

**1. You are developing a React application that consists of multiple functional components**

**(Header, Content, and Footer). The main App component organizes these components**

**and displays them on the screen.**

**(i) Your task is to define and export an App component that contains multiple**

**components:**

**a. A Header component that receives a title as a prop.**

**b. A Content component that displays a random joke when a button is clicked.**

**c. A Footer component that displays a static footer message.**

**(ii) Import and render the App component in index.js using ReactDOM.render().**

**Ensure the index.html file has a root element where React will mount the**

**application.**

I. App.jsx

import './App.css'

import Header from './Header'

import Footer from './Footer'

import MainContent from './MainContent'

function App() {

*return* (

<div className='top'>

<Header title="Satyaprakash Swain 22BCE1351"></Header>

<MainContent></MainContent>

<Footer></Footer>

</div>

)

}

export default App

II. Header.jsx

function Header({title}){

*return*(

<header className="head">

<h1>{title}</h1>

</header>

);

}

export default Header;

III. MainContent.jsx

import { useState } from "react";

function MainContent(){

*const* [joke,setJoke] = useState("");

*const* showJoke = ()=>{

setJoke("Why do programmers prefer dark mode? Because light attracts bugs!")

}

*return*(

<main>

<div>

<button onClick={showJoke}>Display Joke</button>

{joke && <p>{joke}</p>}

</div>

</main>

);

}

export default MainContent;

IV. Footer.jsx

function Footer(){

*return*(

<footer>

<h1>&copy; Vellore Intstitute of Techonology</h1>

</footer>

);

}

export default Footer;

V. index.css

body{

  margin: 0px;

  }

 .top{

  padding: 0px;

  margin: 0px;

  background-color: white;

  min-height: 100vh;

  width: 100vw;

  display: flex;

  flex-direction: column;

  color: black;

 }

 header{

  background-color: rgb(50, 164, 56);

  text-align: center;

 }

 main{

  flex: 1;

  display: flex;

  align-items: center;

  justify-content: center ;

 }

 main div{

  display: flex;

  flex-direction: column;

  align-items: center;

  justify-content: center;

 }

 button{

  padding: 10px;

  font-size: medium;

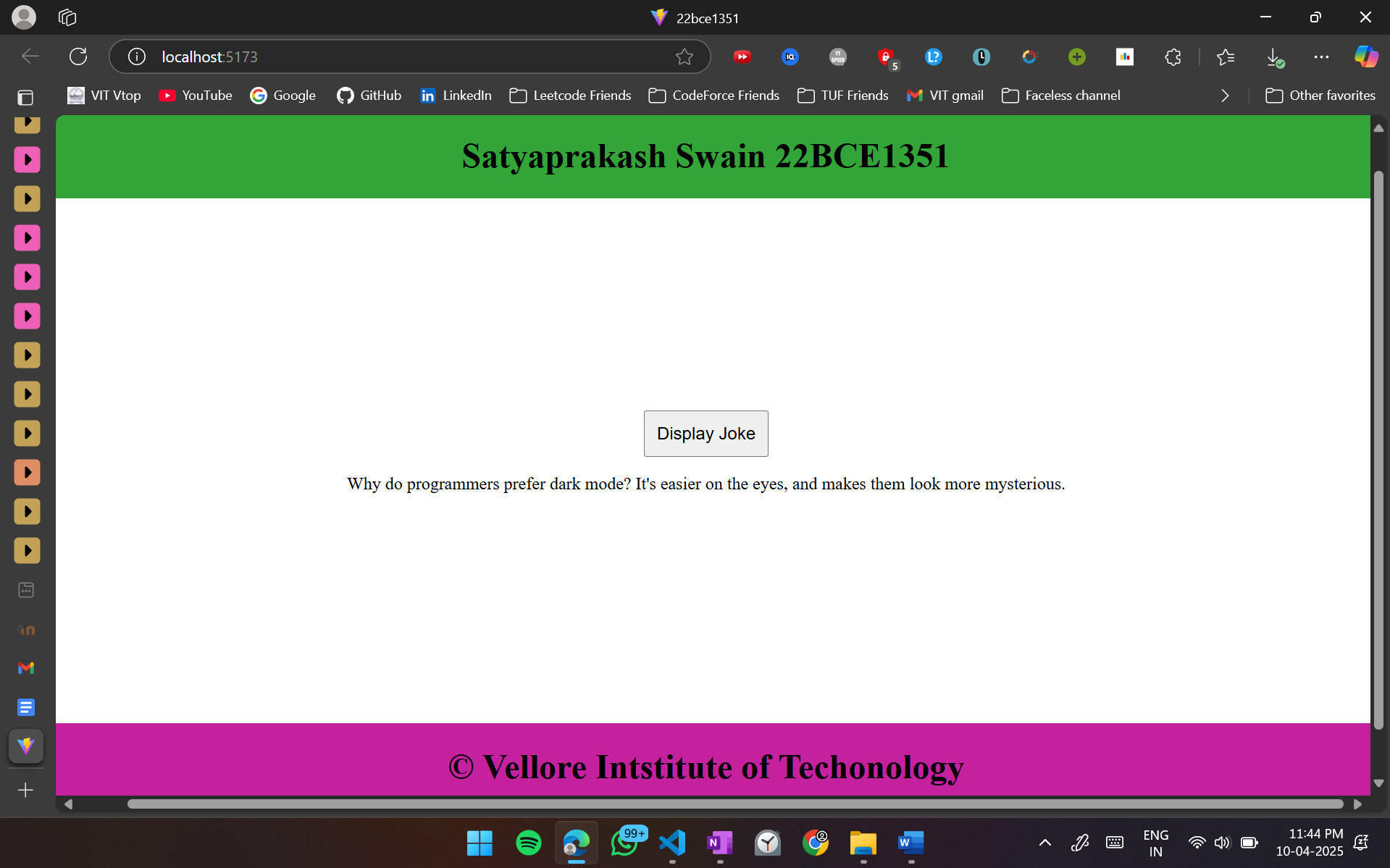
 }

 footer{

  background-color: rgb(197, 33, 159);

  text-align: center;

 }



**2. Styling in React – Inline CSS:**

* **Create a StyledButton component that applies inline CSS for background color, padding, and font size.**

1. StyledButton.jsx

function StyledButton(){

*return*(

<button style={{backgroundColor:"pink", padding:"20px",color:"black",fontSize:"30px"}}>Click Me</button>

);

}

export default StyledButton

1. App.jsx

import './App.css'

import StyledButton from './StyledButton'

function App() {

*return* (

<>

<StyledButton></StyledButton>

</>

)

}

export default App

1. **Index.css**

:root {

  font-family: system-ui, Avenir, Helvetica, Arial, sans-serif;

  line-height: 1.5;

  font-weight: 400;

  color-scheme: light dark;

  color: rgba(255, 255, 255, 0.87);

  background-color: #342299;

  font-synthesis: none;

  text-rendering: optimizeLegibility;

  -webkit-font-smoothing: antialiased;

  -moz-osx-font-smoothing: grayscale;

}

a {

  font-weight: 500;

  color: #30325b;

  text-decoration: inherit;

}

a:hover {

  color: #535bf2;

}

body {

  margin: 0;

  display: flex;

  place-items: center;

  min-width: 320px;

  min-height: 100vh;

}

h1 {

  font-size: 3.2em;

  line-height: 1.1;

}

button {

  border-radius: 8px;

  border: 1px solid transparent;

  padding: 0.6em 1.2em;

  font-size: 1em;

  font-weight: 500;

  font-family: inherit;

  background-color: #7e1212;

  cursor: pointer;

  transition: border-color 0.25s;

}

button:hover {

  border-color: #646cff;

}

button:focus,

button:focus-visible {

  outline: 4px auto -webkit-focus-ring-color;

}

@media (prefers-color-scheme: light) {

  :root {

    color: #213547;

    background-color: #ffffff;

  }

  a:hover {

    color: #747bff;

