Modulo 2

Web Tier

React - Day 4

Presentato da
Vitale Esca
IBM Client Innovation Center - Italy

23/11/2023

IBM Client Innovation Center
Italy

Agenda Day 4

1. Setup ambiente di sviluppo
2. Introduzione a React
3. Pensare a componenti + Quiz
4. Hooks

IBM Client Innovation Center
Italy

Agenda Day 4

5 Comunicazione tra componenti

6 Pratica - Primo Componente

7 Single Page Application e Routing

8 Esercitazione

Prima di iniziare...

Installare *Node*, il nostro ambiente di runtime JavaScript
https://nodejs.org/

Installare *NPM*, il nostro gestore di pacchetti JavaScript
https://www.npmjs.com/

Creiamo il nostro progetto

Dopo esserci assicurati di aver installato una versione di Node >= 14.00 e una versione npm >= 5.6, possiamo creare il nostro primo progetto utilizzando:

- **npx**
- **Create React App**

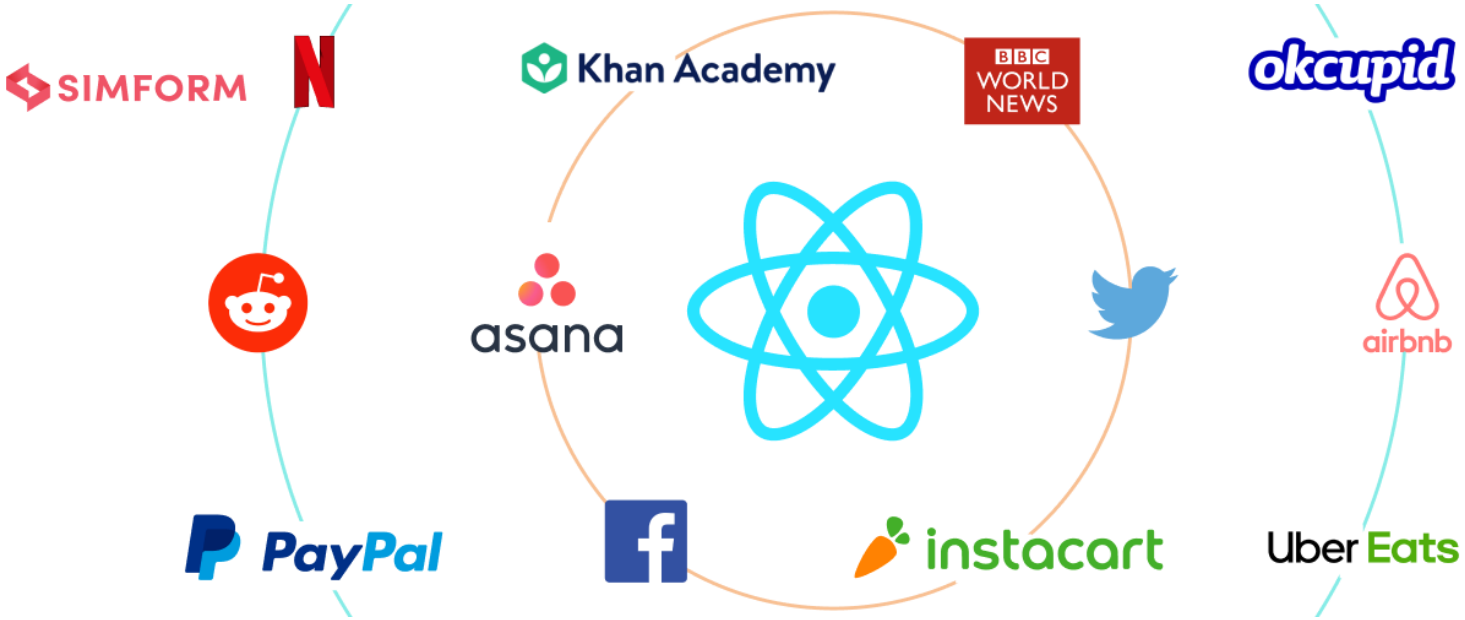
```
npx create-react-app mia-app
cd mia-app
npm start
```

Che cos’è React?

React è una libreria JavaScript open-source utilizzata per costruire User Interface accattivanti.

React ha tre punti di forza:

- Approccio dichiarativo;
- Efficienza;
- Flessibilità



Approccio dichiarativo - JSX

React si basa su un approccio dichiarativo simile all'HTML utilizzando una sintassi chiamata JSX.

