1

Core react এবং javascript দিয়ে যদি একটি html পেইজে কিছু প্রিন্ট করতে চাই তবে প্রথমে আমাদেরকে একটি Html ফাইল নিতে হবে, সেখানে একটি div ট্যাগ নিতে হবে যার আইডি ইচ্ছামত একটা দিয়ে নেবো। লাইক এখানে আইডি দিলাম root .

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>

  <div id="root"></div>

</body>

</html>
```

এবার আমরা যেহেতু raw রিয়েক্ট লাইব্রেরী ব্যবহার করবো, সেহেতু রিয়েক্ট লাইব্রেরীর স্ক্রিপ্ট ফাইল আমাদেরকে ইম্পোর্ট করে নিতে হবে।

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <div id="root"></div>

  <script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>
  <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js" crossorigin></script>
  <script src="test.js"></script>
</body>
```

```
</html>
```

এখানে test.js নামে একটা স্ক্রিপ্ট ফাইল নিয়েছি যার ভেতর আমরা আমাদের সকল রিয়েক্ট কোডগুলো লিখবো। যেহেতু আমরা root নামক div এর ভেতর রিয়েক্ট লাইব্রেরী ব্যবহার করে আমাদের খুশিমত যা ইচ্ছা লিখবো, সেহেতু আমাদেরকে test.js স্ক্রিপ্ট ফাইলে প্রথমে root নামক ডিভটি ধরতে হবে,

```
let container = document.querySelector('#root');
```

এখন আমরা একটি রিয়েক্ট এলিমেন্ট তৈরি করবো। রিয়েক্ট এলিমেন্ট তৈরি করার ফাংশানটি হচ্ছে এরকম

```
let myElement = React.createElement('div', null, 'Hello World');
```

React.createElement ফাংশানের প্রথম প্যারামিটারে বলে দিতে হবে আমরা কি টাইপ ইলেমেন্ট তৈরি করবো। এখানে আমরা Div টাইপ ইলিমেন্ট বানাবো। তারপরেরটা null দিবো কারণ আপাতত সেখানে কোন ভ্যালু সেট করছিনা। তারপরের প্যারামিটারে আমাদেরকে বলে দিতে হবে আমরা ওই ইলেমেন্ট এর ভেতর কোন কন্টেন্ট প্রিন্ট করবো।

এখন আমরা react dom দিয়ে এই ইলেমেন্টকে root নামক div এর ভেতর প্রিন্ট করবো বা রেন্ডার করবো।

```
let container = document.querySelector('#root');

let myElement = React.createElement('div', null, 'Hello World');

ReactDOM.render(myElement, container);
```

এটি আমাদের html পেইজে ‘Hello World’ প্রিন্ট করবে। এখন আমরা ‘Hello World’ এর পরিবর্তে যদি <p> ট্যাগ দিয়ে কিছু প্রিন্ট করতে চাই তবে তার কোড হবে এরকম ,

```
let container = document.querySelector('#root');

let myElement = React.createElement('div', null, React.createElement('p', null, 'Hello world from Paragraph'));

ReactDOM.render(myElement, container);
```

যদি আমরা root ডিভের ভেতর একাধিক p ট্যাগ দিয়ে একাধিক কিছু প্রিন্ট করতে চাই তবে সেক্ষেত্রে root ডিভের জন্য যে ইলেমেন্টটি তৈরি করেছি তার তৃতীয় প্যারামিটারে একটি array নিতে হবে এবং সেই এরের ভেতর p ইলেমেন্টগুলো তৈরি করে নিতে হবে,

```
let container = document.querySelector('#root') ;

let myElement = React.createElement('div', null,[
   React.createElement('p', null, 'Hello world from Paragraph'),
   React.createElement('p', null, 'Hello world from Paragraph 2'),
   React.createElement('p', null, 'Hello world from Paragraph 3'),
]) ;

ReactDOM.render(myElement, container) ;
```

