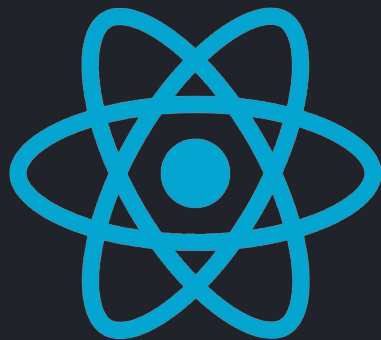


---

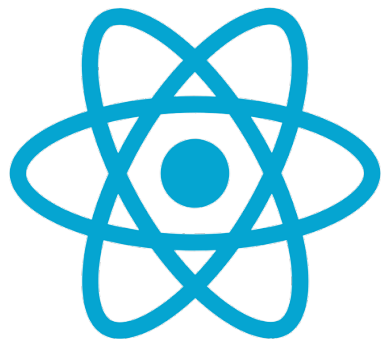
# Oggi parliamo di...

# React Js



Delisio Roberto

---



---

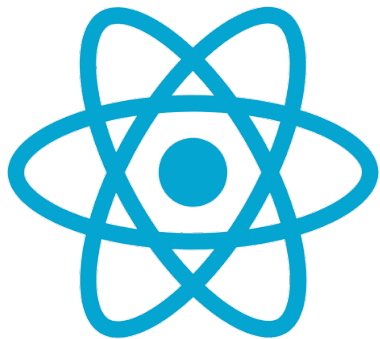
# Risorse utili e prerequisiti

Vediamo alcune risorse online che possono risultare particolarmente utili:

- Node.js
- Chrome
- Visual Studio Code
- [HTML to JSX Compiler](https://htmltojsx.com/)
- <https://codepen.io/>

Cosa dobbiamo conoscere

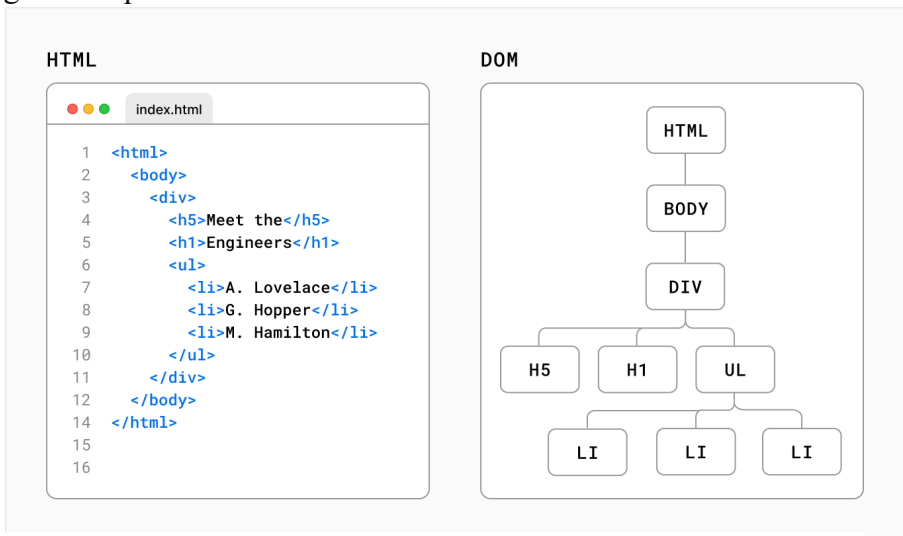
- HTML
  - CSS
  - Javascript
  - Bootstrap
-



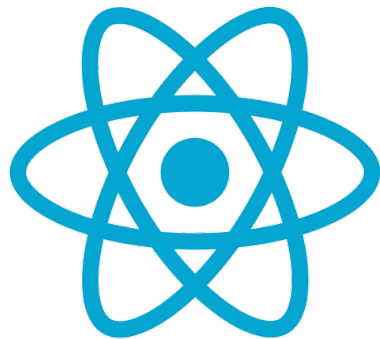
# Come Funziona il DOM

Per capire come funziona React, abbiamo prima bisogno di una conoscenza di base di come i browser interpretano il tuo codice per creare interfacce utente interattive (UI).