JSX ci permette di inserire il nostro codice di markup all'interno del JavaScript:

```
const element = <h1>Hello, world!</h1>;
```

L'assegnazione di codice HTML in un elemento JavaScript è un esempio di espressione JSX che, una volta compilata, viene tradotta come una chiamata a una funzione JavaScript che produce un elemento JavaScript.

Approccio dichiarativo - JSX

Chi si occupa della compilazione del codice JSX in React è **Babel.**

Babel compila JSX in chiamate a *React.createElement() che, a sua volta, crea oggetti chiamati* “elementi React”.


```
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

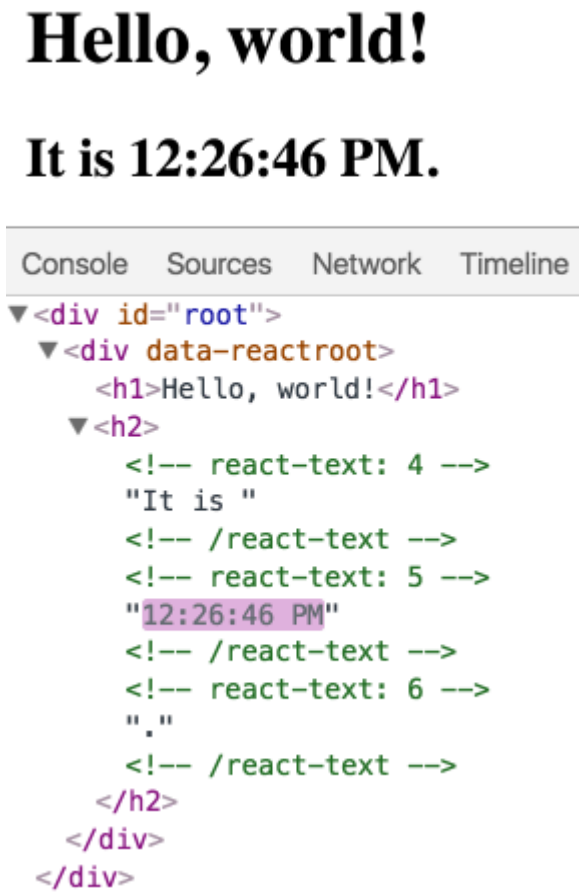
```
// Nota: questa struttura è semplificata
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, world!'
  }
};
```

IBM Client Innovation Center
Italy

Efficienza – Virtual DOM

L’efficienza di React è garantita dall’utilizzo del Virtual DOM.

Il Virtual DOM è un concetto di programmazione in cui una rappresentazione di un’interfaccia utente viene mantenuta in memoria e sincronizzata con il DOM reale, fornendo a quest ultimo un meccanismo che gli consente di calcolare operazioni DOM minime durante il nuovo rendering dell'interfaccia utente.



IBM Client Innovation Center
Italy

Flessibilità – Modularità - Scalabilità

- Altamente modulare;
- Se utilizzato in combinazione con altre tecnologie, permette di creare applicazioni web sempre più complesse;
- Cross-Platform.

Tutto ciò rende React una libreria estremamente flessibile e scalabile.

Che cos’è un componente?

Un componente è una piccola parte dell’interfaccia utente di un applicazione che ha il vantaggio di essere:

- Indipendente;
- Riutilizzabile;
- Isolata.

Proviamo con un esempio...

Immaginiamo di dover implementare una lista di prodotti ricercabili.

Come posso suddividerla in componenti?

Ci sono alcune tecniche da seguire per stabilire se una parte della UI è un componente, una di queste è il **principio di responsabilità singola.**

Proviamo con un esempio...

Abbiamo identificato cinque componenti:

1. **TabellaProdottiRicercabile**
2. **BarraRicerca**
3. **TabellaProdotti**
4. **RigaCategoriaProdotti**
5. **RigaProdotto**

Proviamo con un esempio...

Una volta identificati i componenti, bisogna ordinarli gerarchicamente.

- **TabellaProdottiRicercabile**
  - **BarraRicerca**
  - **TabellaProdotti**
    - **RigaCategoriaProdotti**
    - **RigaProdotto**

Che cos’è un componente in React?

In React un componente è una funzione pura JavaScript che renderizza un elemento React (JSX).

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

export default Welcome;
```

