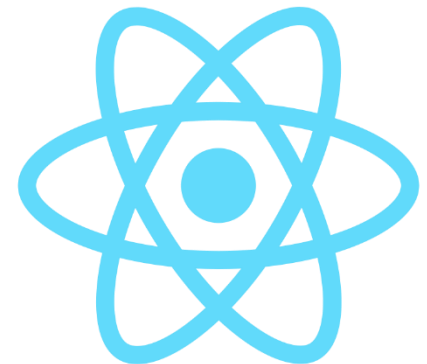


React

Riccardo Cattaneo



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 :

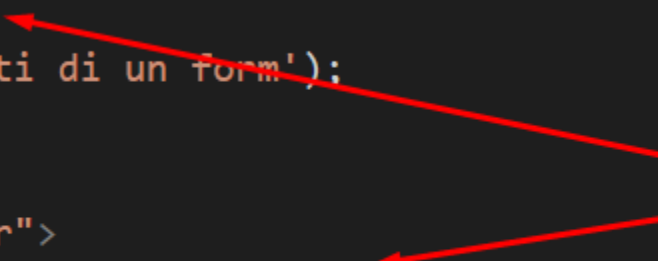
- Sostituire class con **className**
- Sostituire il for della label con **htmlFor**

```
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>  
  )  
}
```

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">
```

A diagram consisting of two red arrows. The first arrow originates from the function name 'gestioneDati' in the code snippet above and points to the 'onSubmit' prop in the JSX element below. The second arrow originates from the 'onSubmit={gestioneDati}' prop and points to the 'onSubmit' attribute in the JSX element below, effectively highlighting the connection between the function and the form's submission event.

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.

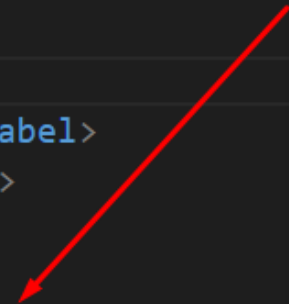
Per «prevenire» o meglio, per impedire che l'evento onSubmit completa il suo lavoro, possiamo intervenire passando alla funzione onSubmit l'evento e richiamare al suo interno la funzione preventDefault. Con questa funzione stiamo dicendo al compilatore di non effettuare il submit, ma che verrà gestito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

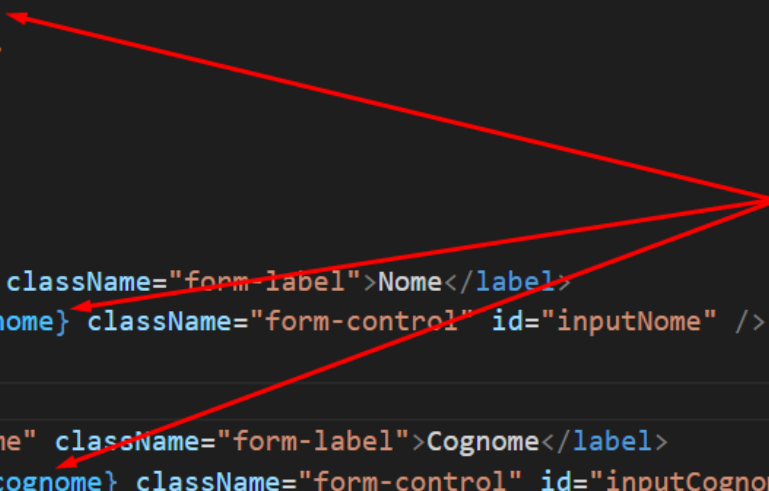
```
<form className="row g-3" >
  <div className="col-md-6">
    <label htmlFor="inputNome" className="form-label">Nome</label>
    <input type="text" className="form-control" id="inputNome" />
  </div>
  <div className="col-md-6">
    <label htmlFor="inputCognome" className="form-label">Cognome</label>
    <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. One arrow points from the 'nome' variable in the first useState declaration to the 'value={nome}' attribute of the first input field. A second arrow points from the 'cognome' variable in the second useState declaration to the 'value={cognome}' attribute of the second input field. A third arrow points from the 'useState' function in both declarations to the 'value' attributes of both input fields.

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) => setName(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... banalmente dare un alert se il campo viene lasciato vuoto... in questo modo :

Submit

Per vedere come è possibile gestire un submit 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();  
  console.log('gestione dati di un form');  
  setPersone([  
    ...persone,  
    {  
      nome,  
      cognome  
    }  
  ]);  
}
```

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

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).