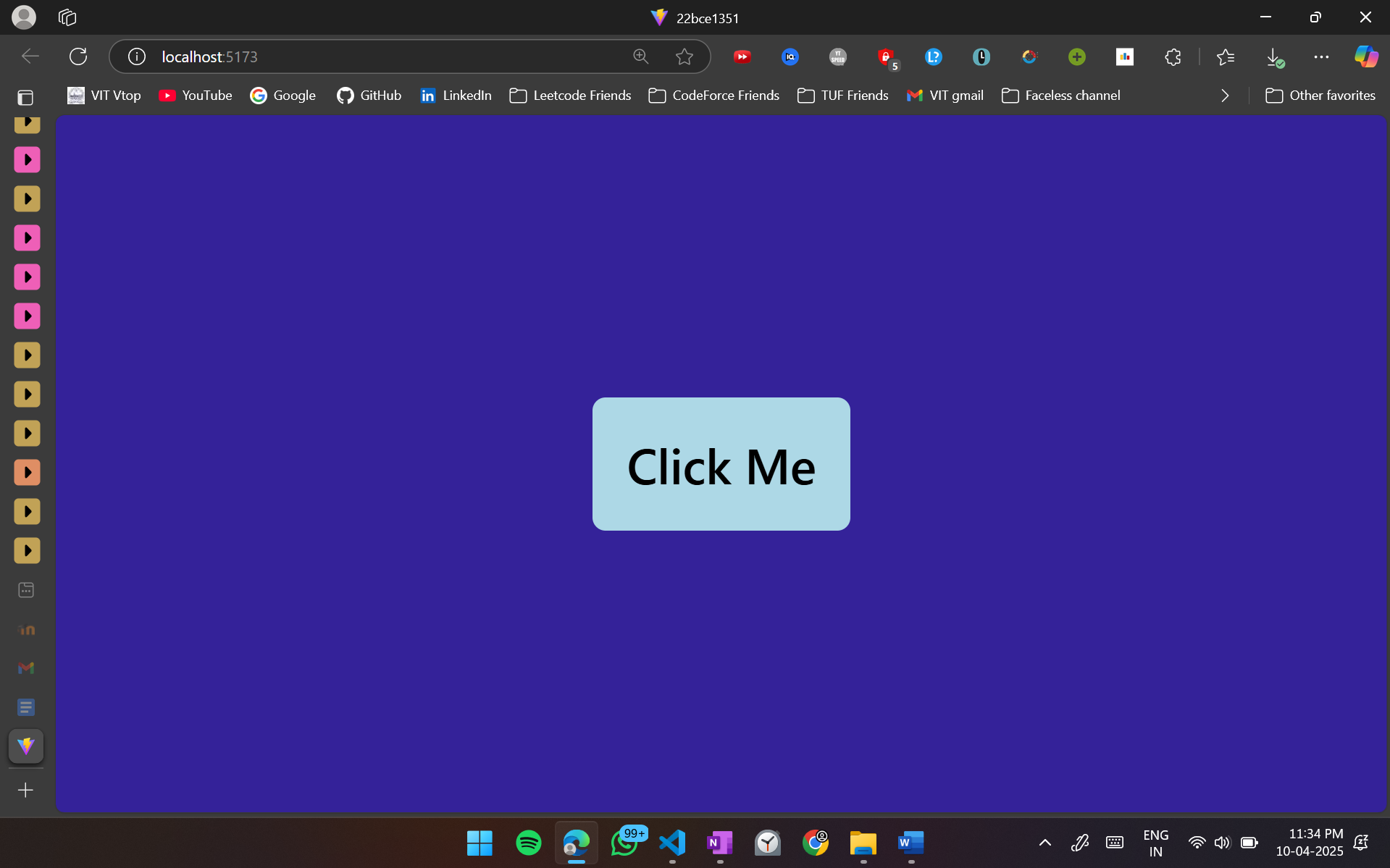
  }

  button {

    background-color: #f9f9f9;

  }

}

****

**3. Styling in React – Internal CSS:**

* **Modify the StyledButton component to include an internal <style> tag within the component for styling.**

1. StyledButton.jsx

function StyledButton(){

*const* btnStyle = {

backgroundColor:"lightblue",

padding:"20px",

color:"black",

fontSize:"30px"

}

*return*(

<button style={btnStyle}>Click Me</button>

);

}

export default StyledButton

1. App.jsx

import './App.css'

import StyledButton from './StyledButton'

function App() {

*return* (

<>

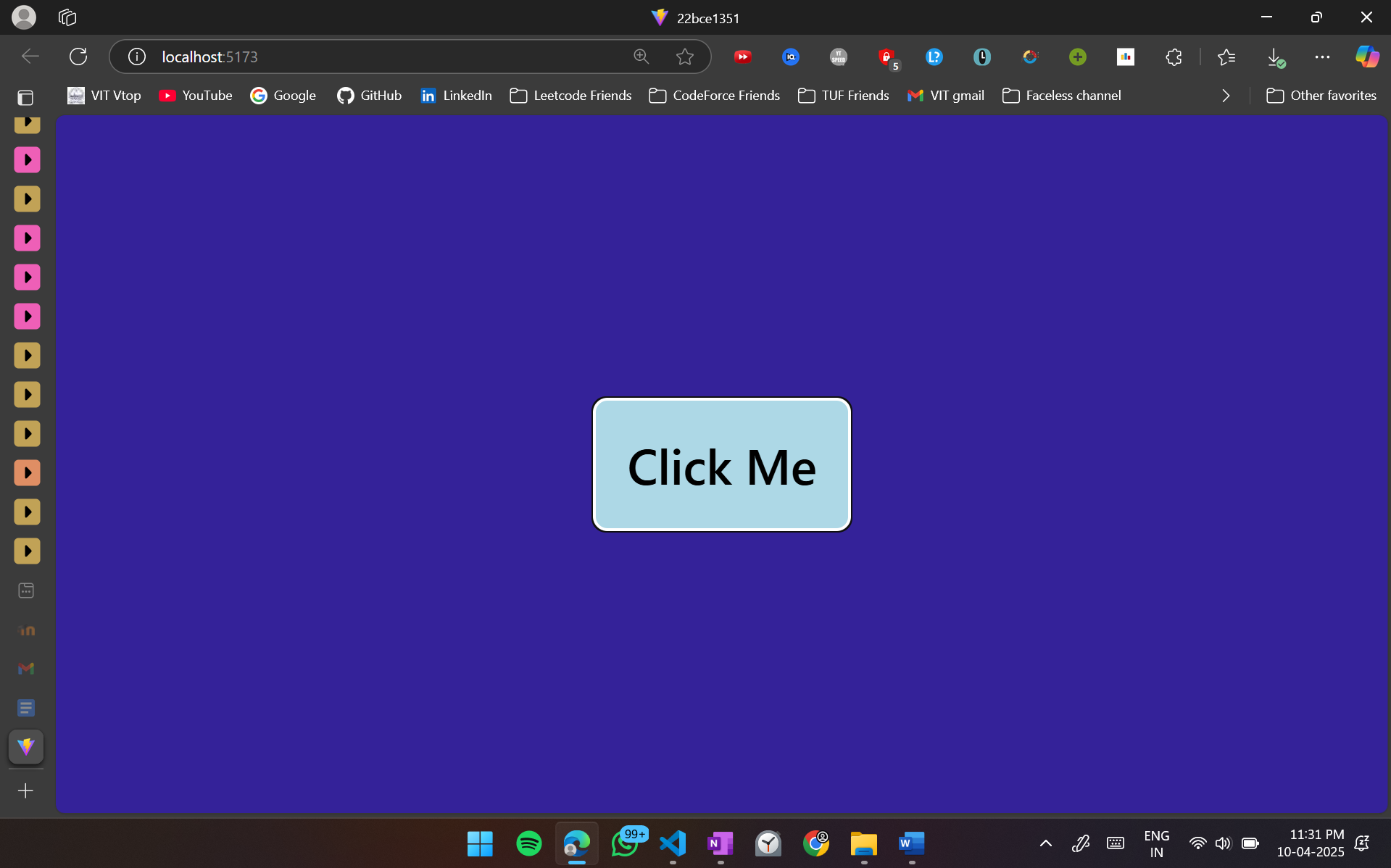
<StyledButton></StyledButton>

</>

)

}

export default App

****

**4. Styling in React – External CSS:**

* **Create a separate styles.css file and apply external styling to the StyledButton component by importing the CSS file.**

1. StyledButton.jsx

import '../Button/style.css'

function StyledButton(){

*return*(

<button className='button'>Click Me</button>

);

}

export default StyledButton;

1. style.css

.button {

background-color:lightblue;

padding:20px;

color:black;

font-size:30px;

}

1. App.jsx

import './App.css'

import StyledButton from './Button/StyledButton'

function App() {

*return* (

<>

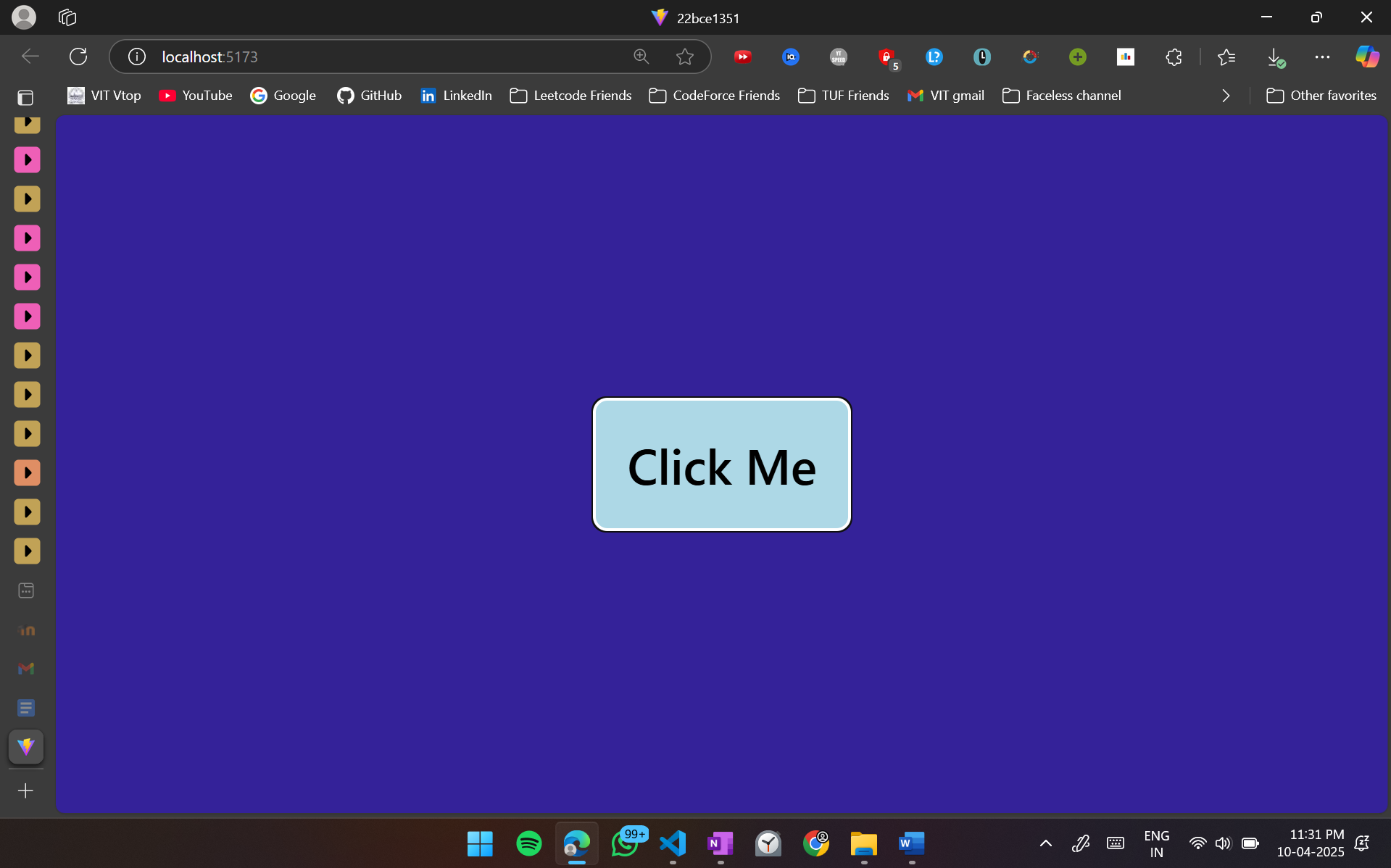
<StyledButton></StyledButton>

</>

)

