Umberto Emanuele

React – Giorno 4

Novembre 2023

Form e Validazione Input lato utente

https://it.reactjs.org/docs/forms.html
https://jaredpalmer.com/formik/docs/overview

Form e Validazione Input lato utente

```
class FormNome extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('E\' stato inserito un nome: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Nome:
          <input type="text" value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Con un **componente controllato**, ogni mutazione dello **stato** deve aver associata una funzione **handler**. Tutto ciò rende il processo di **modifica** e la **validazione** dell'input dell'utente semplice e lineare.

https://it.reactjs.org/docs/forms.html

Form e Validazione Input lato utente

```
state = {
  foo: {
   a: 1,
   b: 2,
   c: 3
  }
}
```

```
this.setState({ foo: {
  ...this.state.foo,
  c: 'updated value'
}})
```

Fare attenzione se si deve modificare una proprietà di un oggetto contenuto all'interno dello **state**.
In questo caso si deve utilizzare lo **spred operators** per evitare la riscrittura completa dell'oggetto.

https://it.reactjs.org/docs/forms.html

Form e Validazione Input lato utente

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.handleSubmit = this.handleSubmit.bind(this);
    this.input = React.createRef();
  }

  handleSubmit(event) {
    alert('A name was submitted: ' + this.input.current.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" ref={this.input} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

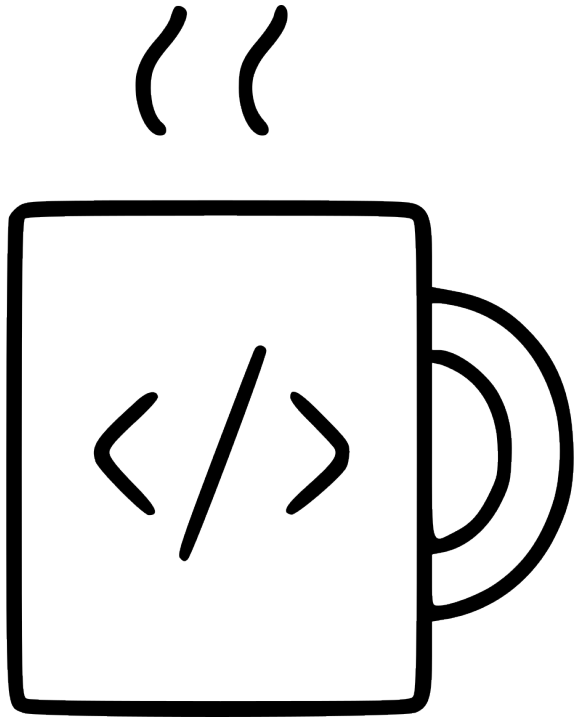
Un' alternativa ai Componenti Controllati, che ci evita di scrivere un **event handler** per ogni modo in cui i tuoi dati possono cambiare, sono i **componenti non controllati**, una tecnica alternativa per implementare **forms** ed i relativi **campi di input**.

https://it.reactjs.org/docs/forms.html

Form e Validazione Input lato utente

Una soluzione che include la **validazione dei dati**, il tener traccia dei campi visitati e la sottomissione del form è **Formik**. Comunque, si basa sugli stessi principi dei componenti controllati e della gestione dello stato.

https://jaredpalmer.com/formik/



PAUSA

Ci vediamo alle ore 11.25

Chiamata a API

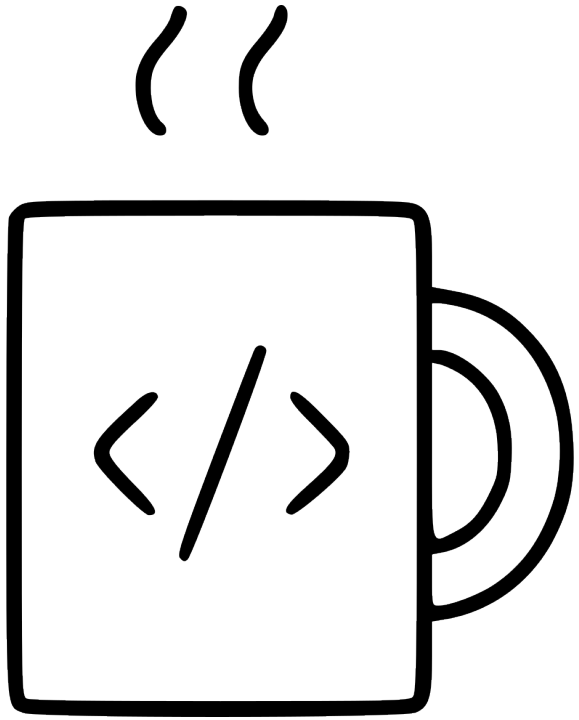
Chiamata a API

Come effettuare chiamate API con **fetch** e gestire il formato **JSON** nel **then** o l'errore nel **catch**

