JavaScript ReactJs

Introduzione a Reactjs

Giovanni Iovino: giovanni.iovino@intecs.it

Contenuto:

Introduzione a React: Componenti/Classi/JSX

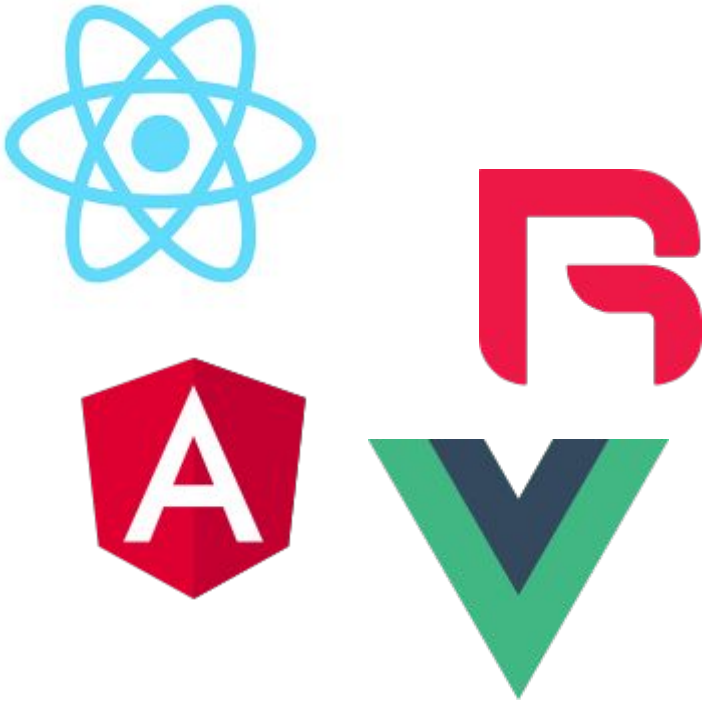
React - Cosa è

React è una **libreria** open-source front-end per lo sviluppo di componenti UI e GUI.

Originariamente sviluppata come prototipo interno da un ingegnere di Facebook, è stata rilasciata pubblicamente nel 2013.

Esistono altre librerie e framework che permettono di creare interfacce utente basate su HTML5/CSS3/Js:

- AngularJs - Sviluppato originariamente da Google. Usa il Model-View-Controller pattern
- VueJs - Nato dalla collaborazione di persone di differenti società
- … molte altre (ember.js, riot.js etc.)



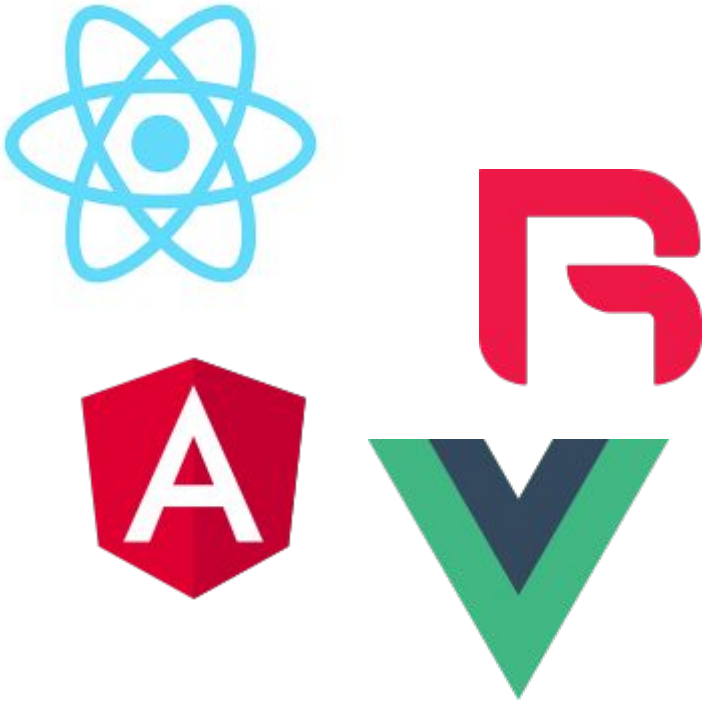
Perché React?