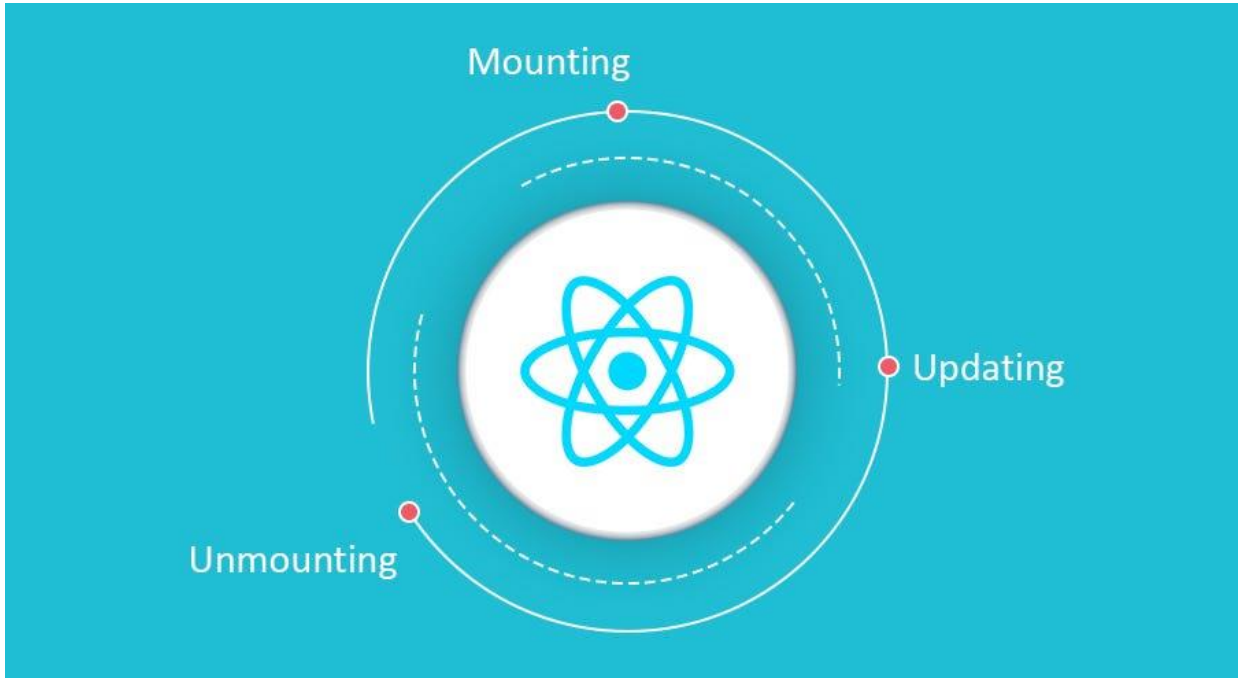
Questo snippet mostrato sopra è un esempio di Functional Component.

In realtà React offre anche un’altra tipologia di componenti (ormai in disuso) che sono i Class Component.

Ciclo di vita dei componenti

Ogni componente in React segue un ciclo di vita, in particolare distinguiamo tre fasi principali:

- Fase di Mounting;
- Fase di Updating;
- Fase di Unmounting.



