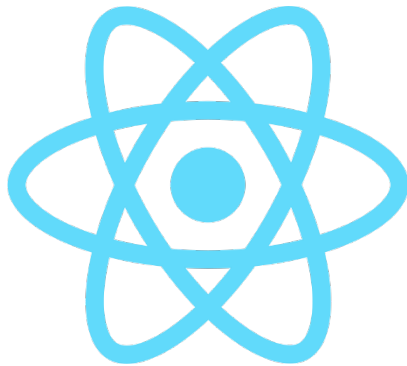


React



Render Condizionale

Ora, parliamo del rendering condizionale. Questo permette di visualizzare diversi componenti a seconda delle condizioni specificate. Per esempio, potremmo decidere di visualizzare un componente o un altro a seconda dello stato di un'applicazione. Per esempio creiamo un componente RenderCondizionale e ritorniamo cosa visualizzare in relazione allo stato della variabile useState

```
import React, { useState } from "react";  
  
const RenderCondizionale = () => {  
  const [loading, setLoading] = useState(false);  
  
  if (loading) {  
    return <h2>Loading...</h2>;  
  }  
  
  return (  
    <div>RenderCondizionale</div>  
  )  
}  
  
export default RenderCondizionale
```

Render Condizionale

Quanto appena visto può essere sicuramente utile quando dobbiamo fetchare dei dati così da mostrare loading il tempo necessario al caricamento. Vediamo come fare. Riprendiamo quando visto prima per il fetching. Aggiungiamo anche uno `useState` `isError`. Ovviamente se `error` sarà `true` manderemo a schermo il messaggio di errore. I due `return` per `loading` ed `error` possiamo farli come due componenti.

```
const url = "https://jsonplaceholder.typicode.com/users";
const RenderCondizionale = () => {

  const [isLoading, setIsLoading] = useState(true);
  const [isError, setIsError] = useState(false);
  const [users, setUsers] = useState([]);

  const getData = async () => {
    setIsError(false);
    setIsLoading(true);
    try {
      const response = await axios.get(url);
      setUsers(response.data);
    } catch (error) {
      console.log(error);
      setIsError(true);
    }
    setIsLoading(false);
  };
}
```

```
if (isLoading) {
  return <Loading></Loading>;
}
if (isError) {
  return <ErrorRender></ErrorRender>;
}

return (
  <div>RenderCondizionale</div>
)

const Loading = () => {
  return <h2>Loading...</h2>;
}

const ErrorRender = () => {
  return <h2>Attenzione è avvenuto un errore</h2>;
}
```

Render Condizionale

Come abbiamo visto un eventuale errore verrà intercettato dal try catch. Ora andiamo ad aggiungere lo useEffect e facciamo stampare gli utenti

```
useEffect(()=>{
  setTimeout(()=>{ //aggiunto per vedere chiaramente il cambio di componente
    getData();
  },3000)
},[])
if (isLoading) {
  return <Loading></Loading>;
}
if (isError) {
  return <ErrorRender></ErrorRender>;
}
```

```
return (
  <div><ul className="users">
    {users.map((el) => {
      const { name, id, username, email } = el;
      return (
        <li key={id} className="shadow">
          <div>
            <h5>{id} - {name} - {username} - {email}</h5>
          </div>
        </li>
      );
    })}
  </ul></div>
)
```

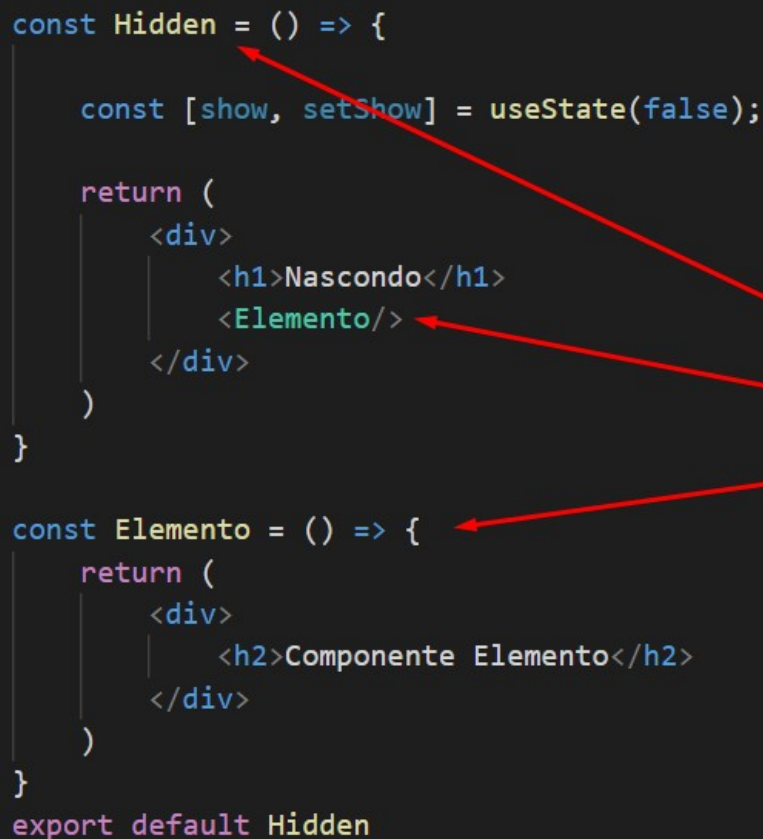
Nascondere Componente

Andiamo a vedere come possiamo nascondere e visualizzare un componente a seconda dello state di una variabile. Ad esempio se abbiamo una situazione del genere :

```
const Hidden = () => {  
  const [show, setShow] = useState(false);  
  
  return (  
    <div>  
      <h1>Nascondo</h1>  
    </div>  
  )  
}
```