La scelta di una libreria o di un framework per un progetto può dipendere da diversi fattori:

- Esperienza pregressa del team di sviluppo con determinate librerie
- Requisiti specifici del progetto
- Complessità del progetto

React è solo una delle soluzioni che si possono utilizzare per la realizzazione di interfacce web. Alcuni motivi per scegliere React possono essere:

- Largamente utilizzato dagli sviluppatori di tutto il mondo
- Permette l'utilizzo di librerie di terze parti
- Performance elevate (Virtual DOM)
- Semplicità e composizione



Creare un'app react in Node

Ci sono diversi modi per utilizzare React in un progetto:

https://reactjs.org/docs/create-a-new-react-app.html

Per creare un nuovo progetto basato su React in node, dalla linea di comando possiamo utilizzare il seguente comando:

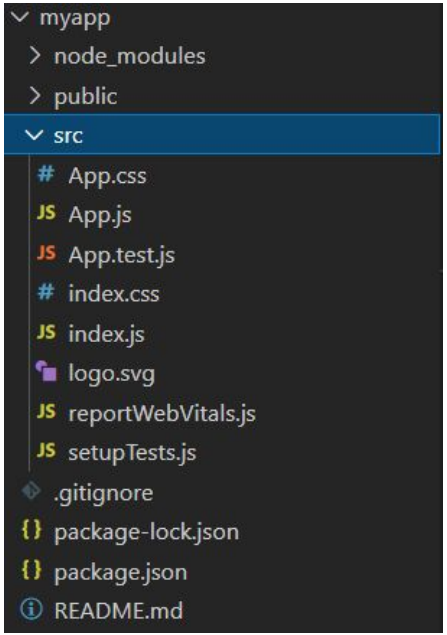
npx create-react-app <nome-app>

Il comando crea la cartella "nome-app", installa le librerie React e React-DOM con le loro dipendenze, configura il package.json e crea i file di un'applicazione di esempio.

https://create-react-app.dev/docs/getting-started

Nota: npx non va confuso con **npm**. Npx è un comando disponibile con le versioni di npm > 5.2. Permette di eseguire dei comandi CLI che non sono stati installati globalmente in precedenza con npm (es. create-react-app)

```
$:> npx create-react-app myapp
```



React - Package.json

Aprendo il file package.json possiamo vedere le librerie e le dipendenze installate. Inoltre notiamo che sono creati automaticamente degli script:

- **start -** Avvia l'app in modalità sviluppatore. Viene creato un web server ed è possibile testare l'app collegandosi con un browser all'indirizzo http://localhost:3000
- **build -** Transpila il codice dell'app convertendolo in codice JS e HTML in modo che possa essere pubblicato per la versione di produzione dell'app
- **test -** Esegue i test scritti per l'app se esistono
- **eject -** Operazione che rimuove le configurazioni di build di default e le dipendenze. Se applicata è irreversibile.

Gli script sono eseguibili con il comando:

npm run <nome-comando>

Per il comando start si può usare direttamente:

npm start

```
{
  "name": "myapp",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.11.10",
    "@testing-library/react": "^11.2.6",
    "@testing-library/user-event": "^12.8.3",
    "react": "^17.0.2",
    "react-dom": "^17.0.2",
    "react-scripts": "4.0.3",
    "web-vitals": "^1.1.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

React - Virtual DOM

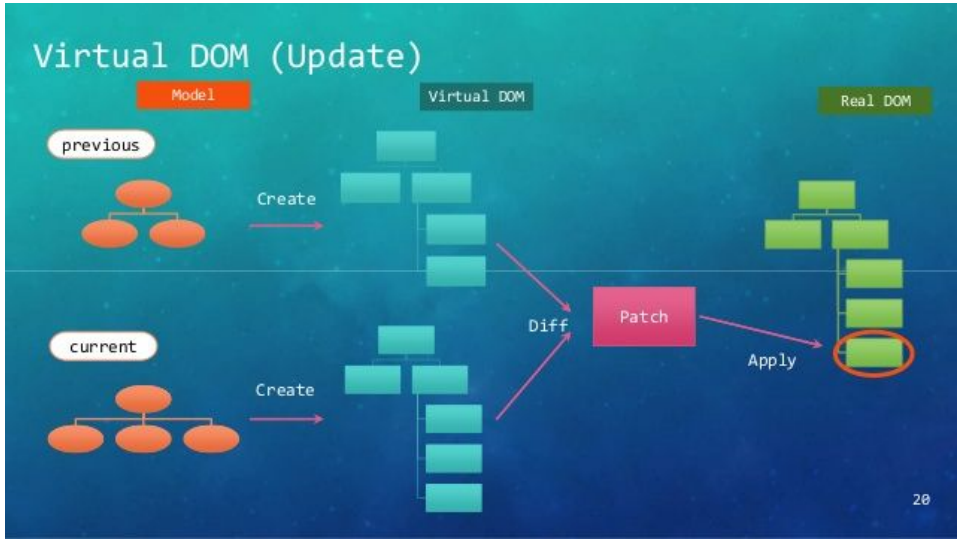
Come funziona React internamente?

Virtual DOM

Sappiamo che il DOM è l'oggetto che rappresenta la struttura della pagina HTML all'interno del browser e che è accessibile da JS.

Il Virtual DOM è una rappresentazione del DOM interna alla libreria React che permette di ottimizzare la modifica degli elementi del DOM programmaticamente. React utilizza il Virtual DOM per determinare quali componenti devo essere aggiornati a fronte di un cambio di stato all'interno della nostra applicazione, evitando di dover aggiornare l'intera pagina, questa operazione è chiamata **reconciliation.**

Il Virtual DOM astrae la manipolazione degli attributi, degli elementi e la gestione degli eventi.



React - Componenti

In react le interfacce utente sono realizzate tramite la composizione di componenti:

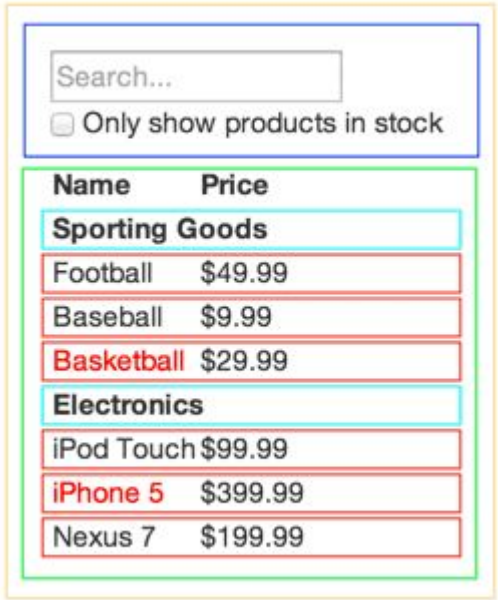
- Possono essere definiti come **classi** o come **funzioni**
- Ogni componente ha il suo **life-cycle**
- I componenti possono essere annidati
- I componenti possono essere stateful e stateless:
    - stateful: utilizzano uno stato interno per il loro funzionamento interno.
    - stateless: non fanno uso di uno stato interno per il loro funzionamento e il loro comportamento dipende solo da variabili esterne al componente.

React - Componenti

I componenti sono composti da:

- Una parte che ne descrive la UI. Si utilizza **JSX** per descrivere la parte grafica del componente.
- Il codice che gestisce il ciclo di vita del componente e le sue funzionalità.

I componenti sono normalmente scritti come unità autosufficienti che assolvono ad un compito preciso e possono essere riutilizzati all'interno dell'app o in più app.



React - JSX

JSX è una sintassi specifica di React con cui descriviamo la struttura della parte grafica del componente.

Simile ad HTML, ma non è HTML! E' un'estensione della sintassi Javascript specifica di React

Quando facciamo la build di un'app React il codice Js e JSX della nostra app sono **transcompilati** in Javascript e HTML affinché i browser riescano ad interpretarne il contenuto.

JSX permette l'uso di espressioni all'interno degli elementi JSX che sono definibili all'interno di parentesi graffe.

In JSX possiamo usare tutti gli elementi HTML più tag aggiuntivi che rappresentano altri componenti scritti in React. Si utilizza CSS per lo stile dei componenti

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

//JSX example syntax with expression
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

//JSX example syntax with expression
const a = 10;
const b = 5;
const element = <h1>Hello, {a+b}</h1>;

//Components define new tags we can use in JSX
<Component1 prop1="20">
  <Component2 c2prop={[10,20,30]}>
    Ciao
  </Component2>
  <Component3 c3prop="ok"/>
</Component1>
```

React - Component Class

I componenti classe permettono la scrittura di componenti React tramite l'uso delle classi JS.