}

export default App

****

**5. Develop a LifecycleDemo class component that logs messages at each stage of its lifecycle**

**o Lifecycle (constructor, componentDidMount, componentDidUpdate, and**

**componentWillUnmount).**

**o Implement a button to update the state and trigger componentDidUpdate().**

**o Unmount the component dynamically to observe the effect of**

**componentWillUnmount()**

1. LifecycleDemo.jsx

import React, { Component } from "react";

class LifecycleDemo *extends* Component {

constructor(props) {

super(props);

this.state = { counter: 0 };

console.log("LifecycleDemo: constructor");

}

componentDidMount() {

console.log("LifecycleDemo: componentDidMount");

}

componentDidUpdate(prevProps, prevState) {

console.log("LifecycleDemo: componentDidUpdate");

}

componentWillUnmount() {

console.log("LifecycleDemo: componentWillUnmount");

}

increment = () => {

this.setState((prevState) => ({ counter: prevState.counter + 1 }));

};

render() {

console.log("LifecycleDemo: render");

*return* (

<div style={{ border: "1px solid black", padding: "10px", margin: "10px" }}>

<h2>Lifecycle Demo</h2>

<p>Counter: {this.state.counter}</p>

<button onClick={this.increment}>Increment</button>

</div>

);

}

}

export default LifecycleDemo;

1. App.jsx

import { useState } from "react";

import LifecycleDemo from "./LifecycleDemo";

import "./App.css";

function App() {

*const* [showDemo, setShowDemo] = useState(true);

*const* toggleDemo = () => {

setShowDemo((prev) => !prev);

};

*return* (

<>

<button onClick={toggleDemo}>

{showDemo ? "Unmount LifecycleDemo" : "Mount LifecycleDemo"}

</button>

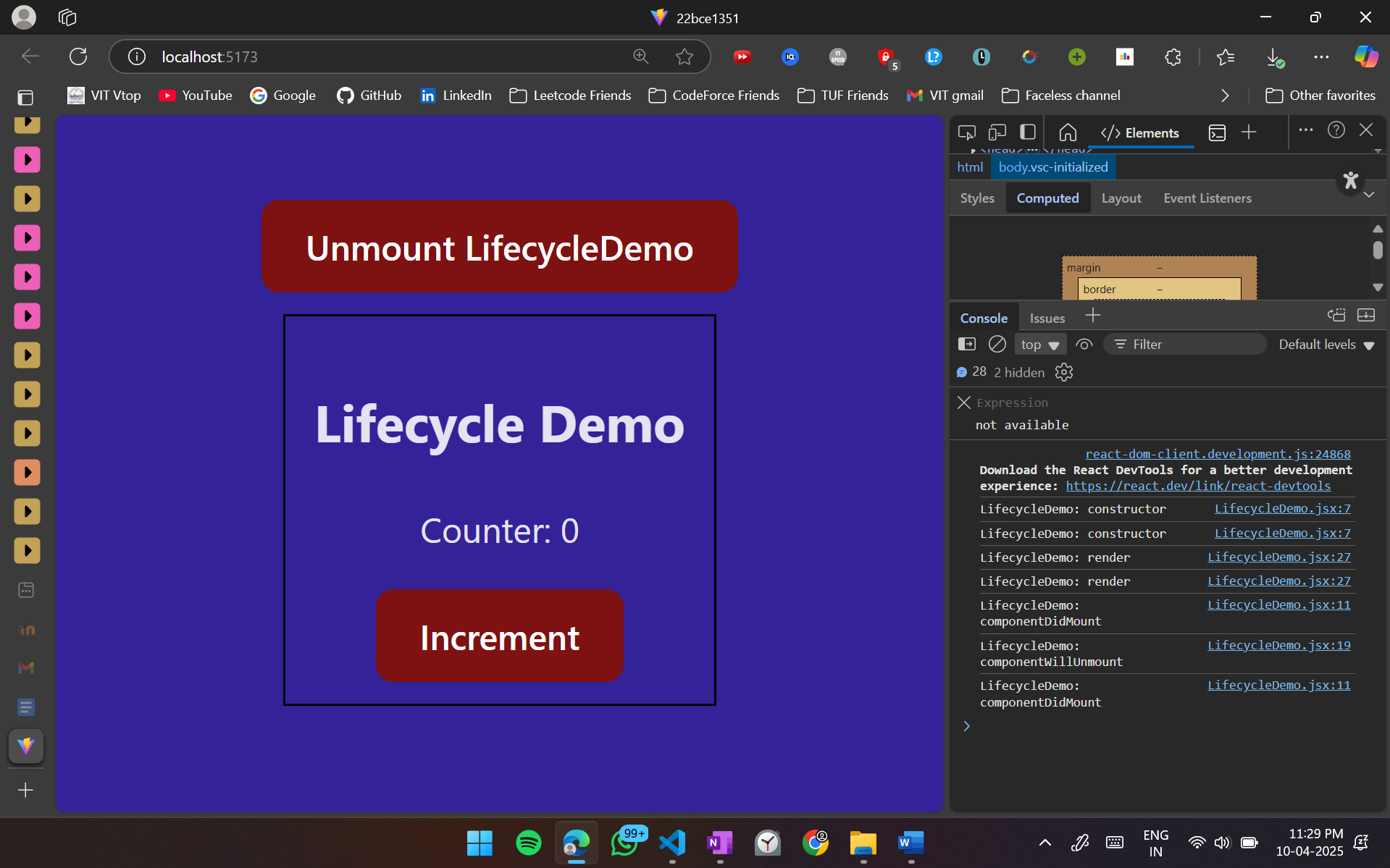
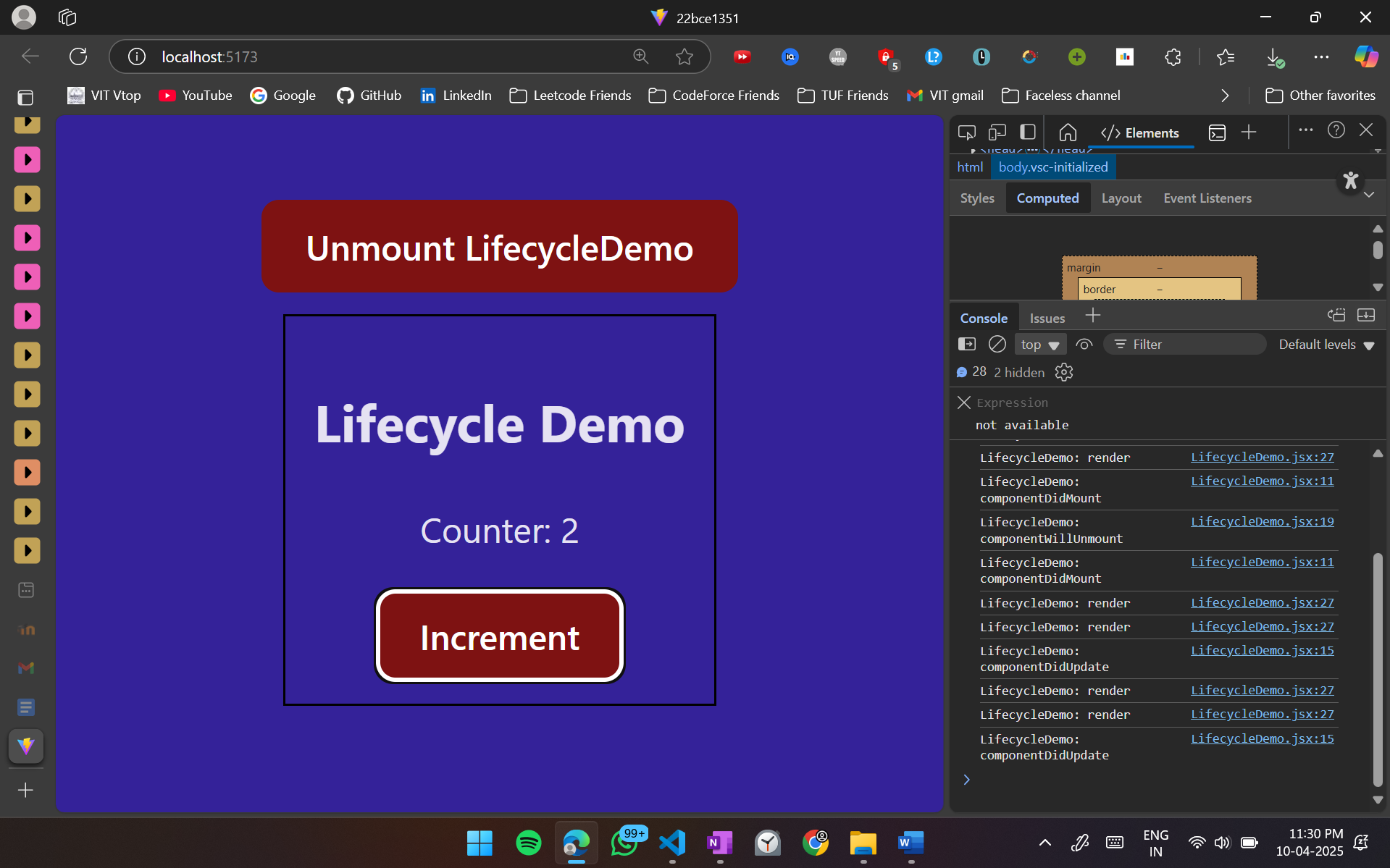
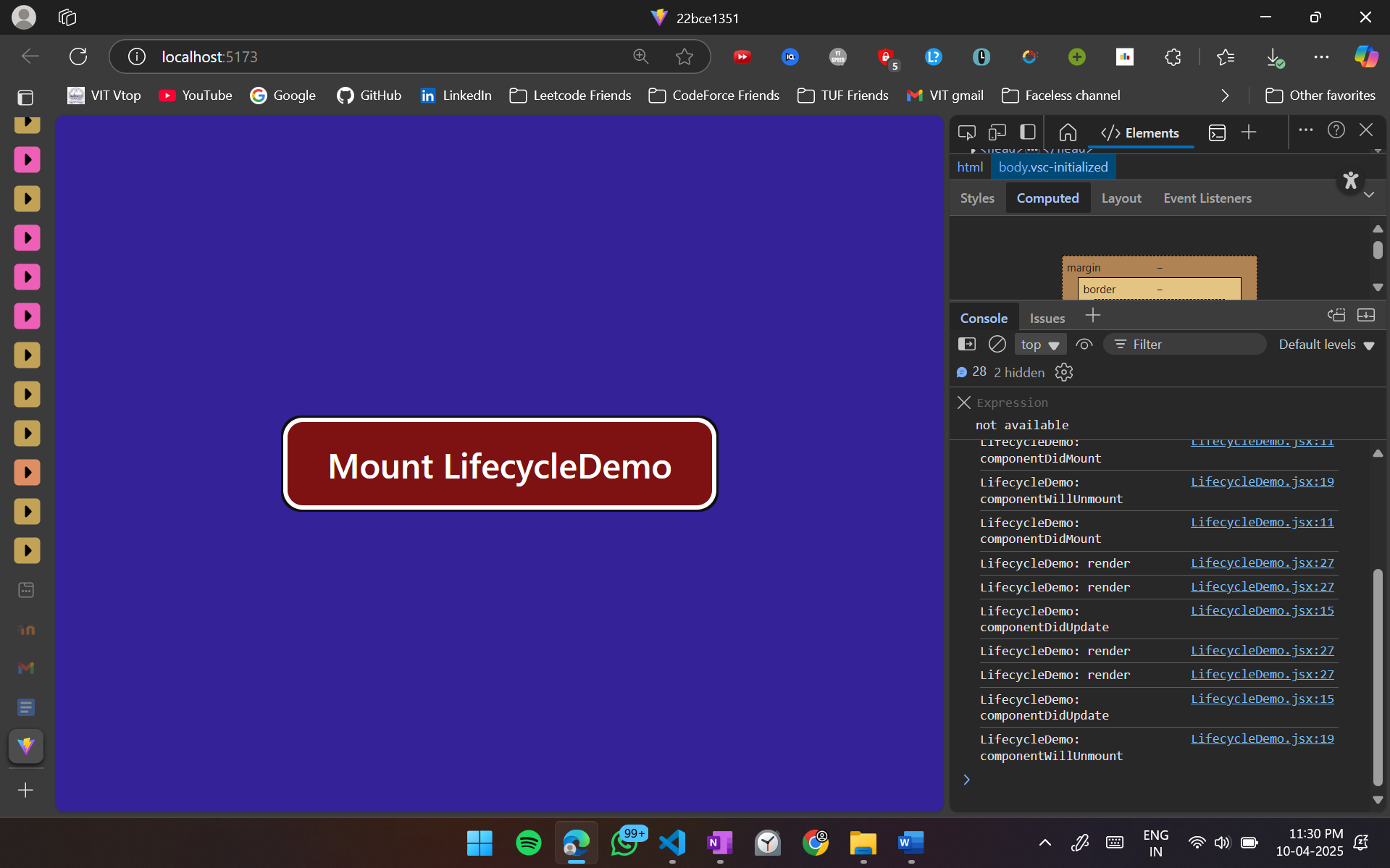
{showDemo && <LifecycleDemo />}