Possiamo ora andare a creare un nuovo componente che utilizzeremo all'interno del componente appena creato in questo modo :

```
const Hidden = () => {  
  const [show, setShow] = useState(false);  
  
  return (  
    <div>  
      <h1>Nascondo</h1>  
      <Elemento/>  
    </div>  
  )  
}  
  
const Elemento = () => {  
  return (  
    <div>  
      <h2>Componente Elemento</h2>  
    </div>  
  )  
}  
  
export default Hidden
```



Nel componente principale andiamo ora a creare un bottone che al verificarsi dell'evento onClick andrà a settare il valore di show a false se era vero, e vero se era falso, in questo modo :

```
const Hidden = () => {  
  
  const [show, setShow] = useState(false);  
  
  return (  
    <div>  
      <h1>Nascondo</h1>  
      <button onClick={() => setShow(!show)}> Visualizza / Nascondi </button>  
      <Elemento/>  
    </div>  
  )  
}
```



Operatore Ternario

Andiamo ora a vedere come sia possibile «cambiare» la visualizzazione di un componente in base al valore di una variabile.

Ci viene naturale pensare che per risolvere il problema, potremmo utilizzare un classico IF, ma ricordiamoci che all'interno del rendering di un componente stiamo utilizzando **JSX e questo linguaggio non ci consente di utilizzare costrutti condizionali.**

Quindi ci viene in aiuto **l'operatore ternario**, che è una scorciatoia di un costrutto if che a differenza di una if **ritorna sempre un valore**. Questa è la sintassi :

```
nomeDaVerificare === 'valore' ? 'sono vero' : 'sono falso'
```

Se invece la variabile è un booleano basta mettere il nome della variabile :

```
nomeDaVerificare ? 'sono vero' : 'sono falso'
```

Per riprendere il nostro esempio possiamo modificare il bottone che cambia il suo valore in base al valore della variabile show in questo modo :

```
return (  
  <div>  
    <h1>Nascondo</h1>  
    <button onClick={() => setShow(!show)}> {  
      show ? 'Nascondi' : 'Visualizza'  
    } </button>  
    <Elemento/>  
  </div>  
)
```



Short Circuit Evaluation

Una short circuit evaluation è un'espressione che torna un valore in base allo stato di una costante o di una variabile che vogliamo andare a valutare. Esistono 2 tipi di short circuit :

And &&

Or ||

||

Esegue il controllo dell'espressione, se è vuota prende il secondo valore, **altrimenti** il primo :

```
const esempio = parola || "paperino"
```

Se la variabile *parola* ha un valore, allora *esempio* assume il valore di *parola*, altrimenti assume il valore "paperino".

&&

Esegue il controllo dell'espressione, se è vera prende il secondo valore :

const esempio = parola && "paperino"

Se la variabile parola ha un valore, allora esempio assume il valore di "paperino".

Quindi ritornando al nostro esempio posso utilizzare lo Short Circuit && per visualizzare o meno il componente Elemento in base al valore di show :

```
const Hidden = () => {  
  const [show, setShow] = useState(false);  
  
  return (  
    <div>  
      <h1>Nascondi</h1>  
      <button onClick={() => setShow(!show)}> {  
        show ? 'Nascondi' : 'Visualizza'  
      } </button>  
  
      {show && <Elemento/>}  
    </div>  
  )  
}
```



Visto come gestire la visualizzazione condizionata degli elementi torniamo un attimo sullo `useEffect`. Mettiamo un contatore sul componente elemento che aumenta ogni secondo:

```
const Elemento = () => {  
  const [contatore, setContatore] = useState(0);  
  
  useEffect(() => {  
    const timer = setTimeout(() => {  
      setContatore(oldValue => oldValue + 1);  
    }, 1000);  
    return ()  
  }, [contatore]);  
  return (  
    <div>  
      <h2>{contatore}</h2>  
    </div>  
  );  
}
```

Se non mettiamo la funzione di pulizia del contatore sul render del componente avremo questo messaggio in console:

```
[HMR] Waiting for update signal from WDS... log.js:24
❌ ▶ Warning: Can't perform a React state update on an unmounted component. This is a no-op, but it indicates a memory leak in your application. To fix, cancel all subscriptions and asynchronous tasks in a useEffect cleanup function. index.js:1
    at Elemento (http://localhost:3000/static/js/main.chunk.js:420:91)
```

```
useEffect(() => {
  const timer = setTimeout(() => {
    setContatore((oldValue) => oldValue + 1);
  }, 1000);
  return () => clearTimeout(timer);
}, [contatore]);
```


Esercizio

Creare un componente con un elenco di News composte da id, titolo e descrizione. Salvare in un file esterno 5 news con testo a piacimento ed all'interno del componente elencarle. Aggiungere un bottone finale che cambia il colore del font e dello sfondo di tutta la pagina. Situazione iniziale sfondo bianco scritte nere. Se clicco sul bottone sfondo nero e scritte bianche. Ad ogni click invertire nuovamente i colori.

Aggiungere bottone Nascondi / Visualizza News

- 1 – Creare File Esterno con le news
- 2 – Creare Componente Card che prende i dati dei prodotti (no img)
- 3 – Elencare i prodotti
- 4 – Aggiungere pulsante che cambia il colore delle card
- 5 – Aggiungere pulsante che cambia il colore dello sfondo della pagina