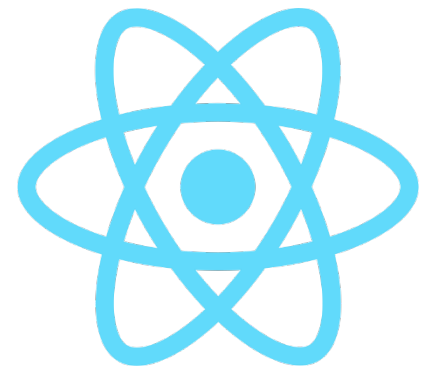


React



Form

Vediamo in questa lezione la gestione dei form con React partendo dal form di esempio messo a disposizione di bootstrap tenendo presente due regole :

- Sostituire class con **className**
- Sostituire il for della label con **htmlFor**

```
const Form = () => {  
  return (  
    <div className="container">  
      <form className="row g-3">  
        <div className="col-md-6">  
          <label htmlFor="inputNome" className="form-label">Nome</label>  
          <input type="text" className="form-control" id="inputNome" />  
        </div>  
        <div className="col-md-6">  
          <label htmlFor="inputCognome" className="form-label">Cognome</label>  
          <input type="text" className="form-control" id="inputCognome" />  
        </div>  
        <div className="col-12">  
          <button type="submit" className="btn btn-primary">Invia</button>  
        </div>  
      </form>  
    </div>  
  )  
}
```

```

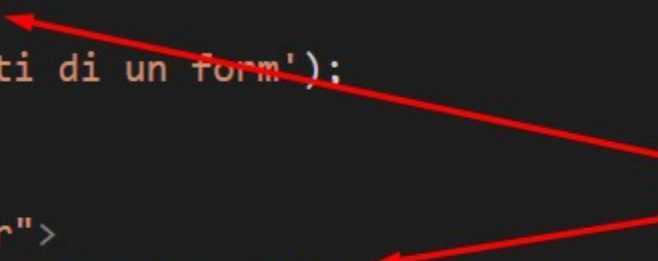
const Form = () => {
  return (
    <div className="container">
      <form className="row g-3">
        <div className="col-md-6">
          <label htmlFor="inputNome" className="form-label">Nome</label>
          <input type="text" className="form-control" id="inputNome" />
        </div>
        <div className="col-md-6">
          <label htmlFor="inputCognome" className="form-
label">Cognome</label>
          <input type="text" className="form-control" id="inputCognome" />
        </div>
        <div className="col-12">
          <button type="submit" className="btn btn-primary">Invia</button>
        </div>
      </form>
    </div>
  )
}

```

onSubmit

L'evento che «intercetta» il submit del form è **onSubmit** che serve per «gestire» l'invio del form e va applicato al tag `<form>` in questo modo :

```
const gestioneDati = () => {  
  console.log('gestione dati di un form');  
}  
return (  
  <div className="container">  
    <form className="row g-3" onSubmit={gestioneDati}>  
      <div className="col-md-6">
```



The diagram consists of two red arrows. The first arrow originates from the `gestioneDati` function definition and points to the `onSubmit={gestioneDati}` prop in the `<form>` tag. The second arrow originates from the `onSubmit={gestioneDati}` prop and points to the `onSubmit` attribute in the `<form>` tag, illustrating how the function is passed as a value to the event handler attribute.

event.preventDefault

Se proviamo ora ad inserire un nome ed un cognome e clicchiamo su invia, cosa succede ? Succede che React intercetta l'evento onSubmit, quindi esegue la funzione corrispondente ed alla fine dell'esecuzione della arrow function procede con **l'invio del form** che si traduce in un caricamento della pagina.

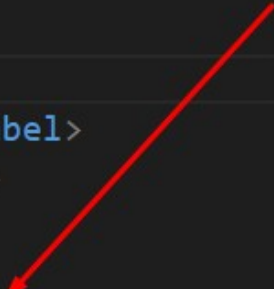
Per «prevenire» o meglio, per impedire che l'evento onSubmit completa il suo lavoro, possiamo intervenire passando alla funzione onSubmit l'evento e richiamare al suo interno la funzione preventDefault. Con questa funzione stiamo dicendo al compilatore di non effettuare il submit, ma che verrà gestito manualmente dal programmatore.

```
const Form = () => {  
  const gestioneDati = (e) => {  
    e.preventDefault();  
    console.log('gestione dati di un form');  
  }  
  return (  
    <div className="container">  
      <form className="row g-3" onSubmit={gestioneDati}>  
        <div className="col-md-6">
```

Se proviamo ora il nostro codice possiamo verificare che chiama comunque la funzione di submit ma non effettua il submit, rimanendo sulla pagina in attesa di un submit «manuale».