I componenti classe di React estendono la classe **React.Component,** da cui ereditano i metodi e le proprietà necessarie al funzionamento del componente all'interno del runtime di React (es. gestione del ciclo di vita, iniezione delle props)

L'unico metodo della classe super React.Component che va sovrascritto obbligatoriamente all'interno di un componente classe è il metodo **render** che deve restituire un oggetto JSX che rappresenta la UI del componente.

```
import React from 'react';

class ExampleComponent extends React.Component {

    render() {
      let a = 10;
      return (
        <div>
          <h1>Hello, world!</h1>
          <h2>It is a great day to learn React.</h2>
          <p>a = {a};</p>
        </div>
      );
    }
}
```

React - Props

I componenti possono ricevere delle proprietà (**props**) dall'esterno. Le props sono passate come primo argomento del costruttore della classe.

Nota: le props sono di sola lettura e non possono essere modificate.

Le props sono oggetti JS, quindi possono avere un qualsiasi tipo previsto da Javascript (comprese le funzioni).

Nel costruttore di un componente vengono dichiarate a inizializzate anche le variabili interne del componente.

```
import React from 'react';
export default class ExampleComponent extends React.Component {
    constructor(props){
        super(props); //Importante
        this.props = props; //Ora possiamo accedere alle props anche
nelle altre funzioni del componente
        this.myvar = 10;
    }
    render() {
      return (
        <div>
          <h2>It is a great day to learn React.</h2>
          <p>a = {this.myvar};</p>
          <p>props.ok = {this.props.ok};</p>
        </div>
      );
    }
}
```

```
import React from 'react';
import ExampleComponent from './examplecomponent.js'

//Other component file ...
render(){
    return(
        <ExampleComponent ok = "OK!" />
    );
}
```

React - Props

E’ possibile e consigliabile assegnare dei valori di default alle props.

Per i componenti classe, si può fare assegnando un valore alla variabile statica **defaultProps** della classe che rappresenta il componente.

```
import React from 'react';
class ExampleComponent extends React.Component {
    static defaultProps={
        ok:"OK!",
        counter:0
    }
    constructor(props){
        super(props);
        this.props = props;
        this.myvar = 10;
    }
    render() {
       // ...
    }
}
export default ExampleComponent;
```

```
import React from 'react';
import ExampleComponent from './examplecomponent.js'

//Other component file ...
render(){
    return(
        <ExampleComponent />
    );
}
```

React - Props

Anche il contenuto di un tag JSX è passato come prop ed è accessibile tramite **props.children**

```
import React from 'react';
class ExampleComponent extends React.Component {
    static defaultProps={
        ok:"OK!",
        counter:0
    }
    constructor(props){
        super(props);
        this.props = props;
        console.log(this.props.children);
    }
    render() {
       // ...
    }
}
export default ExampleComponent;
```

```
import React from 'react';
import ExampleComponent from './examplecomponent.js'

//Other component file ...
render(){
    return(
    <ExampleComponent  a={10}>
      <p>P1</p>
      <p>P2</p>
    </ExampleComponent>

    );
}
```

React - Eventi & this

Essendo di fatto delle classi JS i component class di React possono avere anche metodi interni.

Attenzione: In caso i metodi di un componente vengano utilizzati per la gestione di eventi è necessario utilizzare il binding esplicito per non perdere il riferimento al this!

