04-05 Context

L'idea è quella di avere una variabile che possa essere utilizzata da tutta l'applicazione.

- Impostare tema chiaro/scuro
- Infomrazioni di contesto (utente loggato o meno)
- Lingua in applicazione multi-lingua

Tree componenti principali:

- **Definizione del contesto**
  - Definisce un context object e lo salva nel riferimento ExContect
    - Crea un nuovo oggetto context, contenente due proprietà:
      - Provider
      - Consumer
  - I componenti possono iscrivere (consumare) il contesto

```
const ExContext = React.createContext(defaultValue)
```

- **Provider del contesto**
  - Fornisce il contesto a tutti i componenti
  - Contiene props: **Value**
    - Che diventa disponibile a tutti i componenti annidati

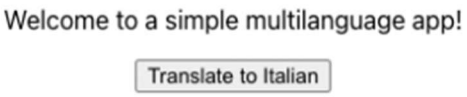
```
<ExContext.Provider value=...>
```

- **Consumer del contesto**
  - Tramite componente
  - Tramite hook

```
<ExContext.Consumer>

useContext(ExContext)
```

App.jsx

```
. . .
function App() {
  const [language, setLanguage] = useState('english');

  function toggleLanguage() {
    setLanguage((language) =>
      (language === 'english' ? 'italian' : 'english'));
  }


  return (
    <div className="App">
      <Welcome />
      <Button toggleLanguage={toggleLanguage} />
    </div>
  . . .
```



Language → stato che da valore al contesto

App.jsx

```
import LanguageContext
    from './languageContext';
```

languageContext.js

```
import React from 'react';

const LanguageContext = React.createContext();

export default LanguageContext;
```

Se la props value cambia, tutti i consumer vengono ri-renderizzati

Context Provider:

il componente che fornisce il contesto deve essere al di fuori dei componenti che vuole aggiornare:

- **Value prop**: disponibile per ogni consumer
- *Quando una props value del provider cambia, tutti i consumer vengono aggiornati*

App.jsx

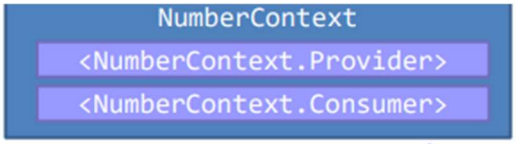
```jsx
import LanguageContext from './languageContext';
. . .

function App() {
  . . .
  return (
    <div className="App">
      <LanguageContext.Provider value={language}>
        <Welcome />
        <Button toggleLanguage={toggleLanguage} />
      </LanguageContext.Provider>
    </div>
  );
}
```

languageContext.js

```jsx
import React from 'react';

const LanguageContext = React.createContext();

export default LanguageContext;
```

Context Consumer:

Necessita di un callback, restituisce l'elemento react da ri-renderizzare



Hook mi permette di accedere al valore

```
function Display() {

  const value = useContext(NumberContext);

  return <div>The answer is {value}.</div>;
}
```

App.jsx

```
import LanguageContext from './languageContext';
. . .

function App() {
  . . .
  return (
    <div className="App">
      <LanguageContext.Provider value={language}>
        <Welcome />
        <Button toggleLanguage={toggleLanguage} />
      </LanguageContext.Provider>
    </div>
  );
}
. . .
```

Components.jsx

```
import { useContext } from 'react';
import LanguageContext from './languageContext';
import translations from './translations';

function Button(props) {
    const language = useContext(LanguageContext);

    return (
        <button onClick={props.toggleLanguage}>
          {translations[language]['button']}
        </button>
    );
}

function Welcome() {
    const language = useContext(LanguageContext);

    return (
        <p> {translations[language]['welcome']} </p>
    );
}
```

Definire contesti multipli:

```
function HeaderBar() {
  return (
    <CurrentUser.Consumer>
      {user =>
        <Notifications.Consumer>
          {notif =>
            <header>
              Welcome back, {user.name}!
              You have {notif.length}
              notifications.
            </header>
          }
        </Notifications.Consumer>
      }
    </CurrentUser.Consumer>
  );
}
```

Consumer Component

```
function HeaderBar() {
  const user = useContext(CurrentUser);
  const notif = useContext(Notifications);

  return (
    <header>
      Welcome back, {user.name}!
      You have {notif.length} notifications.
    </header>
  );
}
```

useContext Hook

quella di dx, migliore

Per cambiare il valore di contesto, si può fare con:

- Props drilling
- Metodo per aggiornarlo creato insieme al contesto

```
Example: { language: 'English', toggleLanguage : toggleLanguage }
```

Nota: Lo stato dove di solito salviamo i valore di contesto è parte del componente provider

***Nota:** lo stato in cui salviamo solitamente il contresto, è nel provider*

Caveats

- Non abusare del contesto
  - Componente diventa strettamente dipendente dal contesto
  - Perdo portabilità
- Non usare per pigrizia
  - In alcuni casi serve passare una props a 10 componenti ma il contesto non è scelta saggia
- Non usare il contesto per correggere errori di design