Quando un utente visita una pagina Web, il server restituisce al browser un file HTML che potrebbe assomigliare a questo:

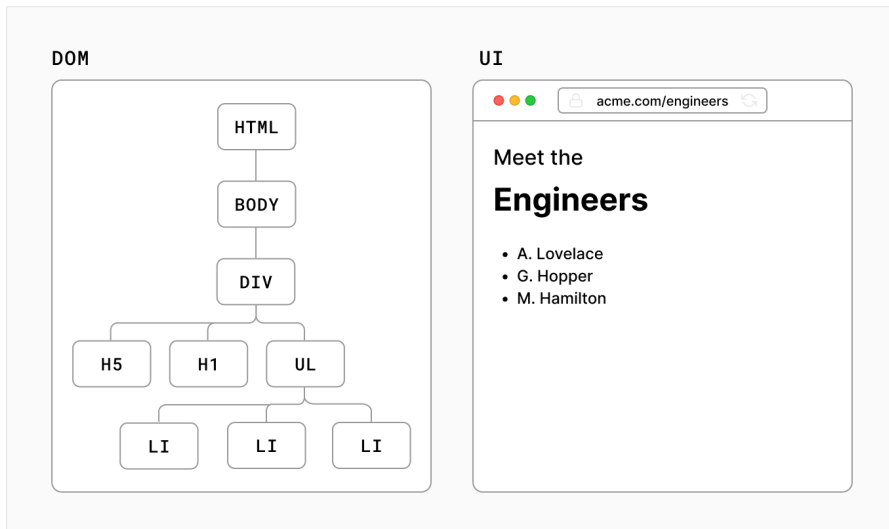


Il browser legge quindi l'HTML e costruisce il **Document Object Model (DOM)**.

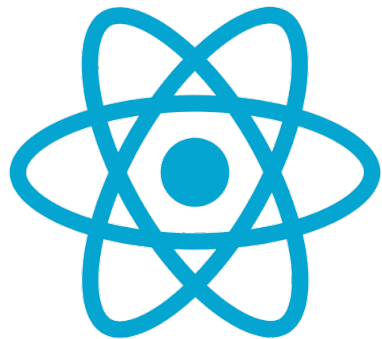


# Cos'è il DOM?

Il DOM è una rappresentazione di oggetti degli elementi HTML. Funge da ponte tra il codice e l'interfaccia utente e ha una struttura ad albero con relazioni padre e figlio.



Puoi utilizzare metodi DOM e un linguaggio di programmazione, come JavaScript, per ascoltare gli eventi utente e [manipolare il DOM](#) selezionando, aggiungendo, aggiornando ed eliminando elementi specifici nell'interfaccia utente. La manipolazione del DOM ti consente non solo di indirizzare elementi specifici, ma anche di cambiarne lo stile e il contenuto.



---

# Cos'è REACTJS

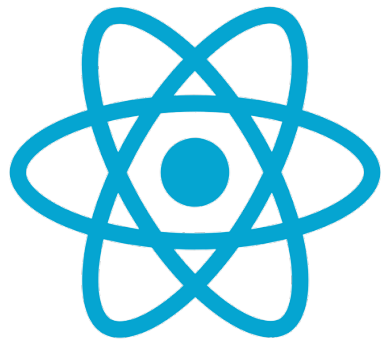
React è una **libreria JavaScript** per la creazione di interfacce utente (UI, User Interface). Sviluppata nel 2013 all'interno di Facebook, adesso React è una libreria open-source supportata da una grande community di programmatori.

React consente di sviluppare applicazioni dinamiche che non necessitano di ricaricare la pagina per visualizzare i dati modificati. Inoltre nelle applicazioni React le modifiche effettuate sul codice si possono visualizzare in tempo reale, permettendo uno sviluppo rapido, efficiente e flessibile delle applicazioni web.

## React è la “V” di MVC

Se dovessimo inquadrare React all'interno di un paradigma, potremmo dire che esso è la “V” di MVC: nasce per gestire la parte relativa alle view, ossia alla presentazione, e all'intercettazione degli eventi di input dell'utente, senza forzare l'adozione di specifiche funzioni né limitare l'interfacciamento ad altre librerie per quanto riguarda l'eventuale comunicazione con un server di backend, o specifiche architetture di binding ai dati, frangenti in cui lo sviluppatore ha libera scelta sugli strumenti con cui integrare React.

---



---

# Perché usare REACTJS

React è probabilmente la prima libreria JavaScript che nasce con una vocazione specifica: diventare la soluzione definitiva per sviluppatori front-end e app mobile basate su HTML5, la proverbiale "panacea per tutti i mali", il tool che permetta di costruire interfacce utente dinamiche e sempre più complesse rimanendo comunque semplice e intuitivo da utilizzare.

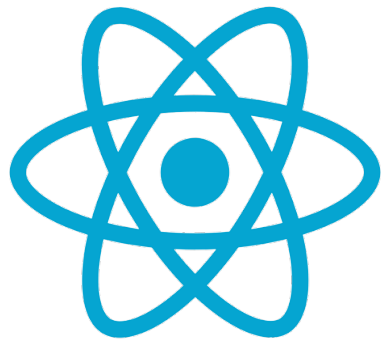
Creata da Facebook, React è la colonna portante del social network più popolare del mondo e su di essa si basa l'interfaccia Web di Instagram.