এভাবেই রিয়েক্ট দিয়ে এলিমেন্ট তৈরি করে আমরা html পেইজে সকল প্রকার html ট্যাগ ব্যবহার করে আমরা যেকোন কিছু দেখাতে পারি।

English Title : How to show simple 'Hello World ' with Core React in a html page ?
Bangla Title : রিয়েক্ট লাইব্রেরী দিয়ে একটি html পেইজে কিভাবে একটি সিম্পল Hello World লেখা প্যারাগ্রাফ প্রিন্ট করতে পারি।

2

```
import React from 'react';
import ReactDOM from 'react-dom'


function Test({user}){
  return (
    <h2>Hello Testing {user}</h2>
  )
}

ReactDOM.render(<Test user="russell" />, document.getElementById('root'));
```

3

```
import React from 'react';
import ReactDOM from 'react-dom'

class Test{
  print(){
```

```
    return(
      <h2>Hello World </h2>
    )
  }
}

const Testcomponent = new Test();
ReactDOM.render(Testcomponent.print(), document.getElementById('root'));
```

4

React দিয়ে স্ক্রীনে একটি hello world প্রিন্ট করতে হলে আমাদেরকে প্রথমে একটি JSX ইলেমেন্ট বানাতে হবে এরকম করেঃ

```
const dummy = (
  <h2 className="storng-text">Hello World</h2>
)
```

এটা মূলত ব্যাবেল দিয়ে ট্রান্সপাইল হয়ে অবজেক্ট আকারে জাভাস্ক্রিপ্টের কাছে যায় এবং স্ক্রীনে প্রিন্ট করেঃ

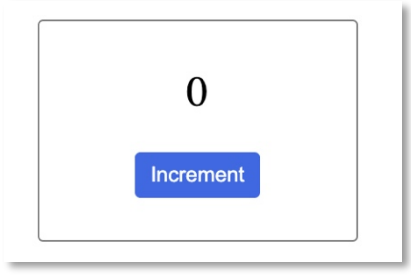
```
const dummy = {
  type : 'h2',
  props : {
    className : 'strong-text',
    children : 'Hello World'
  }
};
```

এই অবজেক্টটাই হলো রিয়েক্ট *ইলেমেন্ট।* এবং ইলেমেন্ট হচ্ছে জাভাস্ক্রিপ্ট অবজেক্ট।

5

কোন একটি ওয়েব সাইটে কোন একটি কাজ বারবার করা লাগতে পারে। হতে পারে কোন একটি ডিজাইনের ডিভ অথবা কোন একটি ফ্রন্টেন্ড ফাংশনালিটি। Raw জাভাস্ক্রিপ্ট দিয়ে সেই কাজটি করতে হলে বারবার ফাংশান লিখতে হবে, বারবার নতুন করে ডিজাইন করতে হবে, বারবার সেই ডিভের ইলেমেন্টগুলোকে DOM দিয়ে ধরতে হবে। বড় ওয়েবসাইট বা এপ্লিকেশানে এটা খুবই প্যারাদায়ক একটি কাজ। বারবার না করে জাস্ট একবার করে বারবার সেটাকে ইউজ করার জন্যই রিয়েক্ট এর জন্ম।

যেমন আমরা একটি ছোট্ট counter এপ্লিকেশান বানাতে চাই। যেখানে একটি বাটনে চাপ দিলে ১ করে নাম্বার বাড়তে থাকবে।



https://i.ibb.co/820F5KX/Xnip2023-09-21-21-15-50.jpg

এখন এটি প্লেইন জাভাস্ক্রিপ্ট দিয়ে করতে হলে প্রথমে একটি html ডকুমেন্টে প্রয়োজনীয় html structure বানিয়ে নিতে হবে,

