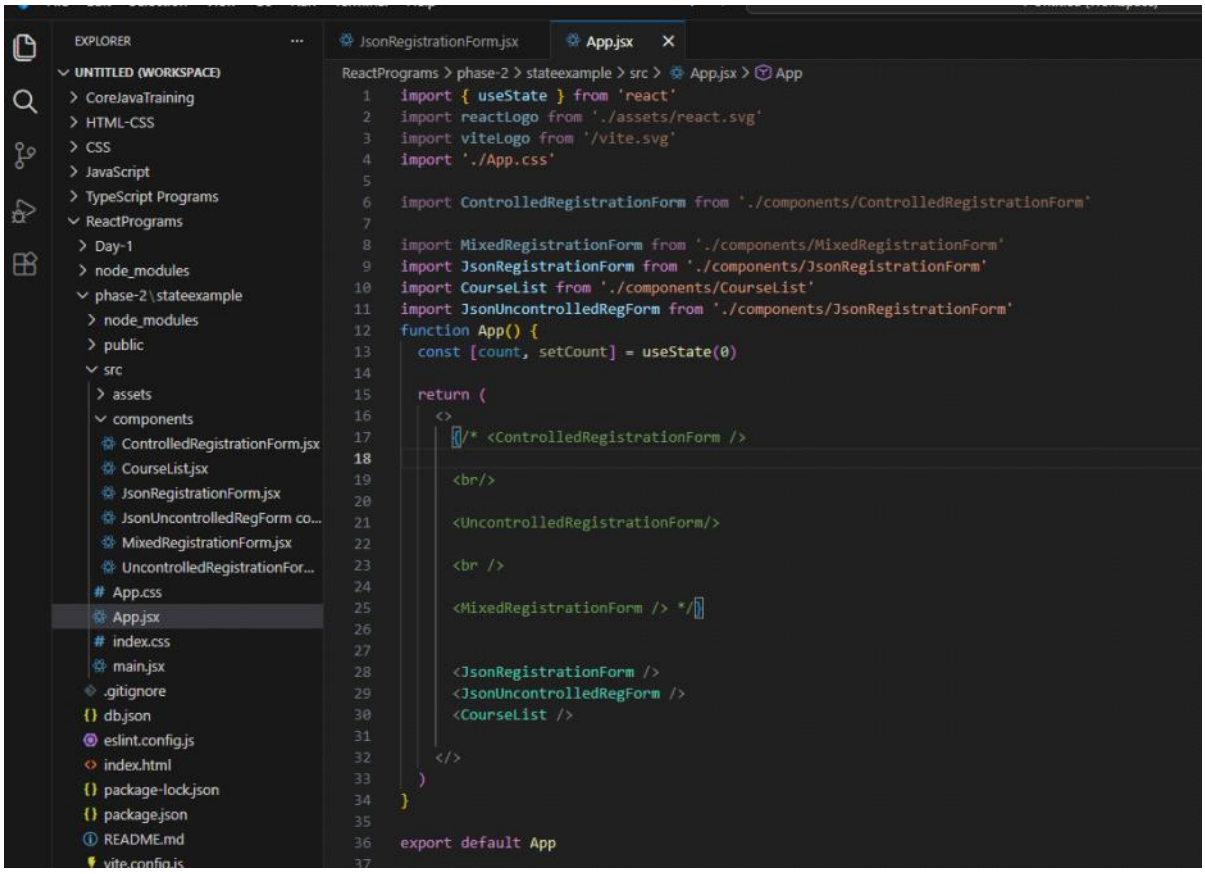
npm cache clean --force
npm install json-server@0.17.4 --save-dev --force
node node_modules/json-server/lib/cli/bin.js --watch db.json --port 3000
npm install vite-plugin-json-server --save-dev
npx json-server --watch db.json --port 3001 to run json server

Npm install to install node modules






```jsx
import { useEffect, useState } from "react";

function CourseList() {
  const [courses, setCourses] = useState([]);

  useEffect(() => {
    fetch("http://localhost:3001/courses")
      .then((res) => {
        if (!res.ok) {
          throw new Error("Network response was not ok");
        }
        return res.json();   // ✅ MUST be json()
      })
      .then((data) => setCourses(data))
      .catch((error) =>
        console.error("Backend error:", error.message)
      );
  }, []);

  return (
    <div>
      <h2>Course List from Backend</h2>
      {courses.map((course) => (
        <p key={course.id}>
          {course.name} - {course.duration}
        </p>
      ))}
    </div>
  );
}

export default CourseList;
```