</>

);

}

export default App;

  **6. React Props:**

* **Design a Parent component that sends a message prop to a Child component.**
* **Ensure the Child component properly receives and displays the message.**

1. Parent.jsx

import Child from "./Child";

function Parent() {

*return* (

<div>

<h2>Parent Component</h2>

<Child message="Hello from the Parent!" />

</div>

);

}

export default Parent;

1. Child.jsx

function Child({ message }) {

*return* <p>{message}</p>;

}

export default Child;

1. App.jsx

import "./App.css";

import Parent from "./Parent";

function App() {

*return* (

<>

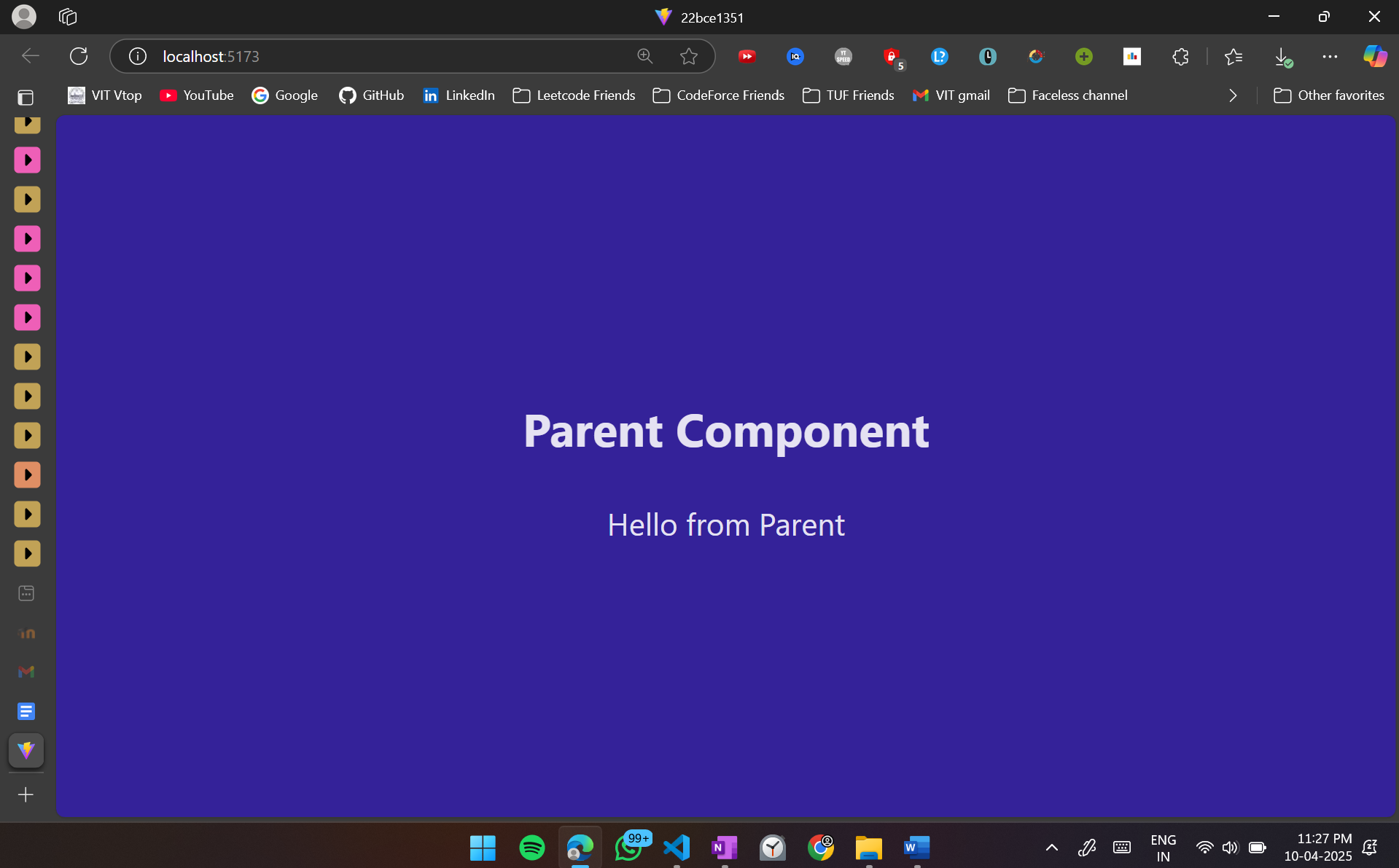
<Parent></Parent>

</>

);

}

export default App;



**7. React Props Validation:**

* **Modify the Child component to validate the message prop using prop-types.**
* **Ensure that the prop is required and of type string.**

1. Parent.jsx

import Child from "./Child";

function Parent() {

*return* (

<div>

<h2>Parent Component</h2>

<Child message="Hello from the Parent!" />

</div>

);

}

export default Parent;

1. Child.jsx

import PropTypes from 'prop-types'

function Child({ message }) {

*return* <p>{message}</p>;

}

Child.propTypes = {

message: PropTypes.string

}

export default Child;