La creazione di applicazioni Web, indipendentemente dal framework scelto per lo sviluppo, coinvolge necessariamente i tre linguaggi fondamentali della piattaforma: HTML per la struttura, CSS per la stilizzazione e JavaScript per la logica applicativa.

Le pagine complete invece sono ridotte al minimo, anzi molto spesso a una sola, tant'è vero che queste applicazioni prendono il nome di Single Page Applications (SPA) e che servono da "contenitore" in cui creare e gestire l'interfaccia utente.

La forza di React rispetto ad altre librerie è quella di consentire l'uso di un approccio dichiarativo simile all'HTML, quindi molto familiare, per definire i componenti che rappresentano parti significative e logiche dell'interfaccia utente, ad esempio un commento a un articolo, o la lista degli stessi commenti.

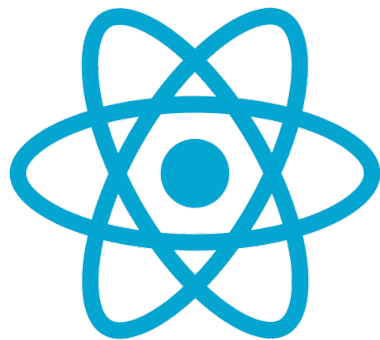
---



---

# Perché usare REACTJS

- È [open source](#), vale a dire che è possibile contribuire a esso inviando problemi o richieste pull. (*Proprio come [questi documenti](#)*)
  - È [dichiarativo](#), ovvero si scrive il codice desiderato e React accetta il codice dichiarato ed esegue tutti i passaggi JavaScript/DOM per ottenere il risultato desiderato.
  - Si tratta [di un componente](#), vale a dire che le applicazioni vengono create usando moduli di codice indipendenti prefabbricati e riutilizzabili che gestiscono il proprio stato e possono essere associati con il framework di React, rendendo possibile passare i dati attraverso l'app mantenendo lo stato fuori dal DOM.
  - Il motto React è "Imparare una volta, scrivere ovunque". L'intenzione è quella di [riutilizzare il codice](#) e di non fare ipotesi su come si userà React utente con altre tecnologie, ma per rendere riutilizzabili i componenti senza la necessità di riscrivere il codice esistente.
-

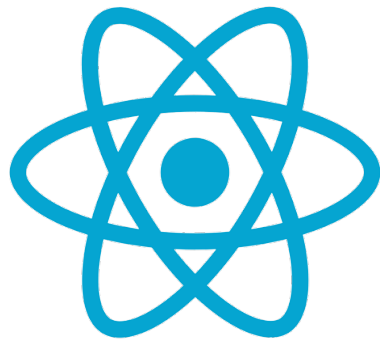


---

# Perché usare REACTJS

- [JSX](#) è un'estensione di sintassi per JavaScript scritta da usare con React, simile a HTML, ma è in realtà un file JavaScript che deve essere compilato o convertito in JavaScript normale.
  - [DOM virtuale](#): [DOM](#) è l'acronimo di Document Object Model e rappresenta l'interfaccia utente dell'app. Ogni volta che lo stato dell'interfaccia utente dell'app cambia, il DOM viene aggiornato per rappresentare la modifica. Quando un DOM viene aggiornato di frequente, le prestazioni diventano lente. Un DOM virtuale è solo una rappresentazione visiva del DOM, quindi quando lo stato dell'app cambia, il DOM virtuale viene aggiornato anziché il DOM reale, riducendo il costo delle prestazioni. Si tratta di una *rappresentazione* di un oggetto DOM, ad esempio una copia leggera.
  - *Visualizzazioni* sono gli elementi visualizzati dall'utente visualizzati nel browser. In React, la visualizzazione è correlata al concetto di [elementi di rendering](#) che si desidera che un utente visualizzi sullo schermo.
-





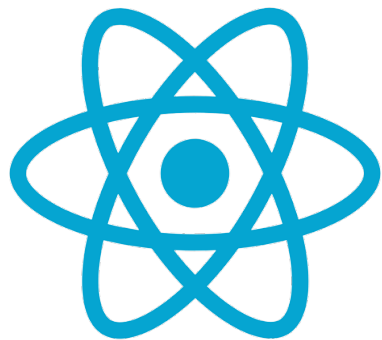
---

# Perché usare REACTJS

- Stato: fa riferimento ai dati archiviati da visualizzazioni diverse. Lo stato si basa in genere sull'utente e sulle operazioni eseguite dall'utente. Ad esempio, l'accesso a un sito Web può mostrare il profilo utente (visualizzazione) con il nome (stato). I dati sullo stato cambieranno in base all'utente, ma la visualizzazione rimarrà invariata.