https://i.ibb.co/4TKMNqX/code.png

এরপরে এটির জন্য css লিখতে হবে , এরপরে একটি স্ক্রিপ্ট ফাইলে এর ফাংশনালিটির জন্য প্রয়োজনীয় জাভাস্ক্রিপ্ট কোড গুলো লিখতে হবে।

https://i.ibb.co/JFHNf5b/zsdf.jpg

এখানে প্রথমে বাটন এবং যে h ট্যাগের মাধ্যমে সংখ্যাটি দেখাবো সেটিকে Dom দিয়ে ধরতে হবে। এরপরে একটি ভ্যারিয়াবল নিতে হবে। এরপরে বাটনে একটি ক্লিক event listener এড করে তারপরে সংখ্যা বাড়ানোর কাজটি করতে হবে। প্রতিবার বাটনে ক্লিক হলে number নামক ভ্যারিয়েবলটির মান এক করে বাড়বে এবং সেই মানটিকে ডম দিয়ে ম্যানিপুলেট করে text নামক h2 ট্যাগে দেখানো হবে।

এখন এই কাজটি যদি কয়েকবার করতে হয়, তবে আমাদেরকে প্রতিবার নতুন করে html ইলেমেন্ট লিখতে হবে। নতুন করে css লিখতে হবে। এবং নতুন করে ডম দিয়ে html element গুলোকে ধরতে হবে এবং আলাদা করে আবার স্ক্রিপ্ট লিখতে হবে। নিচের ছবির মত

https://i.ibb.co/qCHPX1q/Xnip2023-09-21-22-14-35.jpg
html : https://i.ibb.co/jZWYJ31/4.jpg
js : https://i.ibb.co/wQBb4Q1/5.jpg

এটি একটি প্যারাদায়ক কাজ। রিয়েক্ট এসে এই সমস্যার সমাধান করেছে। রিয়েক্ট এর কারণে একই কাজের জন্য html, css এবং javascript জাস্ট একবার লিখেই যতবার ইচ্ছা ততবার ব্যবহার করা যাবে নতুন করে কোড না লিখে। উপড়ের কোডগুলোই যদি রিয়েক্টে লিখতাম ,

Html : https://i.ibb.co/mJFjqNn/6.jpg
Js : https://i.ibb.co/1b1cJc3/7.jpg

```
ReactDOM.render(<>
    <Element />
    <Element />
    <Element />
</>, main)
```

ReactDOM.render() ফাংশান দ্বারা main আইডি নামক ডিভের ভেতর আমরা একই ডিভ element যতবার ইচ্ছা ততবার ব্যবহার করতে পারি। এতে করে element ডিভটি আমাদের মাত্র একবার তৈরি করে নিলেই হয়ে যাচ্ছে।

6

Core React দিয়ে একটি স্ক্রিণে একটি hello world প্রিন্ট করা

7

useState এর ভেতর অবজেক্ট ব্যবহার করার প্রাকটিক্যাল এক্সাম্পলঃ

(Note : এখানে typescript ব্যবহার করা হয়েছে)

```tsx
// using object inside usestate
  type TUserProfile = {
    name : string;
    age : number;
    profession : string
  }
  const userProfile : TUserProfile = {
    name : 'Anonymouse',
    age : 21,
    profession : 'toto'
  }
  const [user, setUser] = useState(userProfile);

const handleSubmit = (e : React.ChangeEvent<HTMLFormElement>) => {
  e.preventDefault();
}
```

```tsx
    <form className="my-5" onSubmit={handleSubmit}>
      <input
        onChange={(e) => setUser({ ...user, name: e.target.value })}
        className="border-2"
        type="text"
        name="name"
        id=""
      />
      <input
        onChange={(e) => setUser({ ...user, age: e.target.value })}
        className="border-2 ms-3"
        type="number"
        name="age"
        id=""
      />
      <button className="btn-sm btn ms-3 btn-primary" type="submit">
        Submit
      </button>
    </form>
```