Questa gestione del submit posso farla attraverso l'evento onSubmit applicato al form (come abbiamo fatto in questo esempio) oppure attraverso l'evento onClick applicato al bottone, è la stessa cosa!

```
<form className="row g-3" >
  <div className="col-md-6">
    <label htmlFor="inputNome" className="form-label">Nome</label>
    <input type="text" className="form-control" id="inputNome" />
  </div>
  <div className="col-md-6">
    <label htmlFor="inputCognome" className="form-label">Cognome</label>
    <input type="text" className="form-control" id="inputCognome" />
  </div>
  <div className="col-12">
    <button type="submit" className="btn btn-primary" onClick={gestioneDati}>Invia
  </div>
</form>
```



Ora andiamo a dare un valore al nostro input e lo colleghiamo ad una variabile dichiarata utilizzando useState in questo modo :

```
const [nome, setName] = useState('');
const [cognome, setCognome] = useState('');

return (
  <div className="container">
    <form className="row g-3">
      <div className="col-md-6">
        <label htmlFor="inputNome" className="form-label">Nome</label>
        <input type="text" value={nome} className="form-control" id="inputNome" />
      </div>
      <div className="col-md-6">
        <label htmlFor="inputCognome" className="form-label">Cognome</label>
        <input type="text" value={cognome} className="form-control" id="inputCognome" />
      </div>
      <div className="col-12">
        <button type="submit" className="btn btn-primary" onClick={gestioneDati}>Invia</button>
      </div>
    </form>
  </div>
)
```

A diagram with three red arrows pointing from the state variables to the input values in the JSX. One arrow points from `useState('')` for `nome` to `value={nome}` in the first input. Another arrow points from `useState('')` for `cognome` to `value={cognome}` in the second input. A third arrow points from the `useState('')` for `nome` to the `htmlFor="inputNome"` attribute of the first label.

Oved422 - Ac

Così facendo possiamo notare che ora non posso scrivere nulla all'interno dell'input in quanto ho passato il «controllo» alla funzione `useState`.

Per ovviare a questo problema devo aggiungere nel nostro input l'evento **`onChange`** passando in ingresso l'evento e tramite una arrow function passare alla funzione `set` della `useState` l'evento `target.value` (`target` è l'evento del click sull'input) in questo modo :

```

<form className="row g-3" >
  <div className="col-md-6">
    <label htmlFor="inputNome" className="form-label">Nome</label>
    <input type="text" value={nome} onChange={(e) => setNome(e.target.value)} className="form-control" id="inputNome" />
  </div>
  <div className="col-md-6">
    <label htmlFor="inputCognome" className="form-label">Cognome</label>
    <input type="text" value={cognome} onChange={(e) => setCognome(e.target.value)} className="form-control" id="inputCognome" />
  </div>
  <div className="col-12">
    <button type="submit" className="btn btn-primary" onClick={gestioneDati}>Invia</button>
  </div>
</form>

```

In questo modo possiamo interagire con il nostro input ed andare ad inserire dei controlli sui dati inseriti. Possiamo anche creare una funzione handler da chiamare sull'onchange

```

const handleChange = (e) => {
  const { value } = e.target;
  setCellulare(value);
};

```

React

```

    onChange={handleChange}
    className="form-control col-9"
  ></input>

```


Submit

Per vedere come è possibile gestire un submit aggiungiamo all'interno del nostro componente il seguente codice. Vediamo che se clicchiamo sul pulsante effettivamente abbiamo il log nella console e il form viene svuotato

```
const gestioneDati = (e) => {  
  e.preventDefault();  
  
  if (nome && cognome) {  
    console.log(nome, cognome)  
    setNome("");  
    setCognome("");  
  } else {  
    alert("riempi il form");  
  }  
  
};
```

Ora aggiungiamo all'interno del nostro componente un array di persone ed ogni volta che clicchiamo su invio viene aggiunta una persona a questo array :

```
const [nome, setNome] = useState('');  
const [cognome, setCognome] = useState('');  
const [persone, setPersone] = useState([]);  
  
return (
```



A questo punto utilizzando il metodo gestioneDati eseguito al momento del click del form possiamo popolare il nostro array come già sappiamo fare utilizzando lo spread operator :

```
const gestioneDati=(e)=>{  
  e.preventDefault();  
  if (nome && cognome) {  
    setPersone([  
      ...persone,  
      {  
        nome,  
        cognome,  
      },  
    ]);  
    setNome("");  
    setCognome("");  
  } else {  
    alert("riempi il form");  
  }  
}
```

```
persone.map(el => {  
  return (  
    <h1>{el.nome} {el.cognome}</h1>  
  )  
})
```

