مفاهیم ری اکت

سلام من کسری ام امیدوارم این توضیحات بهت کمک کنه ...! 🤓

کامپوننت ها - Components

کامپونت‌ها در ری اکت بلوک‌های ساختاری هستند که به صورت مستقل و قابل استفاده مجدد طراحی می‌شوند

هر کامپونت می‌تواند حاوی کد اچ تی ام ال، وضعیت ، و توابع مختلفی باشد

یک کامپونت ساده برای نمایش یک خوش آمدگویی

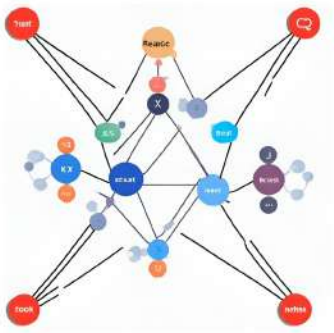
```
function WelcomeMessage() {

return (

<div>

<h1>Welcome to My React App!</h1>

<p>Enjoy exploring the components.</p>

</div>

)};
```

JSX (JavaScript XML)

JSX یک ترکیب از JAVASCRIPT , XML است که به شما این امکان را می‌دهد تا عناصر UI

را به صورت مستقیم در کد JavaScript بنویسید React در زمان اجرا JSX را به کد JavaScript

معمولی تبدیل می‌کند



یک JSX ساده برای نمایش یک دکمه :

```
function Button() {
```

```
return (
<button onClick={() => alert('Button Clicked!')}>
Click Me
</button>
```

---

state - وضعیت

وضعیت یکی از اصلی‌ترین مفاهیم React است و به کامپوننت‌ها امکان می‌دهد

تا اطلاعات دینامیک را در زمان اجرا مدیریت کنند

تغییر در وضعیت کامپونت باعث می‌شود که کامپونت مجدداً رندر شود

یک کامپوننت با وضعیت برای نمایش تعداد کلیک‌ها

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

function ClickCounter() {
const [count, setCount] = useState(0);
return (
<div> <p>Click Count: {count}</p>
<button onClick={() => setCount(count + 1)}>
Click Me
</button> </div>
)}}
```

Props - خصوصیت

Props داده‌هایی هستند که از یک کامپونت به کامپونت دیگر منتقل می‌شوند

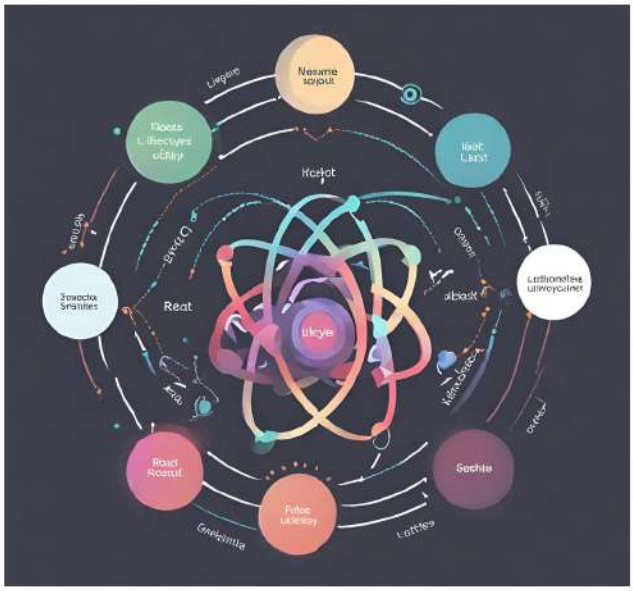
props به کامپونت اطلاعاتی ارائه می‌دهند که ممکن است در آن کامپونت مورد استفاده قرار گیرد

یک کامپونت با props برای نمایش یک نام

```
import React from 'react';
```

```
function Greeting(props) {

return <p>Hello, {props.name}!</p>

}

...