এই কোডটি আরো প্রফেশনাল ওয়েতে করা যেতে পারে। মানে Clean Code এবং Dry(Don't repeat yourself) প্রিন্সিপাল ফলো করে।

```
// using object inside usestate
  type TUserProfile = {
    name: string;
    age: number;
    profession: string;
  };
  const userProfile: TUserProfile = {
    name: "Anonymouse",
    age: 21,
    profession: "toto",
  };
  const [user, setUser] = useState(userProfile);

  const handleSubmit = (e: React.ChangeEvent<HTMLFormElement>) => {
    e.preventDefault();
  };
```

```
const handleChange = (e : React.ChangeEvent<HTMLInputElement>, inputElement) => {
  const value = e.target.value;
  const targetName = e.target.name ;
  setUser({...user, [targetName] : value});
}
```

```
      <form className="my-5" onSubmit={handleSubmit}>
        <input
          onChange={(e) => handleChange(e, e.target.name)}
          className="border-2"
          type="text"
          name="name"
          id=""
        />
        <input
          onChange={(e) => handleChange(e, e.target.name)}
          className="border-2 ms-3"
          type="number"
          name="age"
          id=""
        />
        <button className="btn-sm btn ms-3 btn-primary" type="submit">
          Submit
        </button>
      </form>
```

8

useReducer এর একটি ব্যাসিক এক্সাম্পলঃ

```jsx
const initialState = { count: 50 };

const reducerFunction = (state, action) => {
  switch (action.type) {
    case "increase":
      return { count: state.count + 1 };

    case "decrease":
      return { count: state.count - 1 };

    default:
      return state;
  }
};

const UseReducer4 = () => {
  const [state, dispatch] = useReducer(reducerFunction, initialState);
  return (
    <div>
      <h2>Use Reducer 3</h2>
      <h1>{state.count}</h1>
      <button
        onClick={() => dispatch({ type: "increase" })}
        className="btn btn-neutral me-4"
        type="button"
      >
        Increase
      </button>
      <button
        onClick={() => dispatch({ type: "decrease" })}
        className="btn btn-warning me-4"
        type="button"
      >
        Decrease
      </button>
    </div>
  );
};

export default UseReducer4;
```


useReducer এর সাথে payload ব্যবহার:

```jsx
import { useReducer } from "react";


const initialState = {
  name: "Rasel",
  email: "rasel@gmail.com",
};

const reducerFunction = (currentState: typeof initialState, action) => {
  switch (action.type) {
    case "name":
      return { ...currentState, name: action.payload };

    case "email":
      return { ...currentState, email: action.payload };

    default:
      return currentState ;
  }
};
```

```
const ReducerForm = () => {
  const [state, dispatch] = useReducer(reducerFunction, initialState);
  console.log(state);
  return (
    <div>
      <h2>Reducer form</h2>
      <form action="">
        <input
          type="text"
          name="name"
          id=""
          className="border-2 m-3"
          placeholder="Your Name"
          onChange={(e) =>
            dispatch({ type: e.target.name, payload: e.target.value })
          }
        />
        <input
          type="text"
          name="email"
          id=""
          className="border-2 m-3"
          placeholder="Your Email"
          onChange={(e) => dispatch({type: e.target.name , payload : e.target.value})}
        />
        <button className="btn btn-neutral btn-sm">Submit</button>
      </form>
    </div>
  );
};

export default ReducerForm;
```

10

useContext ব্যবহার করার এক্সাম্পলঃ

UseContextExample.tsx ফাইলঃ

```
// --- type declaration
type TChildren = {
    children : ReactNode
}
// --- type declaration
type TThemeValue = {
    dark : boolean ;
    setDark : Dispatch<SetStateAction<boolean>>
}

export const themeProvider = createContext<TThemeValue | undefined>(undefined) ;


const UseContext3 = ({children} : TChildren) => {
    const [dark, setDark] = useState(false) ;
    const values : TThemeValue = {
        dark,
        setDark
    }

    return (
        <themeProvider.Provider value={values} >{children}</themeProvider.Provider>
    )
}

export default UseContext3 ;
```

main.tsx ফাইলঃ

```
ReactDOM.createRoot(document.getElementById("root")!).render(
  <UseContext3>
    <React.StrictMode>
      <div className="text-center w-screen ">
        {/* <App />  */}
        <ApplyContext3 />
      </div>
    </React.StrictMode>
  </UseContext3>
);
```