Benché dichiarativa, la rappresentazione del componente in realtà si traduce in chiamate all'API di React che intervengono - nel modo più veloce e performante possibile - sul DOM della pagina per creare gli elementi necessari.

---



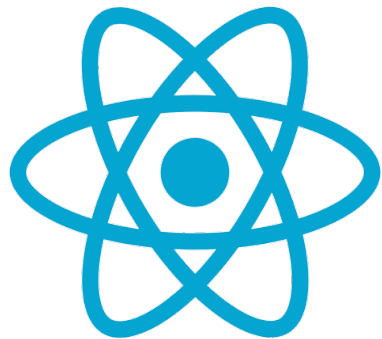
---

# Differenze con JQuery

Rispetto a JQuery, React non richiede l'assegnazione obbligatoria di ID univoci o classi, né richiede allo sviluppatore di intervenire direttamente sul DOM, ma è la libreria stessa che si fa carico di questo compito in base alla struttura del componente dichiarato in `primis` e, in secondo luogo, in base allo stato interno del componente Web e alla variazione dei valori delle proprietà assegnate allo stesso.

In breve, React esclude un intervento diretto sul DOM, lasciando a noi il compito di definire i componenti che costituiscono l'interfaccia dell'applicazione e i dati da utilizzare per la generazione del markup. Sarà la libreria ad intervenire sul DOM di conseguenza, utilizzando peraltro il modo che garantisce le maggiori performance possibili.

---



---

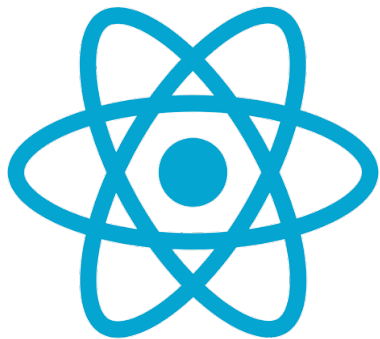
# Differenze con AngularJS

AngularJS è una soluzione più completa, a discapito di una complessità maggiore di apprendimento e uso, laddove React si presenta più focalizzato e accessibile all'apprendimento, con la possibilità di ricorrere a librerie note (es. anche la stessa JQuery) per le parti che non sono strettamente legate all'interfaccia utente.

La contestazione maggiore che viene mossa a React da parte degli sviluppatori AngularJS è quella di mescolare logica di gestione degli eventi alla presentazione, riducendo quindi la separazione delle responsabilità degli elementi del progetto.

Questo approccio viene giustificato da React con il criterio della “Separation of Concerns” a discapito della “Separation of Responsibility”: in termini pratici, con React si tende a isolare codice e lo stato dei componenti in base al loro ambito di utilizzo, rendendoli il più possibile autonomi e riutilizzabili in un medesimo contesto, piuttosto che estremizzare la separazione della parte di presentazione e del codice che la governa.

---



---

# Usiamo la libreria react e ReactDOM

React è una libreria e possiamo importarlo ed usarlo in qualsiasi pagina html per aiutarci a generare layout reattivi.

## React

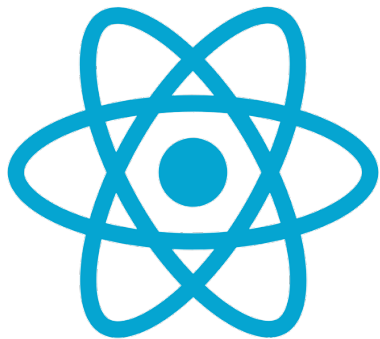
React è una libreria JavaScript sviluppata da Facebook per costruire interfacce utente. È basata sulla creazione di componenti riutilizzabili, che gestiscono il proprio stato e si compongono per formare applicazioni complesse. React è focalizzata principalmente sulla creazione della vista dell'applicazione, permettendo agli sviluppatori di definire come l'app dovrebbe apparire in base ai cambiamenti nei dati.

## ReactDOM

ReactDOM è un pacchetto che fornisce metodi specifici del DOM (Document Object Model) che possono essere utilizzati con React. La funzione più nota di ReactDOM è ReactDOM.render(), che viene utilizzata per montare un componente React al DOM. Questo è il punto di ingresso per un'applicazione React in una pagina web, collegando la logica React all'interfaccia utente definita nel markup HTML.

```
<!-- React e ReactDOM --> <script crossorigin  
src="https://unpkg.com/react@18/umd/react.development.js"></script> <script crossorigin  
src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
```