<Greeting name="John" />;
```

چرخه زندگی کامپوننت - Component Lifecycle



چرخه زندگی کامپوننت از تولد تا مرگ آن را مدیریت می‌کند و مراحل مختلفی از زندگی کامپوننت را نشان می‌دهد.

مراحل اصلی شامل componentDidMount، componentDidUpdate، componentWillUnmount هستند.

mounting : این متد در ابتدای ایجاد یک نمونه از کامپوننت صدا زده می‌شود

render : این متد مسئول ایجاد و برگرداندن JSX مربوط به کامپوننت است

componentDidMount :این متد پس از نصب (رندر اولیه) کامپوننت فراخوانی می‌شود

و امکان اجرای عملیات پس از رندر و نصب را فراهم می‌کند

componentDidUpdate : این متد پس از انجام عملیات به‌روزرسانی فراخوانی می‌شود

و امکان اجرای عملیات پس از به‌روزرسانی را فراهم می‌کند

componentWillUnmount : این متد پیش از حذف کامپوننت از DOM

فراخوانی می‌شود و امکان اجرای عملیات پیش از حذف را فراهم می‌کند

Context API

به شما امکان ایجاد یک محیط اشتراکی بین چندین کامپوننت را می‌دهد

مثال 😀👇

```
import React, { createContext, useContext } from 'react';


ایجاد یک Context

const ThemeContext = createContext('light');


function ThemedComponent() {

خواندن مقدار از  Context

const theme = useContext(ThemeContext);


return <p>Current Theme: {theme}</p>;

}

function App() {

return (

ارسال مقدار به  Context

<ThemeContext.Provider value="dark">

<ThemedComponent />

</ThemeContext.Provider>

)}
```

Hooks

هوک ها شامل : useState - useRef - useEffect - useContext -useRedcuer - useCallback - useMemo

useState

برای ایجاد و مدیریت وضعیت (state) در کامپوننت‌های فانکشنال استفاده می‌شود

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


function Counter() {

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

return (

<div> <p>Count: {count}</p>

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

)}
```

useEffect

برای انجام افعال (side effects) مانند ارتباط با سرور، عملیات دستگاه و... بعد از رندر کامپوننت استفاده می‌شود

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


function DataFetcher() {

const [data, setData] = useState(null);
```

```
useEffect(() => {
درخواست به سرور
fetch('https://api.example.com/data')
.then(response => response.json())
.then(data => setData(data));
}, []) آرایه خالی برای اجرای این افعال تنها یکبار
return (
<div>
{data ? <p>Data: {data}</p> : <p>Loading...</p>}
</div>
)}
```

useReducer

برای مدیریت وضعیت‌های پیچیده‌تر و اجرای افعال بر اساس اقدامات (actions)

در کامپوننت‌های تابعی استفاده می‌شود

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

const counterReducer = (state, action) => {
switch (action.type) {
case 'INCREMENT':
return { count: state.count + 1 };
case 'DECREMENT':
return { count: state.count - 1 };
default:
return state;
}};
```

```
function Counter() {

استفاده از useReducer

const [state, dispatch] = useReducer(counterReducer, { count: 0 });

return (

<div>

<p>Count: {state.count}</p>

<button onClick={() => dispatch({ type: 'INCREMENT' })}>Increment</button>

<button onClick={() => dispatch({ type: 'DECREMENT' })}>Decrement</button>

</div>

)}
```

useCallback

برای بهینه‌سازی عملکرد تابع‌ها و جلوگیری از رندر مجدد غیرضروری در هر بار رندر کامپوننت استفاده می‌شود.

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

function ClickCounter() {

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


// استفاده از useCallback برای بهینه‌سازی تابع

const handleClick = useCallback(() => {

setCount(count + 1);

}, [count]);

// اگر count تغییر کند، تابع رندر نمی‌شود

return (

<div>

<p>Click Count: {count}</p>
```

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

</div>

)}
```

useMemo

برای بهینه‌سازی محاسبات محاسبه‌گرها درون یک کامپوننت تابعی استفاده می‌شود

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

function ExpensiveCalculation({ data }) {

// استفاده از useMemo برای بهینه‌سازی محاسبات

const result = useMemo(() => {

// عملیات محاسبه گران‌قیمت

return performExpensiveCalculation(data);

}, [data]); //اگر data تغییر کند، محاسبه مجدد انجام می‌شود

return <p>Result: {result}</p>;

}
```

useRef

برای دسترسی به DOM یا نگهداری مقادیری که تغییر نکنند در طول زمان استفاده می‌شود

```
import React, { useRef, useEffect } from 'react';

function AutoFocusInput() {

const inputRef = useRef();

useEffect(() => {

// فوکوس به صورت خودکار بعد از رندر
```

```
inputRef.current.focus();

}, []);

return <input ref={inputRef} />}
```

امیدوارم بهت کمک کرده باشه و قابل فهم بوده باشه

راستی آدرس لینکدین من هم اینه : Kasra Tavakoli

هرچند وقت یکبار بهش سر بزن آپدیتش میکنم

🥰