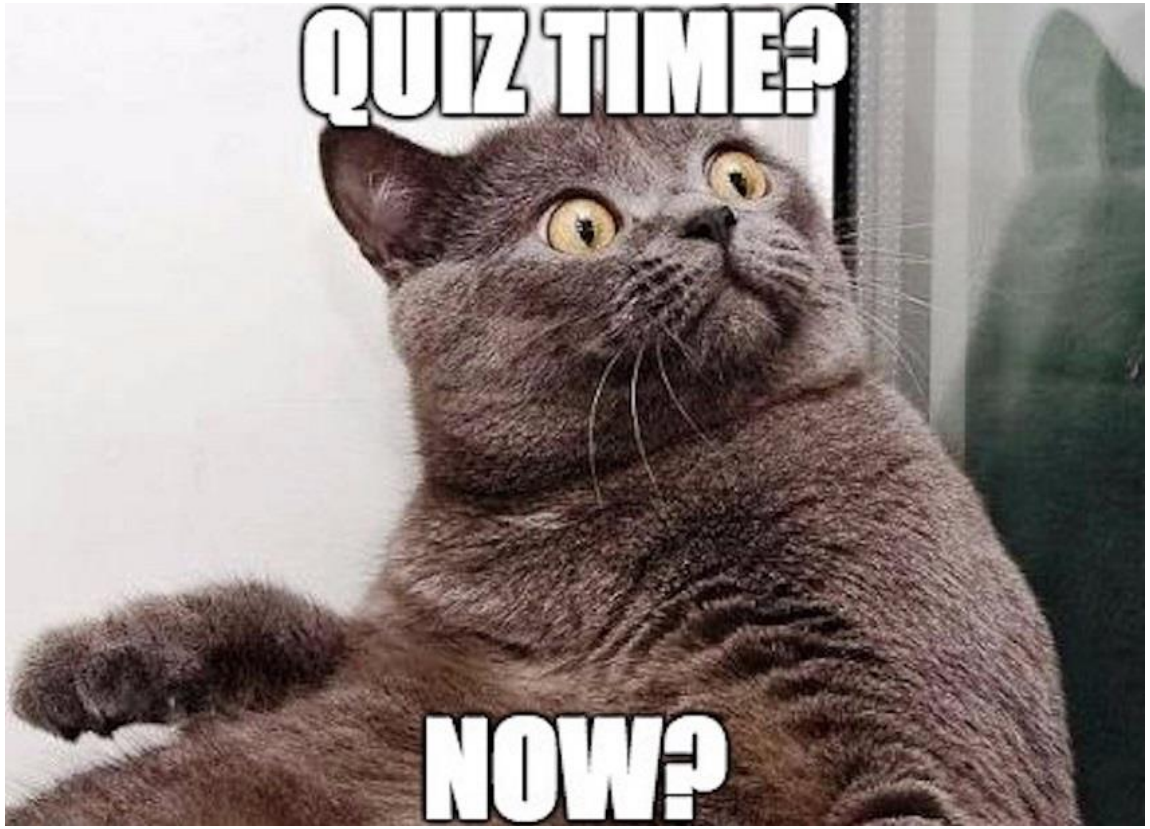
Props e State

Props e State sono due oggetti JavaScript che caratterizzano un componente React e che influenzano la renderizzazione di quest'ultimo all'interno del DOM.

La differenza tra props e state è che:

- le **props** vengono passate al componente dall'esterno;
- lo **state** è gestito internamente al componente.

Day 4 – Quiz



Hooks

Dalla versione *16.8* di React sono stati introdotti gli *Hooks*, che sono delle funzioni che ti consentono di agganciarti al ciclo di vita dei componenti e di utilizzare lo stato o altre funzionalità di React senza dover scrivere necessariamente una classe.

I principali Hooks offerti da React sono:

- useState;
- useEffect;

Hook useState

useState è un Hook utilizzato per definire uno stato locale di un componente.

Restituisce una coppia formata dal valore dello stato corrente e da una funziona che permette di aggiornarlo.

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Hook useEffect

useEffect è un Hook utilizzato per gestire l'esecuzione di effetti collaterali durante il ciclo di vita di un componente funzionale.

```
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Hook useEffect

useEffect accetta come parametri:

- una funzione di callback;
- un array di dipendenze;
- una funzione di pulizia,

```
// 1. importa useEffect
import { useEffect } from 'react';

function MyComponent() {
  // 2. chiamalo prima che il JSX venga restituito
  // 3. passagli due argomenti: una funzione e un array
  useEffect(() => {}, []);

  // return ...
}
```

Hook useEffect - Esempi

Di seguito alcuni esempi dell'utilizzo di useEffect:

Caso d'uso 1 – Una volta montato il componente, vogliamo fare una chiamata fetch ad una API per recuperare dei dati.

[Soluzione Sbagliata]

```
function MyComponent() {
  const [data, setData] = useState([])

  useEffect(() => {
    fetchData().then(myData => setData(myData))
    // Errore! useEffect viene eseguito dopo ogni rendering senza l'array delle dipendenze,
  });
}
```

Hook useEffect - Esempi

Di seguito alcuni esempi dell'utilizzo di useEffect:

Caso d'uso 1 – Una volta montato il componente, vogliamo fare una chiamata fetch ad una API per recuperare dei dati.

[Soluzione Corretta]

```
function MyComponent() {
  const [data, setData] = useState([])

  useEffect(() => {
    fetchData().then(myData => setData(myData))
    // Corretta! Viene eseguito una volta dopo il rendering con array vuoto
  }, []);

  return <ul>{data.map(item => <li key={item}>{item}</li>)}</ul>
}
```

Hook useEffect - Esempi

Di seguito alcuni esempi dell'utilizzo di useEffect:

Caso d'uso 2 – Il mio componente riceve in *props* una variabile name e ogni volta che cambia voglio modificare di conseguenza il titolo del document.