1. App.jsx

import "./App.css";

import Parent from "./Parent";

function App() {

*return* (

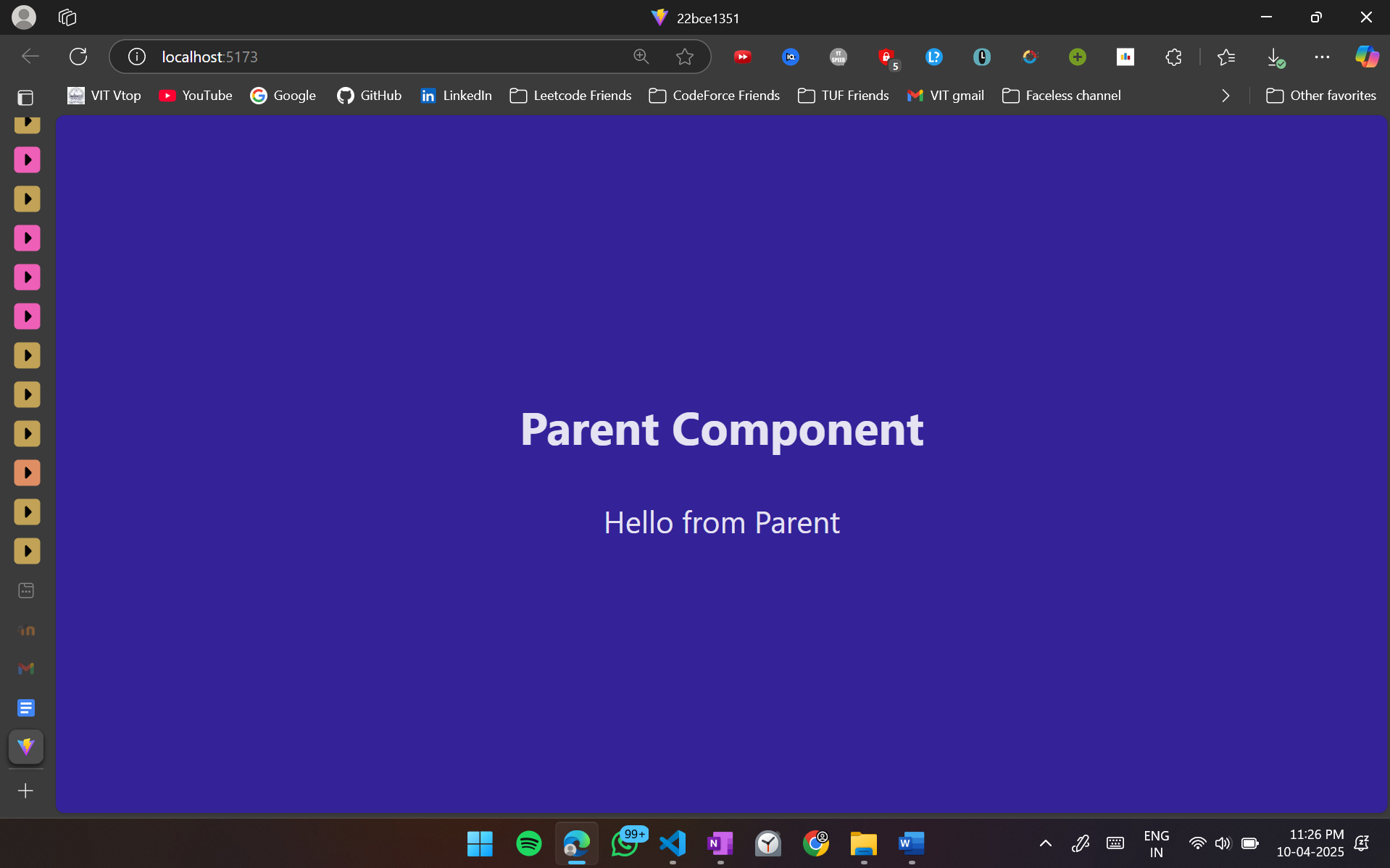
<>

<Parent></Parent>

</>

);

}

export default App;

**8. State Hooks: (useState and useReducer)**

* **Create a React component called Counter using the useState() hook. The component should display a count with two buttons: Increase and Decrease.**
* **Modify the component to use the useReducer() hook instead of useState(), handling increment and decrement actions efficiently.**

1. Counter.jsx using useState()

import React,{useState} from 'react';

function Counter(){

*const* [count,setCount] = useState(0);

*const* incrementCount = ()=>{

setCount(count+1);

}

*const* decrementCount = ()=>{

setCount(c=>c-1);

}

*return*(

<div>

<h1>{count}</h1>

<div>

<button onClick={incrementCount} style={{margin:"10px"}}>Increment Count</button>

<button onClick={decrementCount}>Decrement Count</button>

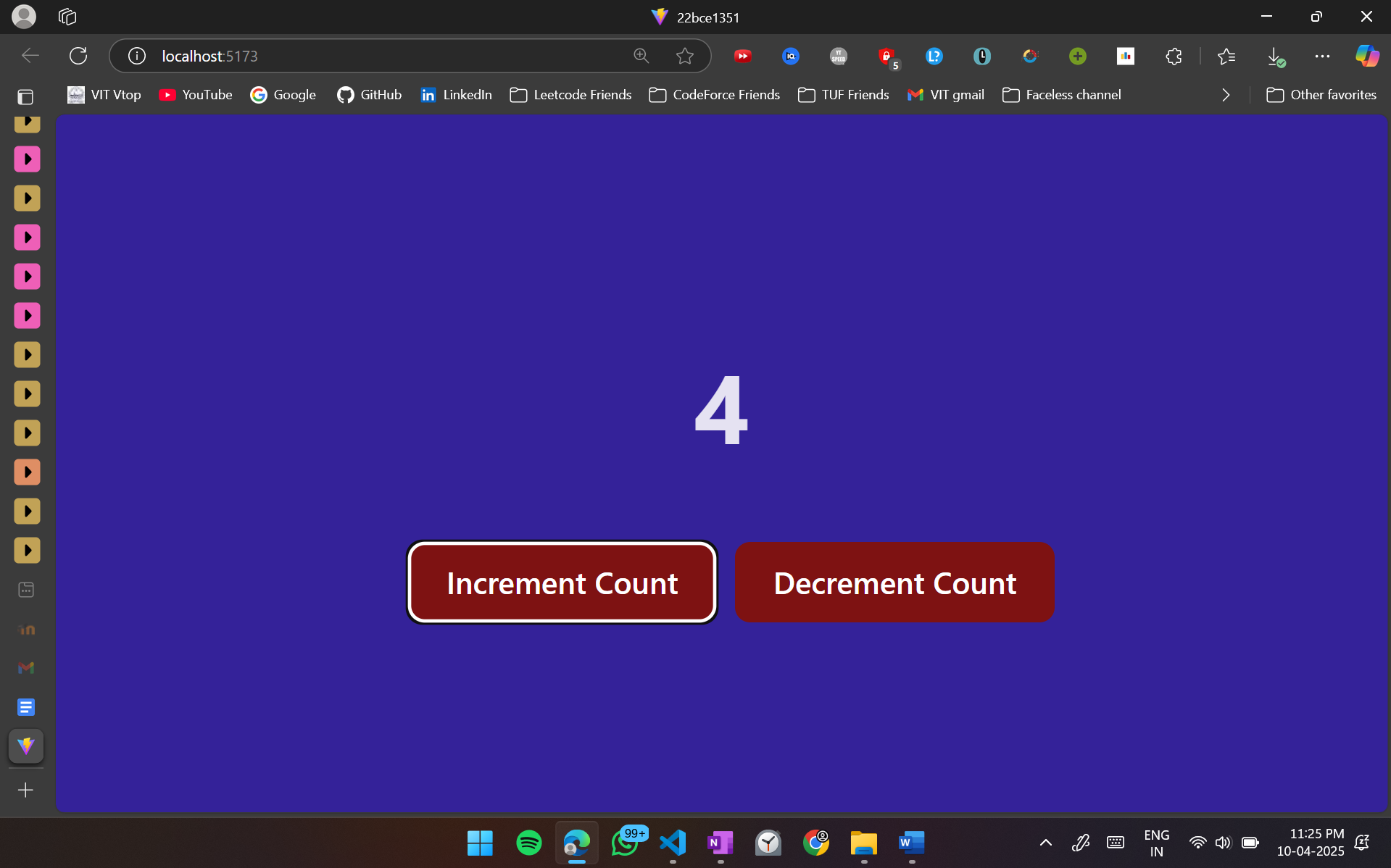
</div>

</div>

);

}

export default Counter;



1. Counter.jsx using useReducer

import React,{useReducer} from 'react';

function Counter(){

*const* reducer = (state,action)=>{

switch (action){

case "increment":

*return* state+1;

case "decrement":

*return* state-1;

default:

*return* state;

}

}

*const* [count,dispatch] = useReducer(reducer, 0);

*return*(

<div>

<h1>{count}</h1>

<div>

<button onClick={()=>dispatch("increment")} style={{margin:"10px"}}>Increment Count</button>