(((Steps also you can use as below:)
BACKEND SETUP (json-server)
Install json-server

npm install -g json-server
Create db.json (project root)

```
{
  "courses": [
    { "id": 1, "name": "React", "duration": "30 Days" },
    { "id": 2, "name": "Spring Boot", "duration": "45 Days" },
    { "id": 3, "name": "Node JS", "duration": "25 Days" }
  ]
}
```

Start backend server

json-server --watch db.json --port 3001
API URL:

http://localhost:3001/courses

src/main.jsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
import "./index.css";
ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

src/App.jsx

```
import RegistrationForm from "./components/RegistrationForm";
import UncontrolledForm from "./components/UncontrolledForm";
```

```
import CourseList from "./components/CourseList";
function App() {
 return (
  <div className="container">
   <h1>Training Portal</h1>
<RegistrationForm />
   <UncontrolledForm />
   <CourseList />
  </div>
 );
}
export default App;
```

src/components/RegistrationForm.jsx
(useRef + useState – industry optimized)

```
import { useRef, useState } from "react";
function RegistrationForm() {
 const emailRef = useRef();
 const passwordRef = useRef();
const [message, setMessage] = useState("");
const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
 const passwordRegex =
  /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&]).{8,}$/;
const handleSubmit = (e) => {
  e.preventDefault();
  setMessage("");
const email = emailRef.current.value;
  const password = passwordRef.current.value;
if (!emailRegex.test(email)) {
   setMessage(" Invalid email format");
   return;
  }
if (!passwordRegex.test(password)) {
   setMessage(" Password pattern incorrect");
   return;
  }
setMessage(" Registration Done Successfully");
  e.target.reset();
 };
return (
  <div className="card">
   <h2>Registration Form</h2>
<form onSubmit={handleSubmit}>
    <input ref={emailRef} placeholder="Email" />
    <input
     ref={passwordRef}
     type="password"
     placeholder="Password"
    />
    <button type="submit">Register</button>
   </form>
{message && <p>{message}</p>}
  </div>
 );
}
export default RegistrationForm;
```

src/components/UncontrolledForm.jsx
(Pure DOM access – useRef)

```
import { useRef } from "react";
function UncontrolledForm() {
 const nameRef = useRef();
const handleSubmit = () => {
  alert("Entered Name: " + nameRef.current.value);
 };
return (
  <div className="card">
   <h2>Uncontrolled Form</h2>
<input ref={nameRef} placeholder="Enter Name" />
   <button onClick={handleSubmit}>Submit</button>
  </div>
 );
```

```
}
export default UncontrolledForm;
```

src/components/CourseList.jsx
(Backend integration – REST API)

```
import { useEffect, useState } from "react";
function CourseList() {
  const [courses, setCourses] = useState([]);
useEffect(() => {
   fetch("http://localhost:3001/courses")
     .then((res) => res.json())
     .then((data) => setCourses(data))
     .catch(() => console.log("Backend not running"));
  }, []);
return (
   <div className="card">
     <h2>Courses (From Backend)</h2>
{courses.length === 0 && <p>Loading...</p>}
{courses.map((course) => (
       <p key={course.id}>
         {course.name} – {course.duration}
       </p>
     ))}
   </div>
  );
}

export default CourseList;
```

src/index.css (basic styling)

```
body {
  font-family: Arial, sans-serif;
}
.container {
  padding: 20px;
}
.card {
  border: 1px solid #ccc;
  padding: 15px;
  margin-bottom: 20px;
}
input {
  display: block;
  margin: 8px 0;
  padding: 5px;
}
button {
  padding: 5px 10px;
}
```

RUN APPLICATION
Terminal 1 (Backend)

```
json-server --watch db.json --port 3001
```