---



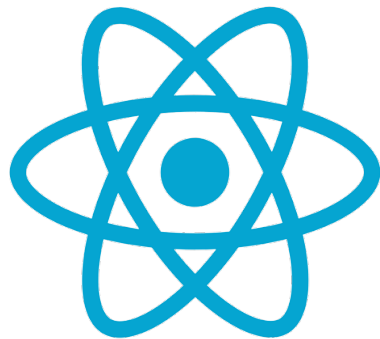
# Usiamo la libreria react e ReactDOM

React è una libreria e possiamo importarlo ed usarlo in qualsiasi pagina html per aiutarci a generare layout reattivi. Creiamo una semplice pagina HTML ed importiamo i file react.js e reactdom.js

```
2 <html>
3
4   <head>
5     <meta charset="UTF-8" />
6     <title>Hello World</title>
7     <!--
8     <script src="https://unpkg.com/react@18/umd/react.development.js"></script>
9     <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
10  -->
11  <script src="react.js"></script>
12  <script src="reactdom.js"></script>
13  </head>
14
15  <body>
16    <div id="root"></div>
17
18  </body>
19
20
21 </html>
```

---

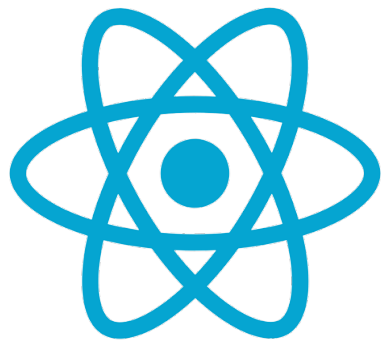
# Usiamo la libreria react e ReactDOM



Nel div andremo arenderizzare i nostri componenti. Andiamo a creare subito un elemento di react. L'app deve avere un componente principale che sarà root

```
<body>
  <div id="root"></div>

  <script>
    const rootElement=document.getElementById("root");
    const root=ReactDOM.createRoot(rootElement);
    console.log(rootElement);
  </script>
</body>
```



---

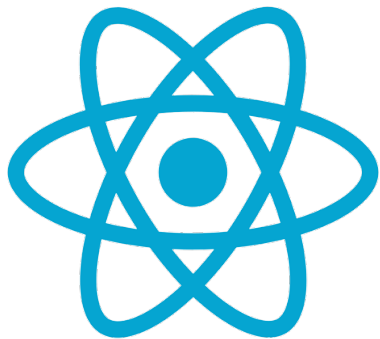
# Usiamo la libreria react e ReactDOM

Consideriamo la creazione di un componente `<h1>` in una normale pagina HTML. Inizialmente, questa pagina è statica, senza interattività. Si potrebbe pensare di aggiungere interattività manualmente, utilizzando event listener direttamente sul DOM. Tuttavia, React opera in modo diverso: non interagisce direttamente con il DOM della pagina. Al contrario, React gestisce un Virtual DOM - una rappresentazione leggera in memoria.

Quando si aggiunge interattività in React, ad esempio aggiungendo un listener di eventi a un componente, React aggiorna il Virtual DOM. Dopodiché, React confronta il Virtual DOM aggiornato con il DOM reale tramite un processo chiamato **'reconciliation'**. Basandosi sulle differenze rilevate, React applica in modo efficiente le modifiche necessarie al DOM reale.

Questo approccio è più performante rispetto all'interazione diretta con il DOM e permette a React di gestire le proprietà degli elementi e lo stato dell'applicazione in modo più efficace.

---



---

# Usiamo la libreria react e ReactDOM

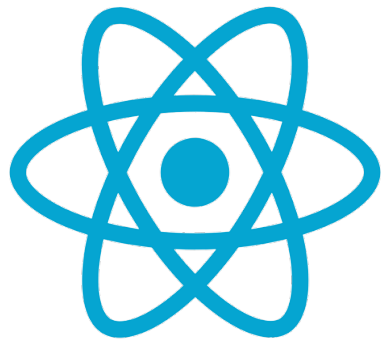
Possiamo creare elementi HTML e passargli gli attributi, tipo style, class che diventa className ed eventi tipo onClick.

```
<script>
  const rootElement = document.getElementById('root');

  const root = ReactDOM.createRoot(rootElement);

  const h1 = React.createElement('h1', {
    style: {
      color: 'green',
      width: '200px'
    },
    className: 'big',
    onClick: ele => { console.log(ele.target) }
  },
    'Hello Worl|'
  );
  root.render(h1);
</script>
```





---

# Usiamo la libreria react e ReactDOM

Supponiamo ora di voler creare un div chiamiamolo `main` e renderizziamo al suo interno `h1`.