Ora modifichiamo l'esempio usando un oggetto piuttosto che singole variabili. Aggiungiamo quindi l'oggetto persona, modifichiamo il value dei campi del form e creiamo una funzione handleChange per gestire la modifica del valore

```
const [persona, setPersona] = useState({  
  nome: "",  
  cognome: "",  
});
```

```
<div className="col-md-6">  
  <label htmlFor="inputNome" className="form-label">Nome</label>  
  <input type="text" value={persona.nome} onChange={handleChange} className="form-control">  
</div>  
<div className="col-md-6">  
  <label htmlFor="inputCognome" className="form-label">Nome</label>  
  <input type="text" value={persona.cognome} onChange={handleChange} className="form-control">  
</div>  
<div className="col-md-12">
```

```
const handleChange = (e) => {  
  const { name, value } = e.target;  
  setPersona({ ...persona, [name]: value });  
};
```

Andiamo quindi a popolare il nostro array persone in questo modo

```
const gestioneDati=(e)=>{  
  e.preventDefault();  
  
  if (persona.nome && persona.cognome) {  
    setPersone([  
      ...persone,  
      {  
        ...persona,  
      },  
    ]);  
    setPersona({  
      nome: "",  
      cognome: ""  
    });  
  } else {  
    alert("riempi il form");  
  }  
}
```


useRef

Con `useRef`, possiamo ottenere un riferimento diretto a un elemento DOM all'interno dei nostri componenti React. Ad esempio, immaginate di avere una serie di paragrafi (`<p>`) e alla fine un bottone che, quando premuto, scorrerà automaticamente fino all'ultimo paragrafo della pagina.

```
const RefExample = () => {  
  const ref = React.useRef(null);  
  console.log(ref);  
}
```

useRef

Associamo il ref al paragrafo e creiamo un pulsante con la funzione di scroll sull'onclick

```
<p>Questo è un paragrafo.</p>
<p>Questo è un paragrafo.</p>
<p>Questo è un paragrafo.</p>
<p>Questo è un paragrafo.</p>
<p ref={ref}>Questo è l'ultimo paragrafo.</p>
<div
  style={{
    height: "30vh",
  }}
></div>
```

```
const handleScroll = () => {
  if (!ref || !ref.current) {
    return;
  }

  ref.current.scrollIntoView({ behavior: "smooth", block: "center" });
};

return (
  <>
    <h1>Use Ref</h1>
    <div
      className="my-5 mx-auto"
      style={{
        height: "100vh",
      }}
    >
      <button className="btn btn-info" onClick={handleScroll}>
        Scroll
      </button>
    </div>
```

useRef

useRef può essere usato anche per riferirci a campi di input. Il precedente esempio sui form potrebbe essere come segue

```
1 import React, { useRef,useState } from 'react'
2
3 const FormUseRef = () => {
4   const nomeRef=useRef(null);
5   const cognomeRef=useRef(null);
6
7   const [persone,setPersone]=useState([]);
8
9   const handlerSubmit=(e)=> {
10     e.preventDefault();
11     const nome = nomeRef.current.value;
12     const cognome = cognomeRef.current.value;
13     if(nome && cognome){
14
15       setPersone([
16         ...persone,
17         { nome, cognome }
18       ])
19       nomeRef.current.value = '';
20       cognomeRef.current.value = '';
21
22     }else{
23       console.log("compilare nome e cognome")
24     }
25
26     console.log("Submit form");
27
28   }
29
30   return (
31     <div className="container">
32       <h1>UseREF</h1>
33       <form className="row g-3" onSubmit={handlerSubmit}>
34         <div className="col-md-6">
35           <label htmlFor="inputNome" className="form-label">Nome</label>
36           <input type="text" ref={nomeRef} name="nome" className="form-control">
37         </div>
38         <div className="col-md-6">
39           <label htmlFor="inputCognome" className="form-label">Cognome</label>
40           <input type="text" ref={cognomeRef} name="cognome" className="form-control">
41         </div>
42         <div className="col-md-12">
43           <button type="submit" className="btn btn-primary">Invia</button>
44         </div>
45       </form>
46     </div>
47     {
48       persone.map((el,index)=>{
49         const {nome,cognome}=el
50         return(<p key={index}>>{nome} {cognome}</p>)
51       })
52     }
53   )
54 }
```

Esercizio

Creare un nuovo progetto con un componente che visualizza un menu con le seguenti voci : Home – Chi Sono – Contatti.

Al click di ogni singola voce viene visualizzato un componente. Sulla home inserire un'immagine di presentazione a piacere. Su chi sono un breve testo che vi presenta e su contatti un form compilabile che all'invio visualizza un messaggio di conferma solo se sono stati compilati tutti i campi (i campi sono nome, cognome, email, cellulare e messaggio).