The file where we want to get/apply the value got from useContext :

```
import { themeProvider } from "./useContextExample3";


const ApplyContext3 = () => {
    const {dark, setDark} = useContext(themeProvider)
    return (
        <div>
            <h2 className={`${dark ? 'bg-black text-white' : 'bg-white text-black'}`}>Hello
world</h2>
            <button onClick={()=>setDark(!dark)} className="btn btn-neutral">Click Me</button>
        </div>
    )
}

export default ApplyContext3 ;
```

11

নীচে useRef এর একটি ব্যাসিক এক্সাম্পল দেখানো হলো, যেখানে useRef ব্যবহার করার কারণে একটি html পেইজ লোড হওয়ার সাথে সাথে সেই পেইজে থাকা ইনপুট ফিল্ডটি অটো ফোকাসে চলে আসবে।

```
const UseRef = () => {
  const myRef = useRef();

  useEffect(()=>{
    myRef.current.focus() ;
  },[])

  return (
    <div>
      <h2>Hello World</h2>
      <form action="">
        <input type="text" className="border border-red-500" ref={myRef} />
      </form>
    </div>
  );
};
```

12

আমরা চাইলেই একটি কম্পোনেন্ট আরেকটি কম্পোনেন্টে props আকারে যেকোন ভ্যালু বা স্টেট পাঠাতে পারি। কিন্তু আমরা props আকারে কখনো useRef কে পাঠাতে পারবোনা।

useRef কে props এর মত করে পাঠাতে হলে forward ref ব্যবহার করতে হবে।

যে কম্পোনেন্ট থেকে আমরা useRef এর ভ্যালু পাঠাবোঃ

```
import ForwardRefExample from "./Components/forwardRef";

const UseRef = () => {
  const myRef = useRef();

  useEffect(() => {
    myRef?.current?.focus();
  }, []);

  return (
    <div>
      <h2>Hello World</h2>
      <form action="">
        <ForwardRefExample ref={myRef} classProps={"border border-green-500"} />
      </form>
    </div>
  );
};

export default UseRef;
```

যে কম্পোনেন্ট-এ আমরা useRef এর ভ্যালু props এর মত করে ব্যবহার করবোঃ

```
const ForwardRefExample = forwardRef(({ classProps }, myRef) => {
  return (
    <div>
      <h2>Forward Ref</h2>
      <input type="text" className={classProps} ref={myRef} />
    </div>
  );
});

export default ForwardRefExample;
```

13

Higher Order ফাংশানঃ

যে ফাংশান প্যারামিটার হিসেবে আরেকটি ফাংশান নেয় , অথবা যে ফাংশান প্যারামিটার হিসেবে আরেকটি ফাংশান রিটার্ন করে , অথবা যে ফাংশান উপরের উভয় কাজটিই করে , সেটিই হচ্ছে Higher Order ফাংশান।

যেমন নিচের ফাংশানটি একটি সাধারণ ফাংশানঃ

```
function addFunc(item) {
  return item + 1;
}

console.log(addFunc(5));
```

এখন এই ফাংশানটিকেই আমরা একটা হায়ার অর্ডার ফাংশানের মধ্যে ব্যবহার করতে পারি।

```
function addFunc(item) {
  return item + 1;
}

console.log('Simple Function => ', addFunc(5));

function higherOrder(fn){
    return (prop)=> {
        return fn(prop) * 2 ;
    }
}

const applyHigherOrderFunction = higherOrder(addFunc);
console.log("Higher order function => ", applyHigherOrderFunction(5));

// output :
// Simple Function =>  6
// Higher order function =>  12
```