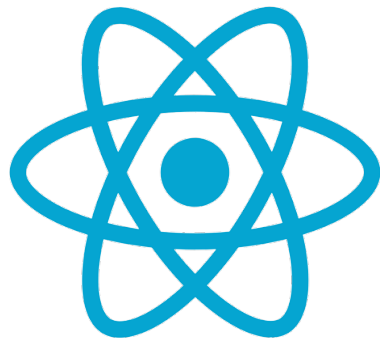
```
    onClick: ele => { console.log(ele.target) }  
  },  
  'Hello World'  
);  
const div = React.createElement('div', {  
  className: 'main'  
},  
  h1  
);
```

```
<style>  
  .main {  
    background-color: ■ brown;  
    color: ■ white;  
    border: 5px;  
    border-radius: 10px;  
  }  
</style>
```

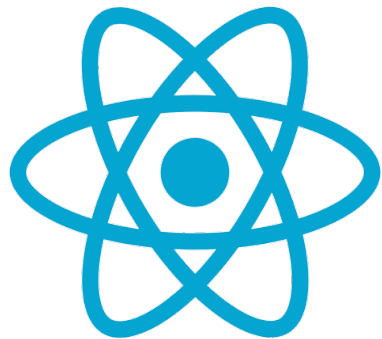
---

# Usiamo la libreria react e ReactDOM

Vediamo ora con aggiungere un altro elemento come una lista



```
const ul=React.createElement("ul",{  
  className:"list"  
},[  
  React.createElement("li",{key:'php'},'PHP'),  
  React.createElement("li",{key:'javascript'},'Javascript'),  
  React.createElement("li",{key:'angular'},'Angular'),  
])
```



---

# Usiamo la libreria react e ReactDOM

Nel contesto di React, la prop `'key'` è utilizzata per aiutare React a identificare quali elementi sono cambiati, sono stati aggiunti, o sono stati rimossi tra diversi render. Le keys dovrebbero essere assegnate agli elementi all'interno di un array per dare agli elementi una identità stabile:

Quando si crea un elenco di elementi React con `'React.createElement'`, l'uso della prop `'key'` è particolarmente importante in situazioni dove l'ordine degli elementi può cambiare tra i render. Senza una key che identifichi ciascun elemento in maniera univoca, React non avrà modo di sapere quale elemento è stato modificato, aggiunto o rimosso, il che potrebbe portare a prestazioni inefficienti e potenziali errori nello stato dell'interfaccia utente.

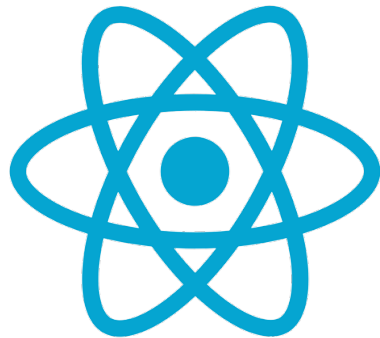
Le keys `'php'`, `'javascript'` e `'angular'` sono utilizzate per identificare univocamente ciascun `<li>` nell'array. Questo significa che se l'ordine degli elementi cambia in un futuro render, React sarà in grado di riconoscere quali elementi sono stati modificati basandosi sulla loro key e sarà in grado di evitare un rirender completo dell'intera lista. Questo è fondamentale per ottimizzare le prestazioni quando si lavora con liste dinamiche che possono subire mutazioni nel tempo.

---

---

# Usiamo la libreria react e ReactDOM

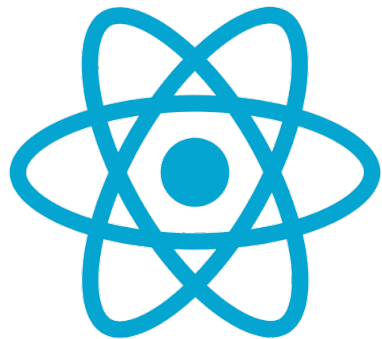
Ecco un esempio di come React potrebbe utilizzare le keys per aggiornare l'interfaccia utente:



- Se l'elemento con key 'php' viene rimosso, React eliminerà solo quell'elemento dalla lista, invece di dover rirenderizzare l'intera lista.
- Se un elemento con key 'sql' viene aggiunto all'inizio dell'array, React aggiungerà quell'elemento all'inizio della lista nel DOM senza rirenderizzare gli altri elementi già presenti.

In assenza di keys, React non avrebbe altra scelta se non quella di rirenderizzare tutti gli elementi dell'array per garantire che l'interfaccia utente rifletta l'effettiva sequenza degli elementi nell'array, perché non sarebbe in grado di identificare quale elemento specifico è cambiato.

---



# Usiamo babel e jsx

Creare elementi in questo modo non è certo molto comodo, per questo ~~react~~ ci dà a disposizione un linguaggio che si chiama jsx che ci permette di scrivere javascript come fosse html. Questo è possibile tramite la libreria babel che interpreta il jsx.

Andiamo quindi a creare una nuova pagina e importiamo sia i file di react che la libreria babel. Importiamo anche un file con estensione jsx.