<button onClick={()=>dispatch("decrement")}>Decrement Count</button>

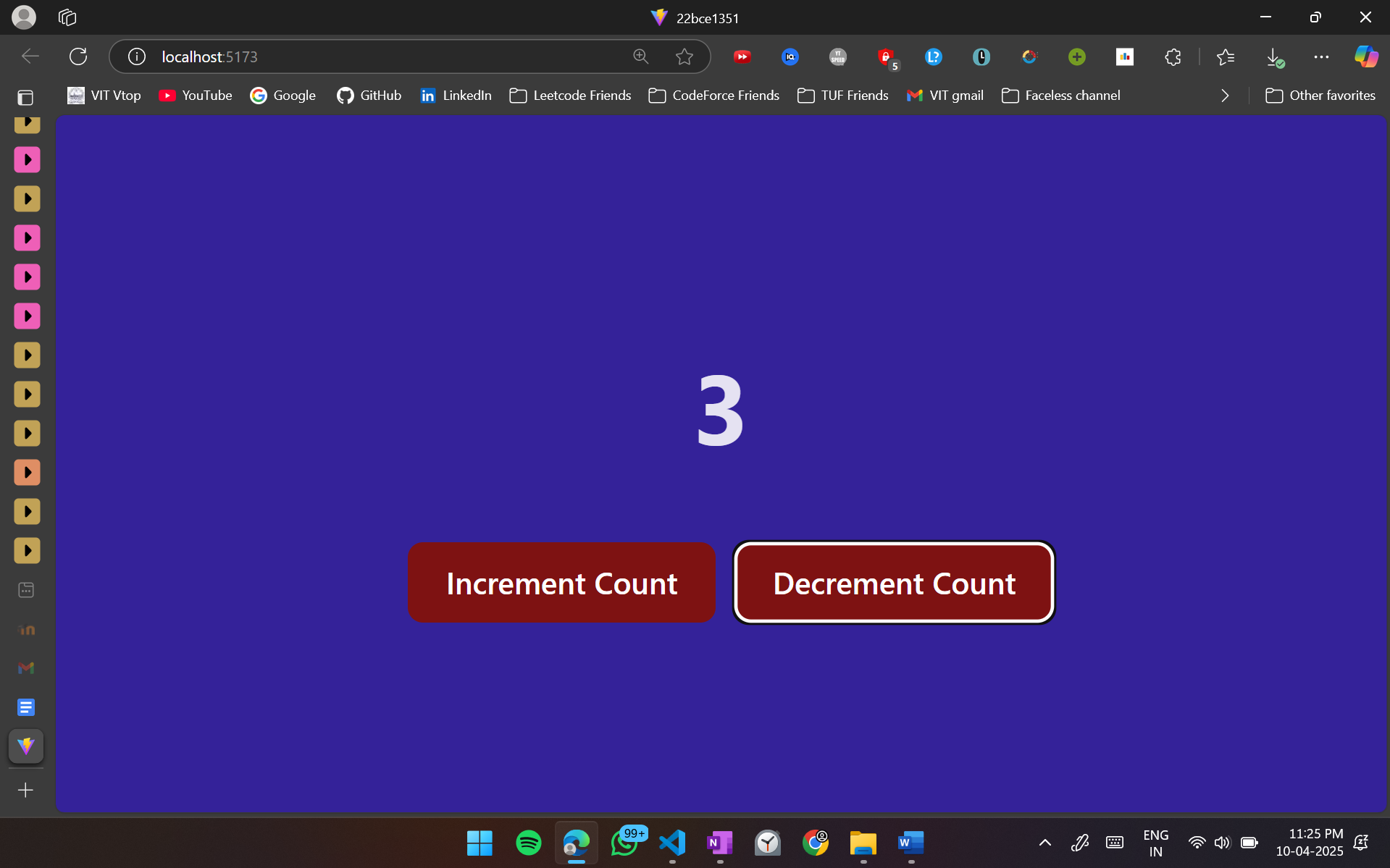
</div>

</div>

);

}

export default Counter;



**9. Effect Hooks (useEffect):**

* **Develop a React component that fetches and displays a random joke from an API when the component mounts.**
* **Add functionality to refresh the joke when a button is clicked.**

1. Joke.jsx

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

function Joke(){

*const* [joke,setJoke] = useState("");

*const* fetchJoke = *async*() => {

try {

*const* response = *await* fetch("https://official-joke-api.appspot.com/random\_joke");

*const* data = *await* response.json();

setJoke(`${data.setup} ${data.punchline}`);

} catch (error) {

console.error(error);

setJoke("Failed to fetch a joke.");

}

};

useEffect(()=>{

fetchJoke();

},[])

*return*(

<div>

<h2>{joke}</h2>

<button onClick={fetchJoke}>Refresh Joke</button>

</div>

);

}

export default Joke

1. App.jsx

import "./App.css";

import Joke from "./Joke";

function App() {

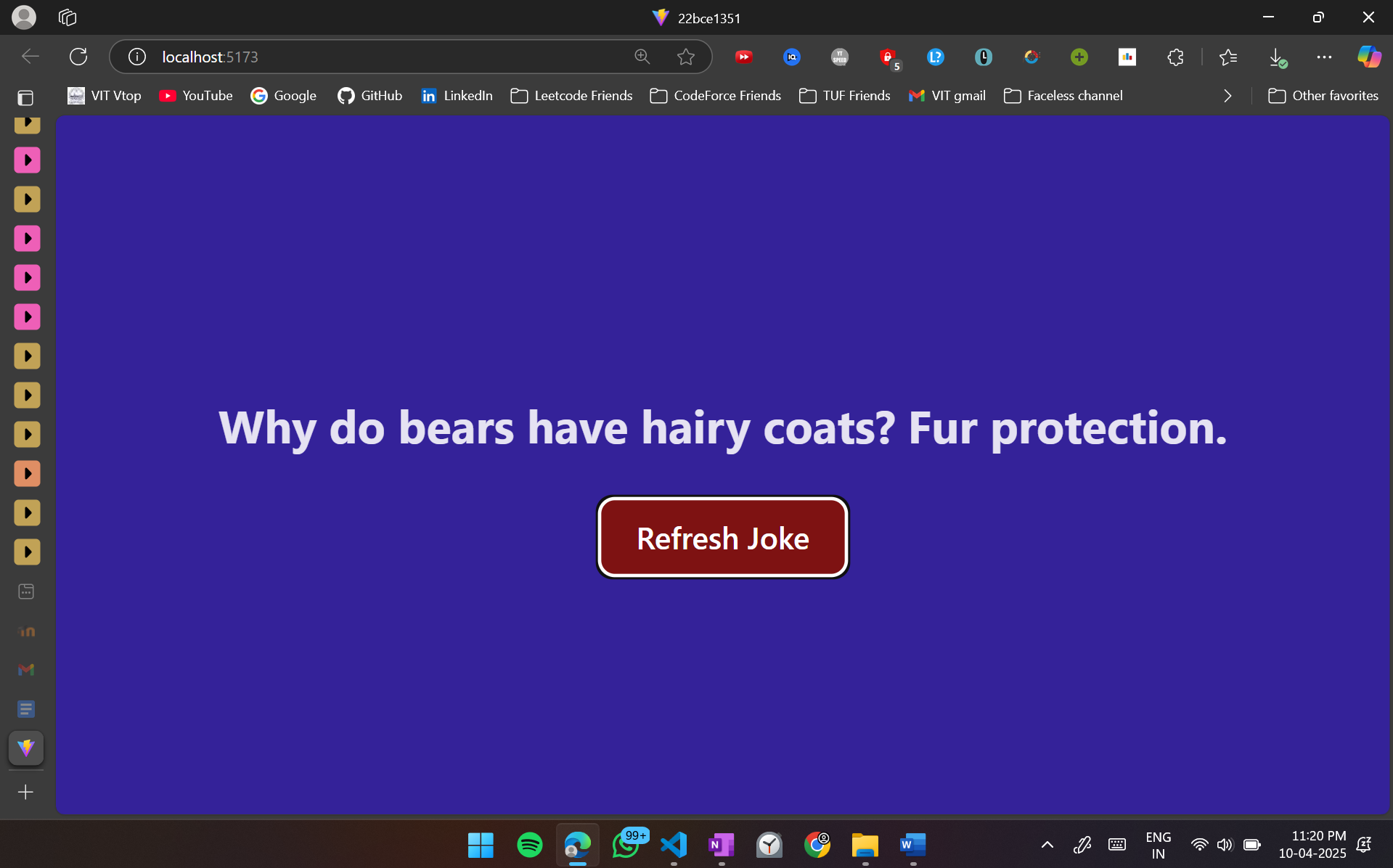
*return* (

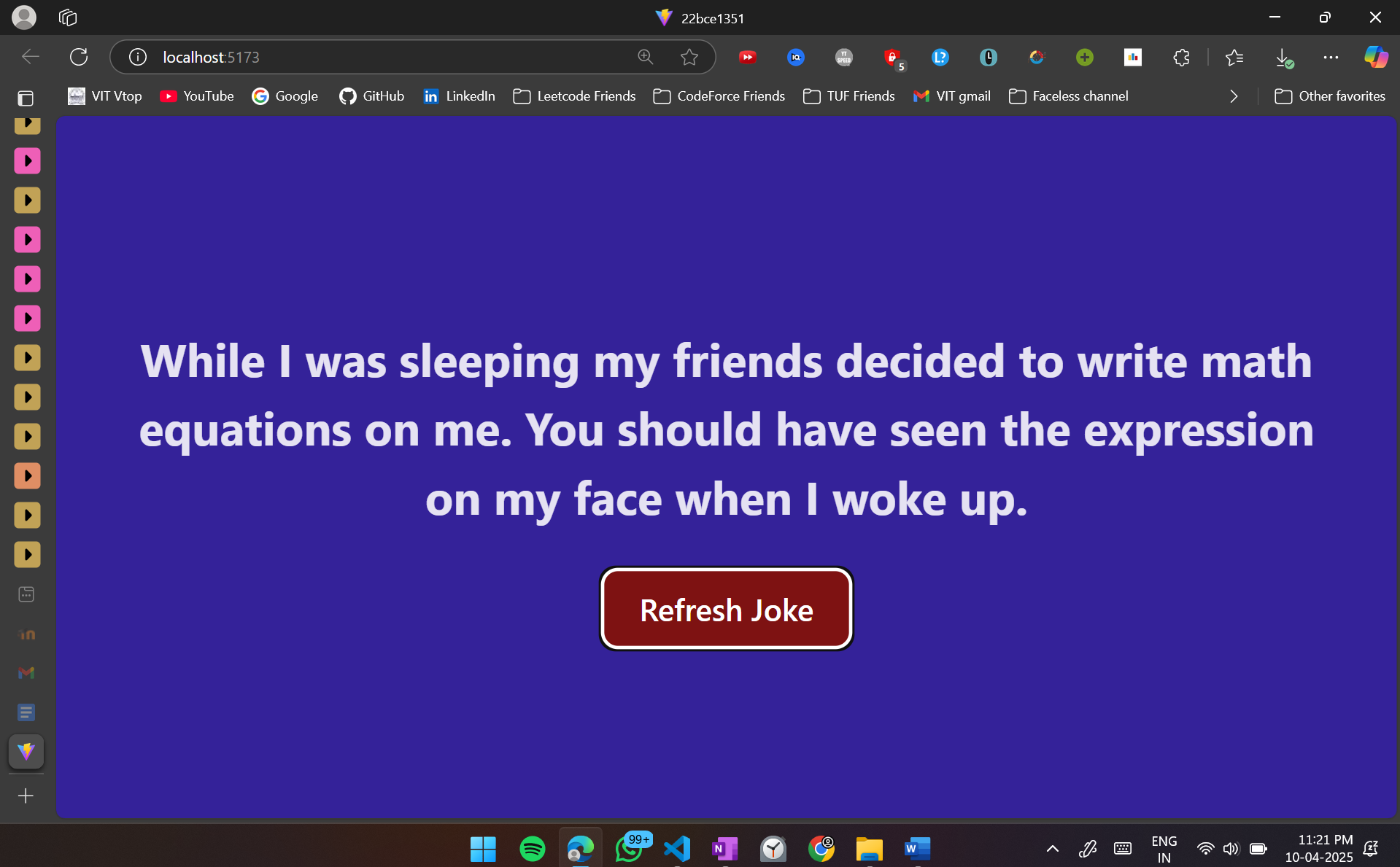
<Joke></Joke>

);