14

Higher order function থেকেই Higher order component এর আইডিয়া এসেছে।

যে কম্পোনেন্ট প্যারামিটার হিসেবে একটি কম্পোনেন্ট নেয় এবং রিটার্ন হিসেবে আরেকটি কম্পোনেন্ট রিটার্ন করে সেটিই হচ্ছে হায়ার অর্ডার কম্পোনেন্ট।

Main.tsx ফাইলঃ

```
<React.StrictMode>
      <div className="text-center w-screen ">
        <HigherOrderComponent3 />
      </div>
</React.StrictMode>
```

HighOrderComponent3.tsx ফাইলঃ

```
import { User3 } from "./User3";
import { AvatarWithBorder3 } from "./avatarBorder";

const imgUrl =
  "https://w7.pngwing.com/pngs/910/606/png-transparent-head-the-dummy-avatar-man-tie-jacket-user.png";

  const UserWithBorder = AvatarWithBorder3(User3) ;
```

```
export const HigherOrderComponent3 = () => {
  return (
    <div className="w-3/5 flex justify-center gap-5 mx-auto">
      <User3 img={imgUrl} />
      <User3 img={imgUrl} />
      <UserWithBorder img={imgUrl} />
      <User3 img={imgUrl} />
      <User3 img={imgUrl} />
    </div>
  );
};
```

AvatarWithBorder3.tsx ফাইলঃ

```
export const AvatarWithBorder3 = (ExistingAvatar) => {
    return (props) => {
        return (
            <div className="border-8 border-red-700 rounded-full">
                <ExistingAvatar {...props} />
            </div>
        )
    }
}
```

User3.tsx ফাইলঃ

```
export const User3 = ({ img } : {img : string}) => {
  return (
    <div>
      <img src={img} className="w-20 h-20 rounded-full" alt="" />
    </div>
  );
};
```

15

Tailwind এ বেইস html ট্যাগগুলোর স্টাইল রিসেট করা থাকে। যেমন h1, h2 , h3 ট্যাগের সকল স্টাইল একইরকম থাকে। এগুলো চাইলে আমরা কাস্টমভাবে পরিবর্তন করতে পারি। এজন্য আমাদেরকে index.css ফাইলে @layer ব্যবহার করে প্রয়োজনীয় স্টাইল লিখতে হবে। যেমনঃ

Index.css ফাইলঃ

```
@layer base{
    h1{
        @apply text-5xl text-green-300
    }
    h2{
        @apply text-2xl font-bold
    }
}
```

এছাড়া @layer দিয়ে আমরা আমাদের ইচ্ছানুযায়ী প্রয়োজনীয় কম্পোনেন্ট তৈরি করে নিতে পারি। যেমন আমরা চাচ্ছি আমাদের মনমত কিছু রিউজেবল বাটন বানিয়ে নিতে, যেটা আমরা আমাদের প্রজেক্টের যেকোন জায়গায় ব্যবহার করতে পারব।

Index.css ফাইলঃ

```
@layer components{
    .btn {
        @apply border rounded
    }
    .btn-primary {
        @apply border rounded bg-blue-800 text-white font-bold px-3 py-3
    }
    .btn-purple{
        @apply border rounded bg-purple-700 text-white font-bold px-3 py-3
    }
    .btn-danger{
        @apply border rounded bg-red-700 text-white font-bold px-3 py-3
    }
}
```