```
import { useEffect } from 'react';

function User({ name }) {
  useEffect(() => {
    document.title = name;
  }, [name]);

  return <h1>{name}</h1>;
}
```

Hook useEffect - Esempi

Di seguito alcuni esempi dell'utilizzo di useEffect:

Caso d'uso 3 – Utilizzare una *setInterval* per attivare un timer e modificare una porzione di stato.

[Soluzione Sbagliata]

```
function Timer() {
  const [time, setTime] = useState(0);

  useEffect(() => {
    setInterval(() => setTime(1), 1000);
    // conta fino a 1 ogni secondo
    // dobbiamo smettere di usare setInterval quando il componente viene smontato
  }, []);
}
```

Hook useEffect - Esempi

Di seguito alcuni esempi dell'utilizzo di useEffect:

Caso d'uso 3 – Utilizzare una *setInterval* per attivare un timer e modificare una porzione di stato.

[Soluzione Corretta]

```
function Timer() {
  const [time, setTime] = useState(0);

  useEffect(() => {
    let interval = setInterval(() => setTime(1), 1000);

    return () => {
      // setInterval azzerato quando il componente viene smontato
      clearInterval(interval);
    }
  }, []);
}
```

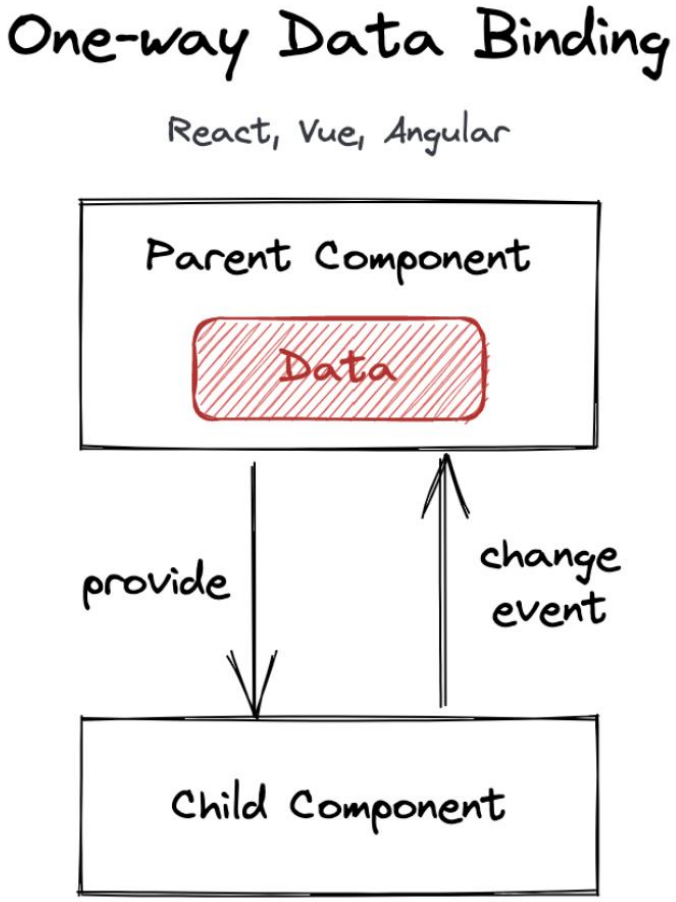
Comunicazione unidirezionale

In React la comunicazione è unidirezionale (one-way Data Binding).

Abbiamo visto come tramite l'oggetto *Props* i dati scorrono nella gerarchia di component dall'alto verso il basso.

Che cosa succede se un componente "figlio" vuole modificare lo stato del componente "padre" oppure vuole condividere delle informazioni con un altro componente ad esso adiacente?

Comunicazione unidirezionale

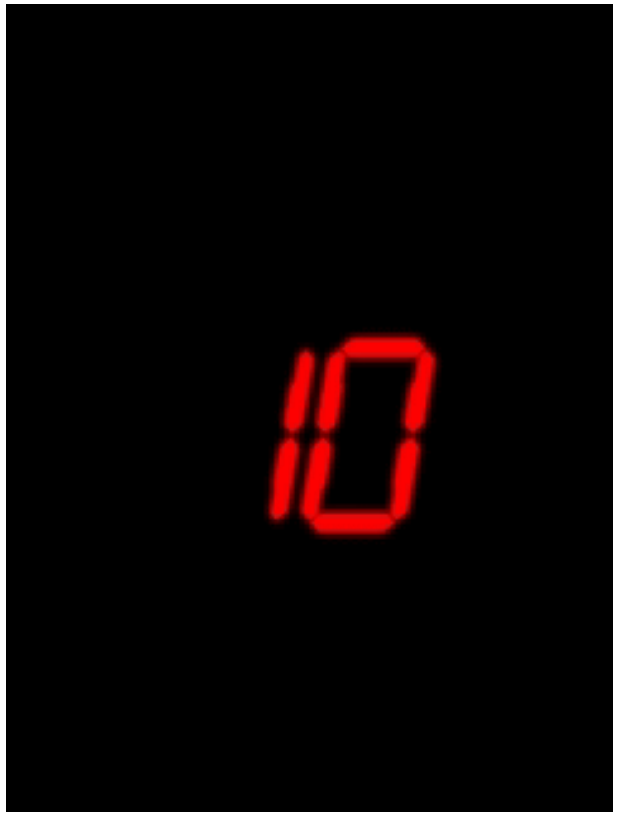


E’ tempo di pratica…

Scriviamo adesso il nostro primo componente React.

Crea un Componente React che rappresenta un contatore (come quello mostrato di fianco) che parte da 10 e si decrementa di una unità ogni secondo mostrando il valore aggiornato a video.

Ogni volta che il contatore arriva a 0 deve ripartire.



Come naviga tra diverse pagine un sito web tradizionale?

Come naviga una SPA(Single-Page-Application)?

Navigazione tra componenti

Il meccanismo che offre la percezione di navigazione tra un componente e l'altro di una SPA è chiamato **Routing**.

Il routing è rappresentato da un insieme di rotte.

Ogni rotta è un'associazione di un percorso ad un componente.

La libreria utilizzata da React per implementare questo meccanismo è *react-router-dom,* per installarla all'interno di un progetto si esegue questo comando:

```
> npm i react-router-dom
```

Routing

Una volta installata la libreria, per costruire il nostro componente di Routing, abbiamo bisogno di importare *BrowserRouter*, *Routes* e *Route.*

```
import ReactDOM from "react-dom/client";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Layout from "./pages/Layout";
import Home from "./pages/Home";
import Blogs from "./pages/Blogs";
import Contact from "./pages/Contact";
import NoPage from "./pages/NoPage";

export default function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="blogs" element={<Blogs />} />
          <Route path="contact" element={<Contact />} />
          <Route path="*" element={<NoPage />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

useLocation

L’hook **useLocation** restituisce l’oggetto location del browser aggiornato e può essere utile per reagire al cambiamento della posizione corrente.

```
import * as React from 'react';
import { useLocation } from 'react-router-dom';

function App() {
  let location = useLocation();

  React.useEffect(() => {
    // Google Analytics
    ga('send', 'pageview');
  }, [location]);

  return (
    // ...
  );
}
```

useNavigate

L’hook **useNavigate** restituisce una funziona che permette di navigare in maniera imperativa l’applicazione.

```
import { useNavigate } from "react-router-dom";

function useLogoutTimer() {
  const userIsInactive = useFakeInactiveUser();
  const navigate = useNavigate();

  useEffect(() => {
    if (userIsInactive) {
      fake.logout();
      navigate("/session-timed-out");
    }
  }, [userIsInactive]);
}
```

E’ tempo di pratica…

Crea un Applicazione React che presenta i seguenti componenti:

- **Header**: deve avere un logo e un titolo
- **Footer**: deve avere lo stesso logo e titolo dell’Header e in più deve mostrare delle informazioni di contatto.
- **Navbar**: composta da due voci di menu,
    1. **Home**, che è anche la pagina iniziale dell’applicazione e presenta un testo di Benvenuto e un tasto “Vai al Timer” che porta sull’altra Rotta dell’applicazione (descritta sotto).
    2. **Timer**, che mostra il componente sviluppato nell’esercizio precedente.

Experience.
Create.
Inspire.

Thank You

Vitale Esca

IBM Client Innovation Center
Italy