```
class App extends React.Component {
  render(){
  return (
    <div className="App">
      <h1>My Movie React App</h1>
    </div>
  );
  }

  componentDidMount() {
    console.log('Inizio Chiamata fetch');
    fetch('http://www.omdbapi.com/?i=tt3896198&apikey=3f90d51f')
      .then( result => result.json())
      .then( json =>  console.log(json));
  }
}

export default App;
```



PAUSA

Ci vediamo alle ore 14.00

Chiamata a API

Aggiungere degli **spinner** o indicatori di caricamento (**loaders**) è un ottima maniera per migliorare la **UX** (user experience) quando si ha a che fare con operazioni asincrone che richiedono del tempo per essere risolte, come fetch di dati remoti.

```
<div className="text-center">
  {/* render condizionale dello Spinner */}
  {this.state.loading && ( // short-circuit operator
    <Spinner animation="border" variant="success" />
  )}
</div>
```

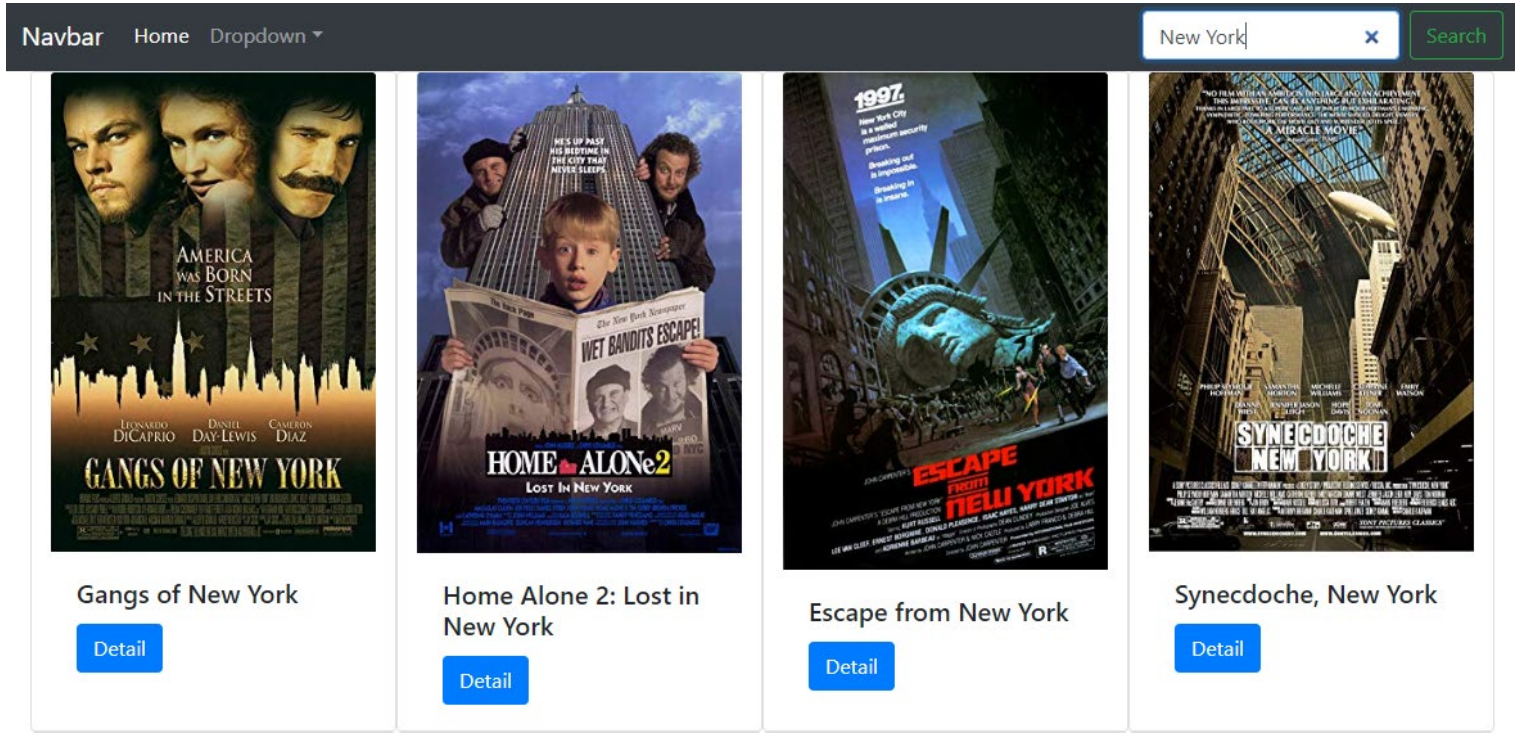
Chiamata a API

Con ReactJS è abbastanza facile informare i nostri utenti di un comportamento inaspettato avvenuto nell'applicazione.