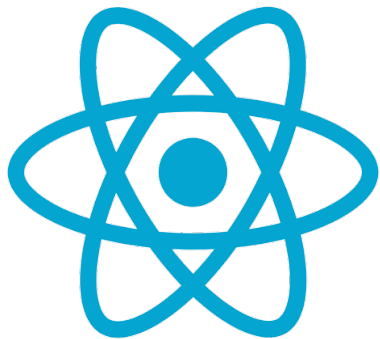
```
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
```

```
<!DOCTYPE html>
<html lang="it">

  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>React with babel</title>
  </head>

  <body>
    <div id="root"></div>
    <!-- end of the page -->
    <script src="https://unpkg.com/react@18/umd/react.prod.js" crossorigin></script>
    <script src="https://unpkg.com/react-dom@18/umd/react-dom.prod.js" crossorigin></script>
    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <script src="script.jsx" type="text/babel"></script>
  </body>

</html>
```



---

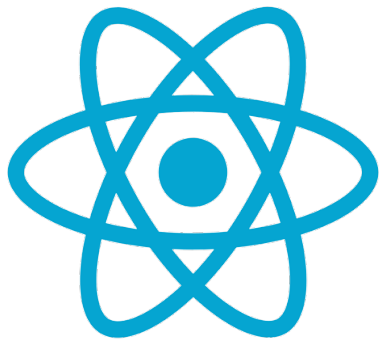
# Usiamo babel e jsx

Nel file jsx creiamo la root della nostra app e poi nella funzione rendere andiamo a scrivere il nostro HTML

```
const rootEle = document.getElementById('root');

const root = ReactDOM.createRoot(rootEle);

root.render(
  <div>
    <h1>Using JSX</h1>
    <ul>
      <li>JAVASCRIPT</li>
      <li>PHP</li>
      <li>HTML</li>
    </ul>
  </div>
);
```



# Usiamo babel e jsx

Strutturiamo ora elementi creati da noi tramite funzioni.

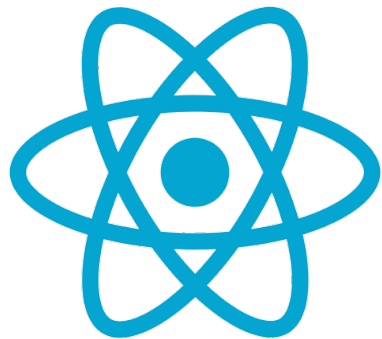
```
const root = ReactDOM.createRoot(rootEle);
const App = (props) => {
  return (
    <main className='main'>
      <h1>This is the main app</h1>
      {props.children}
    </main>
  );
};
root.render(
  <>
    <App>
      <h1>Using JSX</h1>
      <ul>
        <li>JAVASCRIPT</li>
        <li>PHP</li>
        <li>HTML</li>
      </ul>
    </App>
  </>
);
```

```
const rootEle = document.getElementById('root');
const root = ReactDOM.createRoot(rootEle);

const App = ({ children }) => {
  return (
    <main className='main'>
      <h1>This is the main app</h1>
      {children}
    </main>
  );
};

function List() {
  return (
    <ul>
      <li>JAVASCRIPT</li>
      <li>PHP</li>
      <li>HTML</li>
    </ul>
  );
}

root.render(
  <>
    <App>
      <h2>My learning path</h2>
      <List></List>
    </App>
  </>
);
```



---

# Cos'è NODE.JS

Node.js è una runtime di JavaScript Open source multiplatforma orientato agli eventi per l'esecuzione di codice JavaScript, costruita sul motore JavaScript V8 di Google Chrome.

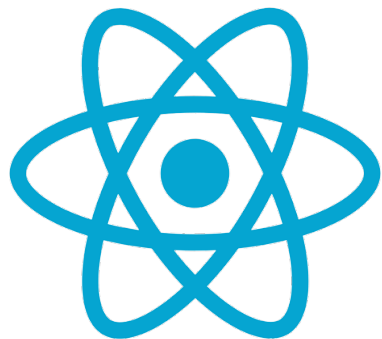
In origine JavaScript veniva utilizzato principalmente lato client. In questo scenario gli script JavaScript, generalmente incorporati all'interno dell'HTML di una pagina web, vengono interpretati da un motore di esecuzione incorporato direttamente all'interno di un Browser. Node.js consente invece di utilizzare JavaScript anche per scrivere codice da eseguire lato server, ad esempio per la produzione del contenuto delle pagine web dinamiche prima che la pagina venga inviata al Browser dell'utente.