Terminal 2 (Frontend)

```
npm install
npm run dev
```

Day-5 Routing Transition Concepts

Friday, January 23, 2026 9:37 AM

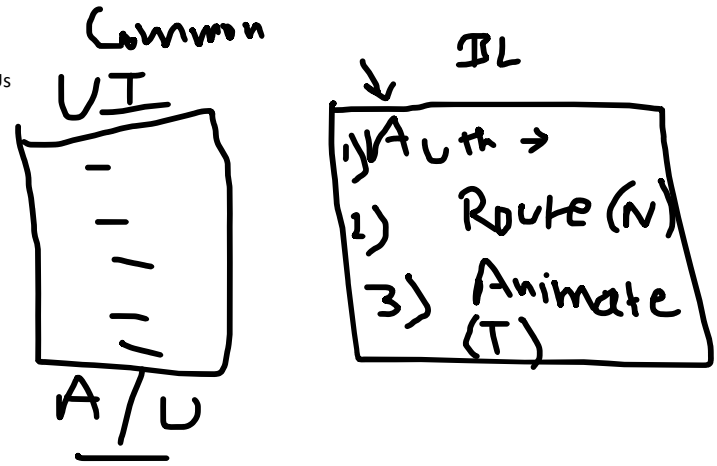
(If you want to transition into any components and from a component then we can use Routing Transition)

Real Problem :

Instantly removes Home and renders Employees/Course/About Us

Abruptly page got changed
No visual Continuity in between them
Sometime it feels unprofessional for enterprise apps

For Eg:
HR
Banking Dashboards
Admin Panel

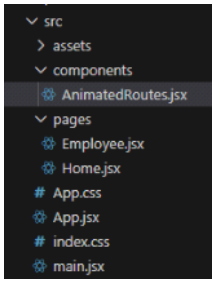


How we implement by creating a separate concerns

To implement this

Router ---- React-router
Animation --- react-transition-group --- lifecycle -based animation

```
> npm install react-router-dom react-transition-group
```

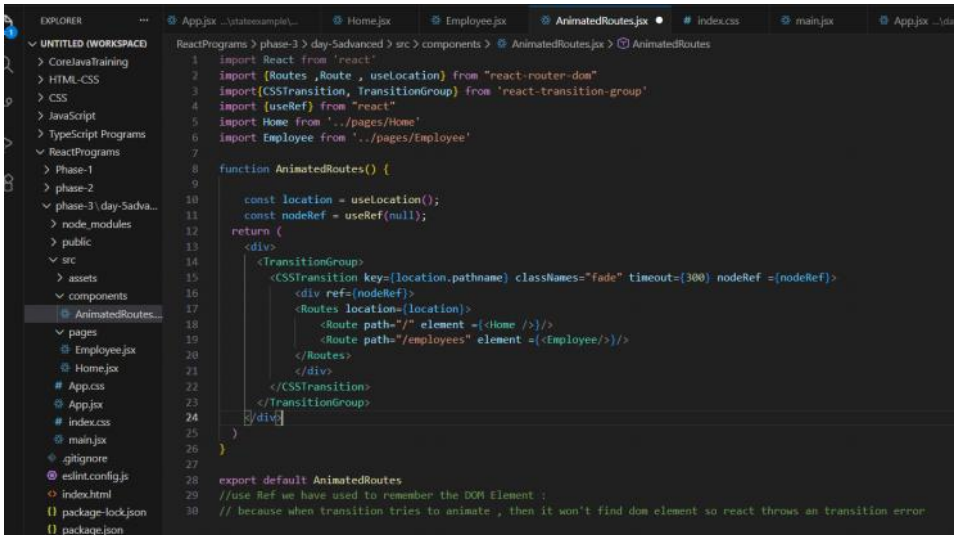


AnimatedRoutes :- centralized and a single component which is responsible for animation + routing

For this we have to create a separate CSS file which is centralized and easy for animation tuning

For React Route transition what is our requirement :

a) Detect the route change
b) Apply enter/exit animation
c) Wait for animation to finish
d) Render next route



```jsx
import React from 'react'
import {Routes ,Route , useLocation} from "react-router-dom"
import{CSSTransition, TransitionGroup} from 'react-transition-group'
import {useRef} from "react"
import Home from '../pages/Home'
import Employee from '../pages/Employee'

function AnimatedRoutes() {

    const location = useLocation();
    const nodeRef = useRef(null);
  return (
    <div>
      <TransitionGroup>
        <CSSTransition key={location.pathname} classNames="fade" timeout={300} nodeRef ={nodeRef}>
            <div ref={nodeRef}>
            <Routes location={location}>
                <Route path="/" element ={<Home />}/>
                <Route path="/employees" element ={<Employee/>}/>
            </Routes>
            </div>
        </CSSTransition>
      </TransitionGroup>
    </div>
  )
}

export default AnimatedRoutes
//use Ref we have used to remember the DOM Element :
// because when transition tries to animate , then it won't find dom element so react throws an transition error
```