}

export default App;

****

****

**10. Ref Hooks (useRef):**

* **Build a simple form with an input field and a button.**
* **When the button is clicked, the input field should automatically get focused using the useRef() hook.**

1. Focus.jsx

import React,{useRef} from "react";

function Focus(){

*const* inputRef = useRef("");

*const* focusField = (inputRef)=>{

inputRef.current.focus();

}

*return*(

<div>

<button onClick={() => focusField(inputRef)} style={{margin:"10px"}}>Focus Field</button>

<input type="text" ref={inputRef}/>

</div>

);

}

export default Focus;

1. App.jsx

import "./App.css";

import Focus from "./Focus";

function App() {

*return* (

<>

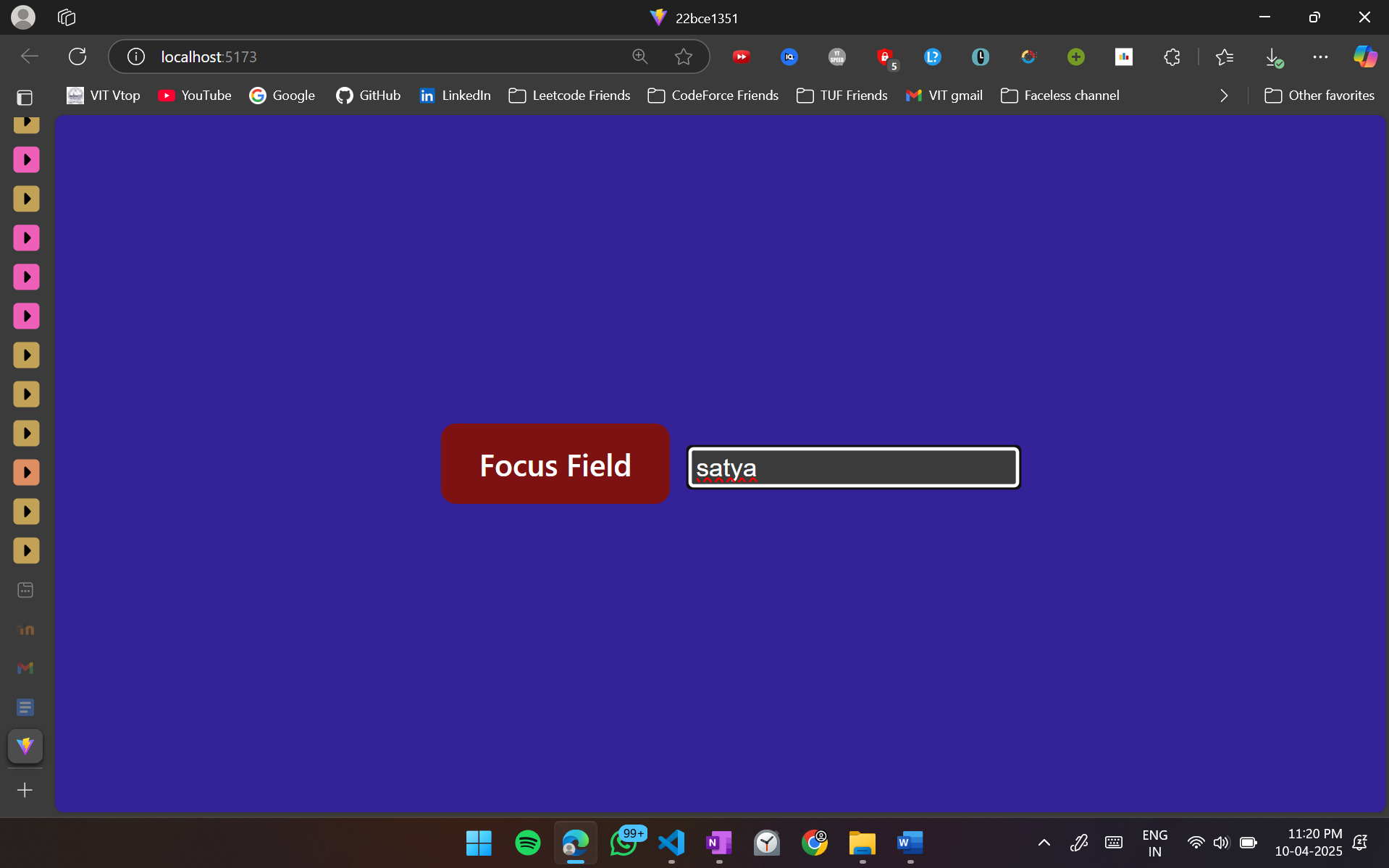
<Focus></Focus>

</>

);

}

export default App;

****

**11. Context Hooks (useContext):**

* **Create a React application where the theme (dark or light mode) is shared across multiple components using useContext().**
* **Implement a button to toggle between dark and light themes.**

ThemeContext.jsx

import React, { createContext, useState, useContext } from "react";

*const* ThemeContext = createContext();

export function ThemeProvider({ children }) {

*const* [theme, setTheme] = useState("light");

*const* toggleTheme = () =>

setTheme((prevTheme) => (prevTheme === "light" ? "dark" : "light"));

*return* (

<ThemeContext.Provider value={{ theme, toggleTheme }}>

{children}

</ThemeContext.Provider>

);

}

export function useTheme() {

*return* useContext(ThemeContext);

}

ThemeToggle.jsx

import React from "react";

import { useTheme } from "./ThemeContext";

function ThemeToggle() {

*const* { theme, toggleTheme } = useTheme();

*return* (

<button onClick={toggleTheme}>

Switch to {theme === "light" ? "dark" : "light"} mode

</button>

);

}

export default ThemeToggle;

ThemedContent.jsx

import React from "react";

import { useTheme } from "./ThemeContext";

function ThemedContent() {

*const* { theme } = useTheme();

*const* style = {

backgroundColor: theme === "light" ? "#fff" : "#333",

color: theme === "light" ? "#000" : "#fff",

padding: "20px",

marginTop: "10px",

};

*return* <div style={style}>This content is themed!</div>;

}

export default ThemedContent;

App.jsx

import "./App.css";

import { ThemeProvider } from "./ThemeContext";

import ThemeToggle from "./ThemeToggle";

import ThemedContent from "./ThemedContent";