```
{this.state.errMess &&
  <Alert variant="warning">
    Cannot load the data: {this.state.errMess}
  </Alert>
}
```

Chiamata a API

Effettuare chiamate **fetch** e gestire il JSON per visualizzare i risultati.



Progetto omdbapi

Chiamata a API

Using Axios with React

Quasi ogni progetto deve interfacciarsi con un'API REST ad un certo punto. Axios è un client HTTP leggero basato sul servizio $ http simile all'API Fetch .

```
# npm
$ npm install axios --save
```

https://alligator.io/react/axios-react/

```
import React from 'react';

import axios from 'axios';

export default class PersonList extends React.Component {
  state = {
    persons: []
  }

  componentDidMount() {
    axios.get(`https://jsonplaceholder.typicode.com/users`)
      .then(res => {
        const persons = res.data;
        this.setState({ persons });
      })
  }

  render() {
    return (
      <ul>
        { this.state.persons.map(person => <li>{person.name}</li>)}
      </ul>
    )
  }
}
```

Chiamata a API


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

export default class PersonList extends React.Component {
  state = {
    name: '',
  }

  handleChange = event => {
    this.setState({ name: event.target.value });
  }

  handleSubmit = event => {
    event.preventDefault();

    const user = {
      name: this.state.name
    };

    axios.post(`https://jsonplaceholder.typicode.com/users`, { user })
      .then(res => {
        console.log(res);
        console.log(res.data);
      })
  }
```

POST Requests

```
  render() {
    return (
      <div>
        <form onSubmit={this.handleSubmit}>
          <label>
            Person Name:
            <input type="text" name="name" onChange={this.handleChange} />
          </label>
          <button type="submit">Add</button>
        </form>
      </div>
    )
  }
}
```

Chiamata a API

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

export default class PersonList extends React.Component {
  state = {
    id: '',
  }

  handleChange = event => {
    this.setState({ id: event.target.value });
  }

  handleSubmit = event => {
    event.preventDefault();

    axios.delete(`https://jsonplaceholder.typicode.com/users/${this.state.id}`)
      .then(res => {
        console.log(res);
        console.log(res.data);
      })
  }
```

```
  render() {
    return (
      <div>
        <form onSubmit={this.handleSubmit}>
          <label>
            Person ID:
            <input type="text" name="id" onChange={this.handleChange} />
          </label>
          <button type="submit">Delete</button>
        </form>
      </div>
    )
  }
}
```

DELETE Requests

challenge
NETWORK

shaping the skills of tomorrow

challengenetwork.it