App.jsx ...\stateexample\... | Home.jsx | Employee.jsx

ReactPrograms > phase-3 > day-5advanced > src > pages > Home.jsx > ...

```jsx
import React from 'react'

function Home() {
  return (
    <div>
      <h2>Home Page Component </h2>
    </div>
  )
}

export default Home

```

App.jsx ...\stateexample\... | Home.jsx | Employee.jsx | AnimatedRoutes.js

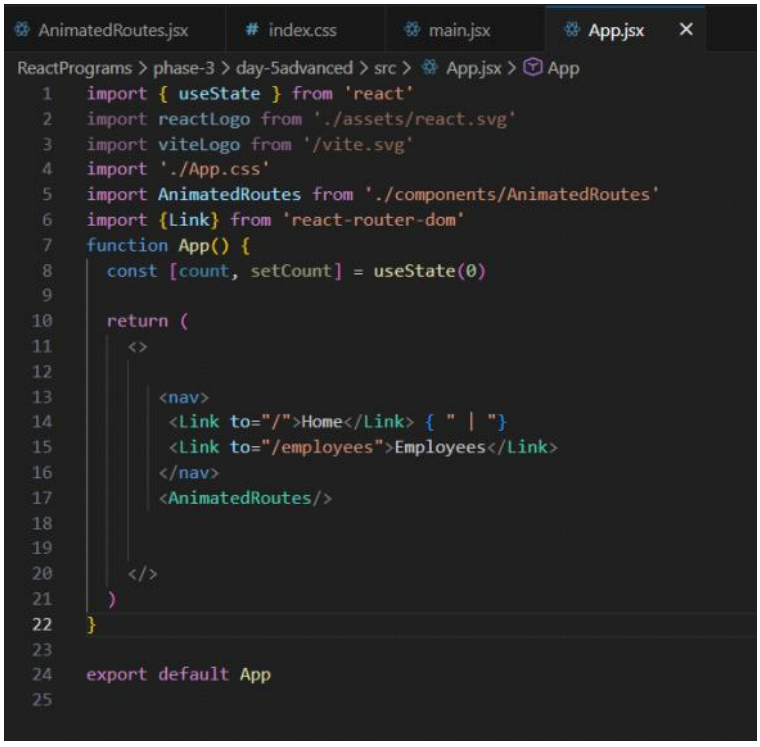
ReactPrograms > phase-3 > day-5advanced > src > pages > Employee.jsx > ...

```jsx
import React from 'react'

function Employee() {
  return (
    <div>
      <h2>Employee Page Component </h2>
    </div>
  )
}

export default Employee

```

AnimatedRoutes.jsx | index.css | main.

ReactPrograms > phase-3 > day-5advanced > src > # ind

```css
.fade-enter{


  opacity:0;
}
.fade-enter-active
{
  opacity:1;
  transition:opacity 300ms;
}

.fade-exit
{
  opacity:1;
}

.fade-exit-active
{
  opacity:0;
  transition:opacity 300ms;
}
```



HOC -- **High Order Component** -

Before HOC

Function Employee{
If(!isLoggedIn)
{return <h2> Access Denied!</h2>
}
Return
<h3>Employee Data</h3>
}

If you will repeat this same login check on all components(Dashboards) let's say 10 components
Then code will be duplicated and also difficult to maintain ,
If there is a bug in a logic then you have to make changes in all the components

So to overcome this we have a concept of High Order Component that says cross-cutting logic must be centralized

Syntax :

Const EnhancedComponent = HOC(OriginalComponent)

So, A HOC is a function that
a) Takes a component as input
b) Adds extra logic
c) Return a new enhanced component

Formik and Yup -- *Form Handling*

Real Problem we were facing as Manual Validation , lot of State handing , everywhere we were using error message

So we have third party library (packages)
FORMIK and Yup are the third party libraries which provides components to use in React

Npm install formik yup

Import {formik , form, field , ErrorMessage} from "formik";

These are prebuilt React components provided by the formik library

<Formik
initialValue={{name:"" , email:"" }}
validationSchema ={schema}
onSubmit={handleSubmit}
/>

It will be first creating a form **state , it will track the value, tracks errors , handles the event (submit)** and also interact with **yup for validation schema**

Why not HTML <form > why Formik:
It attaches the submit handling
Prevents Page reload
Connects to a formik state

Form- > React -> validation ->submit login

```
<input type="text" placeholder="Enter Name" value={getName} onChange={(e)=> setName(e.target.value)}/>
```