Nota: In JSX gli attributi (props) degli elementi seguono la sintassi del DOM in JS (camelCase) e non quella di HTML

```
import React from 'react';

export default class ExampleComponent extends React.Component {

    constructor(props){
        super(props);
        this.props = props;
        this.myvar = 10;
        this.myMethod = this.myMethod.bind(this);
    }

    myMethod(){
       //Questo metodo fa uso del this
        console.log(this.myvar);
    }

    render() {
      return (
        <div>
          <h1>Hello, world!</h1>
          <h2>It is a great day to learn React.</h2>
          <p>a = {this.props.a};</p>
          <button onClick={this.myMethod}>click me</button>
        </div>
      );
    }
}
```

React - Stato

I componenti che dichiarano nel costruttore la variabile **state** sono chiamati componenti stateful.

Lo stato è specifico di un'istanza di un componente e può cambiare durante il ciclo di vita del componente.

L'aggiornamento dello stato di un componente si ottiene utilizzando il metodo **setState** ereditato dalla classe React.Component.

Nota: E' possibile aggiornare parzialmente lo stato di un componente (es. quando va aggiornata solo una particolare proprietà dello stato). React si occuperà di effettuare il merge automatico tra lo stato complessivo e le parti da aggiornare.

Attenzione: setState è asincrona, inoltre React per ottimizzare la gestione dei componenti potrebbe eseguire più aggiornamenti dello stato raggruppandoli. Evitare di utilizzare espressioni che riguardano il valore dello stato in un'operazione di setState

```
import React from 'react';

export default class ExampleComponent extends React.Component {

    constructor(props){
        super(props);
        this.props = props;

        this.state={
            counter: this.props.counter||0,
            pippo:{
                a:10,
                b:11
            }
        }

        this.updateCounter = this.updateCounter.bind(this);
    }
    updateCounter(){
        const counter= this.state.counter + 1;
        this.setState({
            counter: counter
        });
    }

    render() {
      return (
        <div>
          <h1>Hello, world!</h1>
          <h2>It is a great day to learn React.</h2>
          <p>counter = {this.state.counter};</p>
          <button onClick={this.updateCounter}>increment counter</button>
        </div>
      );
    }
}
```

React - Aggiornamento UI

La UI di un componente React è aggiornata:

- Se cambiano le props passate al componente
- Se cambia lo stato del componente

React si occuperà tramite la reconciliation di aggiornare solo gli elemente HTML affetti dai cambiamenti.

```
import React from 'react';
//Stateless
class InnerComponent extends React.Component{
    constructor(props){
        super(props);
        this.props=props;
    }
    render(){
        return <div>I just know that n is {this.props.n}</div>;
    }
}

//Stateful
export default class ExampleComponent extends React.Component {

    constructor(props){
        super(props);
        this.props = props;

        this.state={
            counter: this.props.counter||0,
            n: 100
        }

        this.updateCounter = this.updateCounter.bind(this);
    }
    updateCounter(){
        const counter= this.state.counter + 1;
        this.setState({
            counter: counter,
            n: Math.ceil(Math.random()*100)
        });
    }

    render() {
      return (
        <div>
          <h1>Hello, world!</h1>
          <h2>It is a great day to learn React.</h2>
          <p>counter = {this.state.counter};</p>
          <p>this.n = {this.state.n};</p>
          <InnerComponent n={this.state.n}/>
          <button onClick={this.updateCounter}>increment counter</button>
        </div>
      );
    }
}
```

React - Utilizzo di CSS

Esistono diversi modi per utilizzare CSS nei componenti React:

https://it.reactjs.org/docs/faq-styling.html

Per i componenti classe, un metodo è quello di utilizzare i **moduli CSS:**

https://github.com/css-modules/css-modules

Un modulo css è un file con estensione .css che ha il nome del file scritto secondo la sintassi:

<componentName>.module.css

Un componente può importare il proprio modulo CSS e assegnare gli stili ai tag HTML utilizzando l'attributo **className.**

```
.myClass{
  color:#cc0000;
}
```

```
import React from 'react';
import styles from './MyComponent.module.css'
//Stateless
class MyComponent extends React.Component{
    constructor(props){
        super(props);
        this.props=props;
    }
    render(){
        return <div className={styles.myClass}>I just know
that n is {this.props.n}</div>;
    }
}
```