function App() {

*return* (

<ThemeProvider>

<ThemeToggle />

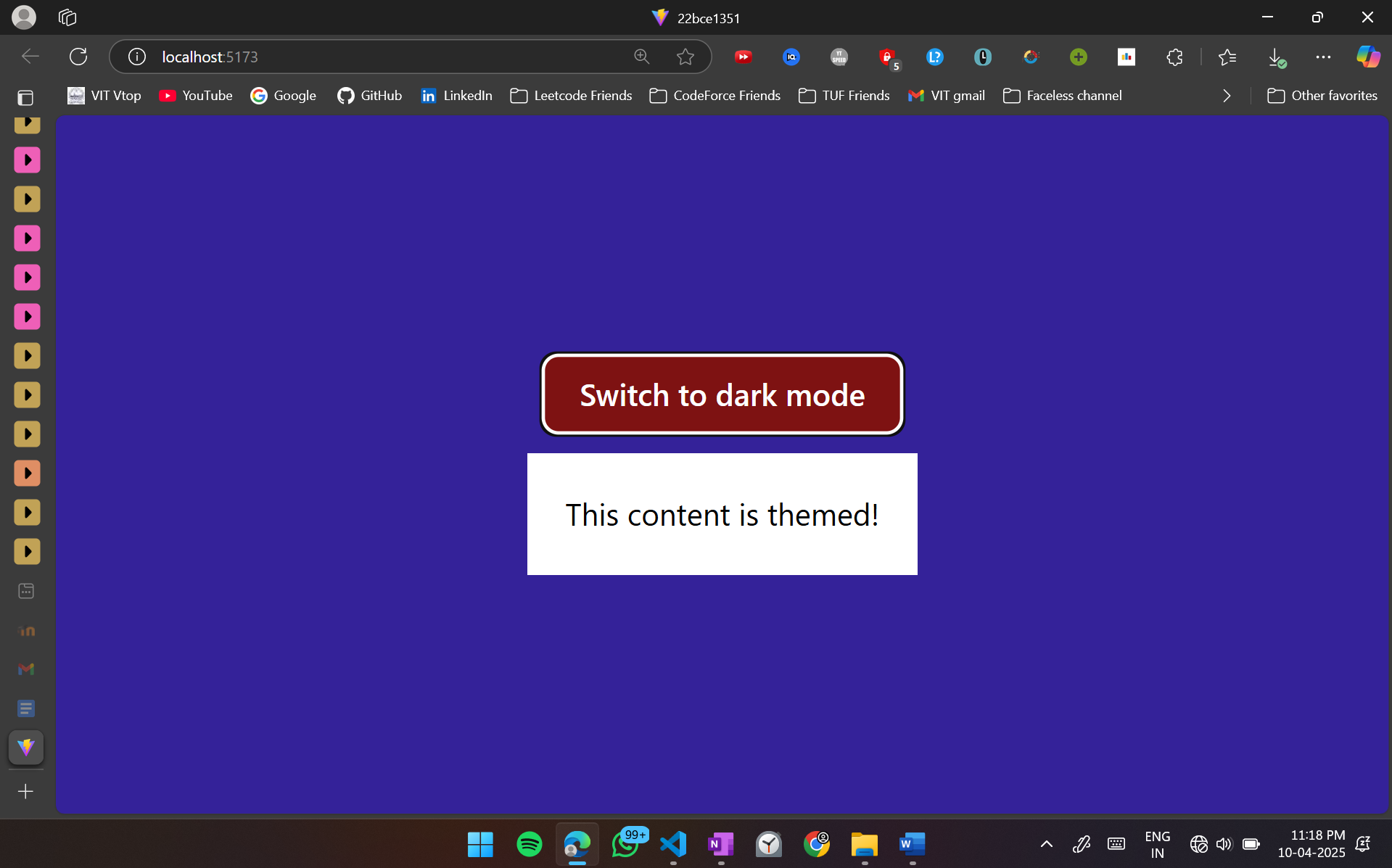
<ThemedContent />

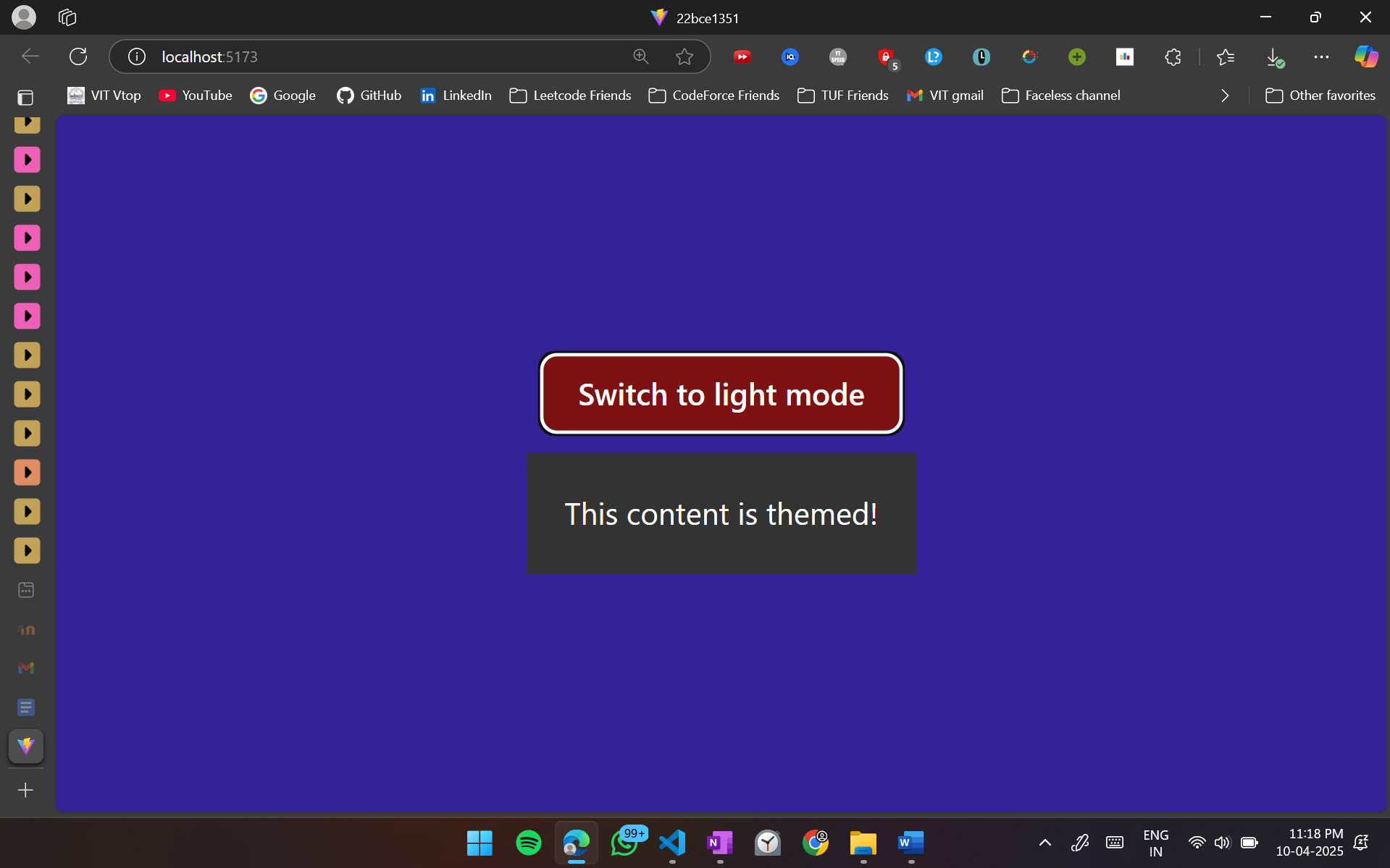
</ThemeProvider>

);

}

export default App;





**12. Passing Values from a Form Using useState and useRef(i)**

**(i) Create a form with fields for Name and Email. Use useState to manage input**

**values and display them dynamically.**

* **Create a new React component.**
* **Use useState to track form values.**
* **Display the values dynamically as the user types.**
* **Submit the form and prevent default page reload.**

Form.jsx

import { useState } from "react";

function Form() {

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

*const* [email, setEmail] = useState("");

*const* handleSubmit = (e) => {

e.preventDefault();

console.log("Submitted Name:", name);

console.log("Submitted Email:", email);

};

*return* (

<form onSubmit={handleSubmit}>

<label>

Name:

<input

type="text"

value={name}

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

/>

</label>

<br />

<br />

<label>

Email:

<input

type="email"

value={email}

onChange={(e) => setEmail(e.target.value)}

/>

</label>

<br />

<br />

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

<section>

<h3>Preview:</h3>

<p>Name: {name}</p>

<p>Email: {email}</p>

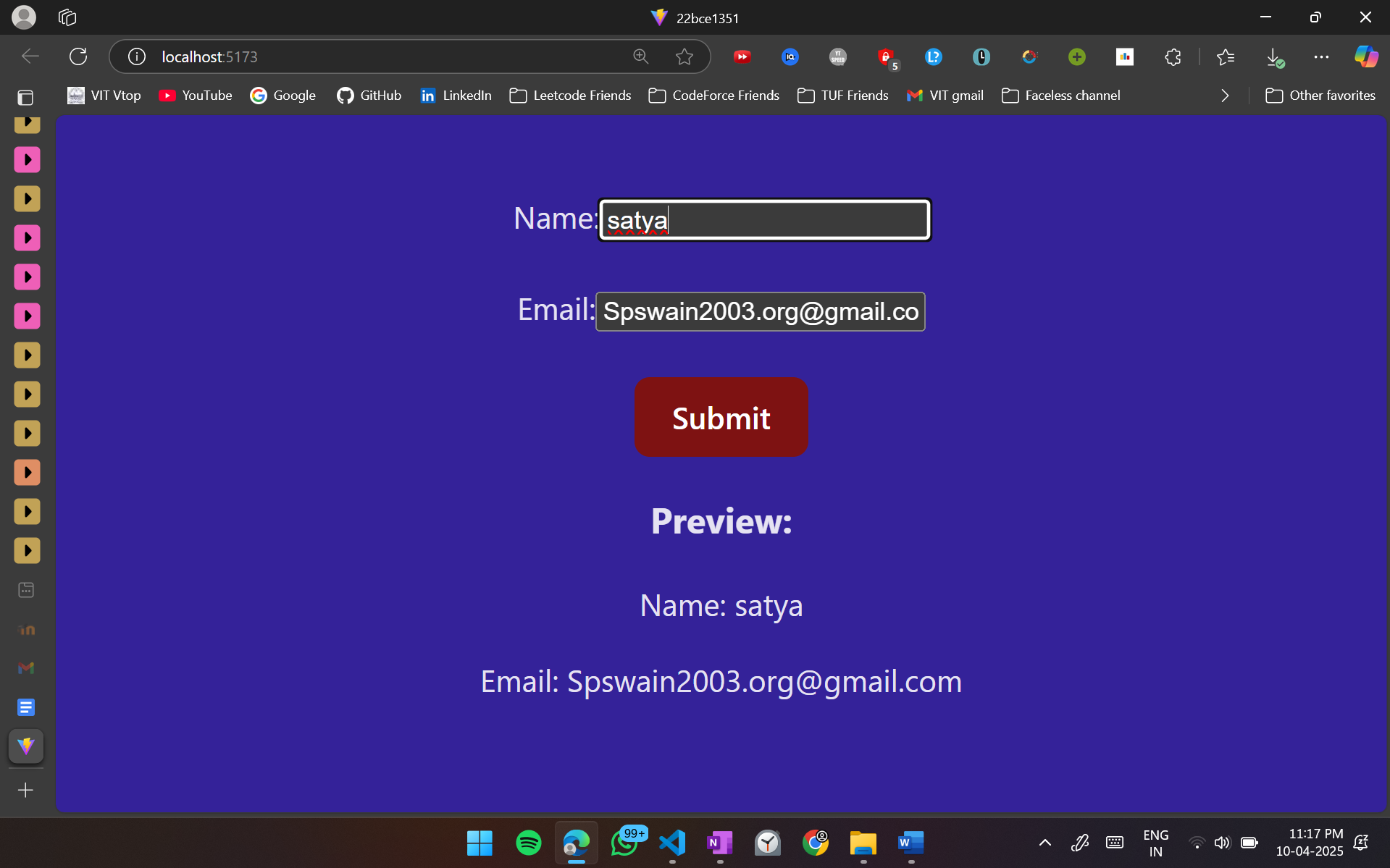
</section>

</form>

);

}

export default Form;

****

**(ii) Create the same form but use useRef to retrieve values on form submission**

**without managing state updates.**

* **Create a new React component.**
* **Use useRef to get form values.**
* **Display values only when the form is submitted.**

NewForm.jsx

import { useRef, useState } from "react";

function NewForm() {

*const* nameRef = useRef(null);

*const* emailRef = useRef(null);

*const* [submittedData, setSubmittedData] = useState(null);

*const* handleSubmit = (e) => {

e.preventDefault();

*const* name = nameRef.current.value;

*const* email = emailRef.current.value;

setSubmittedData({ name, email });

};

*return* (

<form onSubmit={handleSubmit}>

<label>

Name:{" "}

<input

type="text"

ref={nameRef}

placeholder="Enter your name"

/>

</label>

<br />

<br />

<label>

Email:{" "}

<input

type="email"

ref={emailRef}

placeholder="Enter your email"

/>

</label>

<br />

<br />

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

{submittedData && (

<section>

<h3>Preview:</h3>

<p>Name: {submittedData.name}</p>

<p>Email: {submittedData.email}</p>

</section>

)}

</form>

);

}

export default NewForm;