Internally it replaces with

<Field name ="name">
<ErrorMessage name="name"> - This will read the yup validation errors and shows the error for that field which will be auto updated

Yub library is a JS validation engine

Import {*} from "yup"

We are using an * here because all the error message or validation schema which you are applying contains string , number , o bject for eg: as below we created earlier :

```
if (!firstName || !/^[a-zA-Z]+$/.test(firstName)) {
      isValid = false;
      errors.firstName = "Enter valid first name";
    }
```

So here in Yup it will internally wrap in an object

Yup.object({email: Yup.string[].email().required()})

So internally Yup will validates values , returns error messages and Formik will reads these errors

So Formik is not validating itself (Formik uses or communicates with Yup for that)

Steps:
User types --> Formik stores values -> Formik asks Yup -> Yup validates --> Formik shows errors -> Fields updated --> ErrorMessage display it

Simple real example:

React is a car , formik is automatic gear system and yup is traffic rule checker

https://formik.org/docs/examples/with-material-ui
https://mui.com/material-ui/react-button/

https://formik.org/docs/tutorial

Day-6 FLUX

Saturday, January 24, 2026 9:26 AM

Flux is an architecture that enforces unidirectional data flow and centralizes state changes through well -defined actions and stores ..

State must change in a predictable and trackable way

Employee.push(employee)

HR -- Salary (Payroll) - no approval/ no logs / no rollback

EmployeeForm --- > HR Request ---> HR System router --- > Central database
HR -- Update the UI

View --> Action -- dispatcher -store - view

Advantages of implementing the FLUX Architecture is
Separation of responsibility
Clean architecture
Easy debugging

Dependency Injection : DI

A component should not create its own dependencies instead dependencies are provided from outside (It should not be tightly coupled)

```
Class user
{
String username;
Address add;
User(username)
{
This.username = username;
This.add = new Address("gbfhf");
}
```

```
}

Class Address
{
String add;

Address (add)
{
This.add = add;}

}

User obj = new user("niti");

Tightly coupled


Class  smsnotification
{
Send()
{

//bL
}
}

Class userservice
{
Constructor()
{
This.smsservice = new smsnotification();
}
}

SOLID -- Principles

In react DI will be implemented via Props

Function smsnotification()
{
Return
{notify:(message) =>console.log(message)}
}
```

```
Function userservice({smsnotification})
{
Return(<button onclick={() => smsnotification.notify("djfsjlk")}>Message
</button)
}


Const notify = Smsnotification();
Return <userservice smsnotification ={notify}
```

Day-7

24 January 2026 15:01

Component Life Cycle:

Class

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

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

  useEffect(() => {
    console.log(" Notification service started");

    const intervalId = setInterval(() => {
      console.log("Fetching notifications...");
      setCount((prev) => prev + 1);
    }, 3000);


    return () => {
      console.log(" Notification service stopped");
      clearInterval(intervalId);
    };
  }, []);

  return (
    <div>
      <h3>Notifications fetched: {count}</h3>
    </div>
  );
}

export default NotificationPanel;
```

```
import { useState } from "react";
import NotificationPanel from "./components/NotificationPanel";

function App() {
  const [show, setShow] = useState(false);

  return (
    <div>
      <button onClick={() => setShow(!show)}>
        {show ? "Hide Notifications" : "Show Notifications"}
      </button>
```

```
    {show && <NotificationPanel />}
   </div>
  );
}

export default App;
```

useEffect()
    it will mounted and run once after setting the state

Notification where the timer you have set start running

invertal

api fetching

when the component is active
app keeps fetching data and ui
updates automatically

interval cleared ,the memory finished or released

Read based examples
Websocket
api polling
notification services

```
console.log(" Dashboard module loaded");

function Dashboard() {
  console.log(" Dashboard component rendered");
  return <h2>Dashboard Page</h2>;
}

export default Dashboard;
```

```
import { Suspense, lazy, useState } from "react";

const Dashboard = lazy(() => {
  console.log(" Importing Dashboard...");
  return import("./pages/Dashboard");
});
```

```
function App() {
  const [show, setShow] = useState(false);

  return (
    <div>
      <button onClick={() => setShow(true)}>
        Load Dashboard
      </button>