16

রেগুলার css এ যখন আমরা কোন স্টাইল লিখি , একই স্টাইল কয়েকরকম ভাবে লেখা হলে, তখন সবশেষে যে স্টাইলটা লেখা হয় সেটা এপ্লাই হয়, আগেরগুলা ওভাররাইট হয়ে যায়। কিন্তু tailwind এ এরকম হয়না। টেইলউইন্ডে তাদের নিজস্ব order এ যেমন ভাবে ক্লাসগুলো সাজানো থাকে তেমন ভাবে হয়ে থাকে। অর্থাৎ রেগুলার css এর মত সবশেষে থাকা স্টাইলটি এপ্লাই হয়না। উদাহরন হিসেবে নিচের বাটনটি দেখা যাক,

```
<Button className=' border-yellow-600  border-green-700 ' />
```

স্বভাবতই এখানে আমরা ধরে নিবো yellow কালার শো না করে green কালার শো করবে। কিন্তু tailwind এর ক্ষেত্রে এখানে yellow-ই শো করবে, green করবেনা। কারণ tailwind স্ট্রাকচারে সিরিয়াল অনুযায়ী green এর পরে yellow রয়েছে।

এইধরনের সমস্যা সমাধানের জন্য একটি প্যাকেজ ব্যবহার করা হয় যে প্যাকেজটির নাম হচ্ছে , tailwind merge বা twMerge.

twMerge() ফাংশানের ব্যবহারঃ

```
<Button className={twMerge('border-yellow-600  border-green-700')} />
```

বা এইভাবেঃ

```
 <Button className={twMerge(
  'border-yellow-600  border-green-700',
  'bg-red-500',
  className
  )} />
```

Clsx এর ব্যবহারঃ

প্রায়ই আমাদেরকে conditional css ব্যবহার করতে হয়। যেমনঃ

```
<button
className={twMerge(
  clsx(
    "bg-yellow",
    className,
    variant == "outline" && "bg-white text-red-600",
    variant == "solid" &&
      "bg-black text-yellow-500 border-none font-bold rounded-lg"
  )
)}
>
Click Me{" "}
</button>
```

এটা আরো সুন্দর করে সাজিয়ে লেখার জন্য clsx প্যাকেজটি ব্যবহার করা হয়। মানে কন্ডিশানগুলা আলাদা আলাদা ভাবে না লিখে একটা ব্রাকেটের মধ্যে সবগুলা লেখা হয়ঃ

```
    <button
      className={twMerge(
        clsx(
          "bg-yellow",
          className,
          {
            "bg-white text-red-600 border-slate-700 ": variant == "outline",
            "bg-black text-yellow-500 border-none font-bold rounded-lg":
              variant == "solid",
          }
        )
      )}
    >
      Click Me
    </button>
```

Cn ফাংশানের ব্যবহারঃ

আর এই twMerge() এবং clsx() ফাংশানকে একটি ফাংশানের ভেতর নিয়ে রিউজেবল বানানো হয় cn ফাংশান ব্যবহার করে।

Cn.tsx ফাইলঃ

```
import clsx from "clsx";
import { twMerge } from "tailwind-merge";


export default function Cn(...inputs){
    return twMerge(clsx(inputs))
}
```

যেখানে ব্যবহার করা হয়েছেঃ

```
import clsx from "clsx";
import { twMerge } from "tailwind-merge";
import Cn from "../utlis/cn";

export const Button = ({ className, variant }) => {
  return (
    <button
      className={Cn("bg-yellow", className, {
        "bg-white text-red-600 border-slate-700 ": variant == "outline",
        "bg-black text-yellow-500 border-none font-bold rounded-lg":
          variant == "outline",
      })}
    >
      Click Me
    </button>
  );
};
```

6

Core React দিয়ে একটি স্ক্রিণে একটি hello world প্রিন্ট

6

Core React দিয়ে একটি স্ক্রিণে একটি hello world প্রিন্ট

6

Core React দিয়ে একটি স্ক্রিণে একটি hello world প্রিন্ট

6

Core React দিয়ে একটি স্ক্রিণে একটি hello world প্রিন্ট

6

Core React দিয়ে একটি স্ক্রিণে একটি hello world প্রিন্ট

6

Core React দিয়ে একটি স্ক্রিণে একটি hello world প্রিন্ট