      {show && (
        <Suspense fallback={<h2>Loading Dashboard...</h2>}>
          <Dashboard />
        </Suspense>
      )}
    </div>
  );
}

export default App;
```

PURE COMPONENTS

Definition

Simple

A **Pure Component** re-renders **only when its input data changes**.

Technical

A Pure Component performs a **shallow comparison of props and state** to prevent unnecessary re-renders.

Advantages

- Improves performance
- Avoids unnecessary rendering
- Better scalability in large apps

If NOT Used (Industry Problem)

- UI lags
- High CPU usage
- Poor performance on low-end devices
- Hard to scale dashboards

Project Structure

components/Employee.jsx

```
function Employee({ name }) {
  console.log("Rendering Employee:", name);
  return <p>{name}</p>;
}
export default React.memo(Employee);
```

components/Dashboard.jsx

```
import { useState } from "react";
import Employee from "./Employee";
function Dashboard() {
  const [count, setCount] = useState(0);
return (
    <>
      <Employee name="Ravi" />
      <button onClick={() => setCount(count + 1)}>+</button>
    </>
  );
```

```
}
export default Dashboard;
```

ERROR BOUNDARY

Definition

Simple

Error Boundary **prevents the entire app from crashing** due to a UI error.

Technical

A React component that **catches JavaScript errors in child components** during rendering.

Advantages

- Graceful error handling
- Improves user trust
- Prevents white screen

If NOT Used

- One error crashes whole app
- No fallback UI
- Poor UX

components/ErrorBoundary.jsx

```
import React from "react";
class ErrorBoundary extends React.Component {
  state = { hasError: false };
static getDerivedStateFromError() {
    return { hasError: true };
  }
render() {
    return this.state.hasError
      ? <h3>Something went wrong</h3>
      : this.props.children;
  }
}
export default ErrorBoundary;
```

components/BuggyComponent.jsx

```
function BuggyComponent() {
  throw new Error("Crash!");
}
export default BuggyComponent;
```

App.jsx

```
<ErrorBoundary>
  <BuggyComponent />
</ErrorBoundary>
```

PORTALS

Definition

Simple

Portals allow rendering UI **outside the parent DOM hierarchy**.

Technical

ReactDOM.createPortal() renders children into a different DOM node.

Advantages

- Fix modal z-index issues
- Clean UI layering

If NOT Used

- CSS conflicts
- Broken modals
- Complex hacks

components/Modal.jsx

```
import ReactDOM from "react-dom";
function Modal() {
  return ReactDOM.createPortal(
    <div className="modal">Modal Content</div>,
    document.body
  );
}
export default Modal;
```

REACT CONTEXT API

Definition

Simple

Context API allows **sharing data globally** without passing props manually.

Technical

Provides a **global state container** via Provider & Consumer.

Advantages

- Avoids prop drilling
- Centralized config/state

If NOT Used

- Deep prop passing

- Poor readability
- Maintenance issues

context/ThemeContext.js

```
import { createContext } from "react";
export const ThemeContext = createContext();
```

App.jsx

```
<ThemeContext.Provider value="dark">
  <Header />
</ThemeContext.Provider>
```

components/Header.jsx

```
import { useContext } from "react";
import { ThemeContext } from "../context/ThemeContext";
function Header() {
  const theme = useContext(ThemeContext);
  return <h3>Theme: {theme}</h3>;
}
export default Header;
```

PROGRESSIVE WEB APP (PWA)

Definition

Simple

A PWA works **offline and can be installed** like a mobile app.

Advantages

- Offline support
- Faster loading
- Mobile-like experience

If NOT Used

- No offline access
- Poor mobile UX

Industry Use

- Flipkart Lite
- Twitter Lite
- Starbucks

REACT HOOKS

Definition

Simple
Hooks allow using state & lifecycle in functional components.

Advantages
- Cleaner code
- Reusable logic
- No class components

If NOT Used
- Verbose lifecycle code
- Poor reusability

Example

```
import { useState } from "react";
function Counter() {
  const [count, setCount] = useState(0);
  return <button onClick={() => setCount(count + 1)}>{count}</button>;
}
```

STATE MANAGEMENT (REDUX / NGRX)

Definition
Simple
Centralized store for predictable state management.

Advantages
- Single source of truth
- Easy debugging
- Scales well

If NOT Used
- Props chaos
- Inconsistent UI
- Hard debugging

Redux Flow

UI → Action → Reducer → Store → UI

redux/reducer.js

```
const reducer = (state = { count: 0 }, action) => {
  switch (action.type) {
    case "INC":
      return { count: state.count + 1 };
    default:
      return state;
```

```
  }
};
export default reducer;
```