Node.js come detto è un runtime di javascript ovvero un programma che, grazie al «motore» V8 di Google Chrome, viene eseguito codice JavaScript lato client

Quando parliamo di «motore» JavaScript parliamo di quel componente sviluppato inizialmente solo per il browser, il cui compito è quello di prendere il nostro codice sorgente JavaScript e trasformarlo in codice macchina:

---





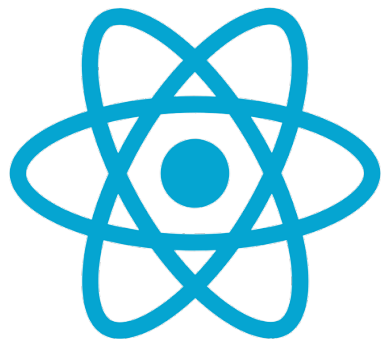
# Cos'è NODE.JS

---



Ogni browser ha il suo «motore» JavaScript specifico Ad esempio FireFox ha « SpiderMonkey », mentre chrome ha « V8 » Inizialmente JavaScript è nato come linguaggio lato client e quindi eseguito dal «motore» residente all'interno del browser.

---



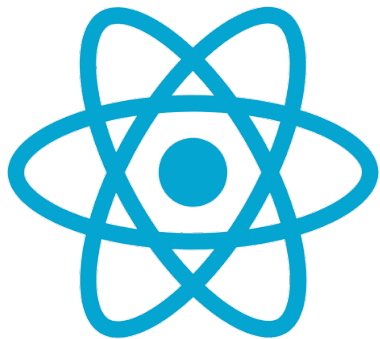
---

# Cos'è NODE.JS



Nel 2009 l'idea «rivoluzionaria» del progettista di Node.js Ryan Dahl è stata quella di prendere il motore V8 ed inserirlo all'interno di un nuovo ambiente in cui eseguire codice JavaScript. Questo ambiente prende il nome di Node js che non opera lato client ma opera lato server

---



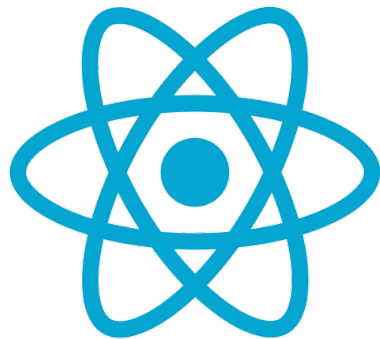
---

# Cos'è NODE.JS

Abbiamo quindi 1 motore JavaScript V 8 e due runtime diversi, uno lato client ed uno lato server Entrambi questi runtime espandono le funzionalità di V 8 in base alla propria esigenza

Ad esempio l'oggetto window e l'oggetto document presente nel runtime browser non è presente nel runtime nodejs in quanto non abbiamo lato server una finestra del browser da gestire, ma ha un altro oggetto necessario al server che vedremo più avanti, cioè l'oggetto **global**

---



---

# Cos'è NPM

**NPM** (Node Package Manager) è il **gestore di pacchetti** ufficiale di Node.js. È uno strumento fondamentale per lo sviluppo JavaScript moderno.

## Cosa fa NPM?

Installa librerie - Scarica e gestisce dipendenze

Gestisce versioni- Controlla compatibilità tra pacchetti

Esegue script- Comandi personalizzati per il progetto

Pubblica pacchetti - Condivide codice con la community

## Componenti principali

### 1. Registry NPM

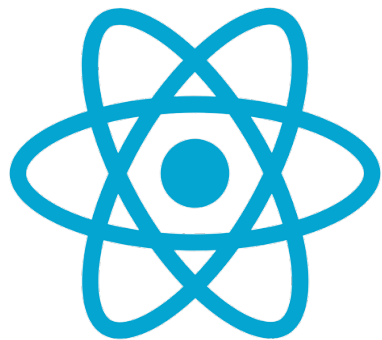
- Database online con milioni di pacchetti JavaScript
- Accessibile su [npmjs.com](https://npmjs.com)
- Pacchetti gratuiti e a pagamento

### 2. CLI (Command Line Interface)

- Strumento da terminale per interagire con NPM
- Si installa automaticamente con Node.js
- Comandi come `npm install`, `npm start`, ecc.

### 3. package.json

- File di configurazione del progetto
  - Elenca dipendenze, script, metadati
  - "Carta d'identità" del progetto
-



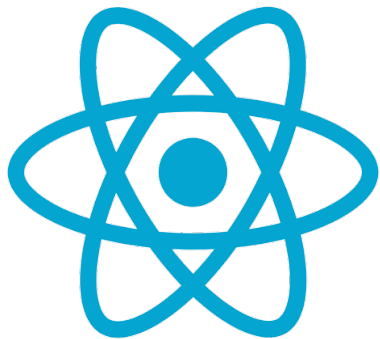
---

# Approccio Asincrono

La caratteristica principale di Node.js risiede nella possibilità che offre di accedere alle risorse del sistema operativo in modalità **event-driven** e non sfruttando il classico modello basato su processi o thread concorrenti, utilizzato dai classici web server.

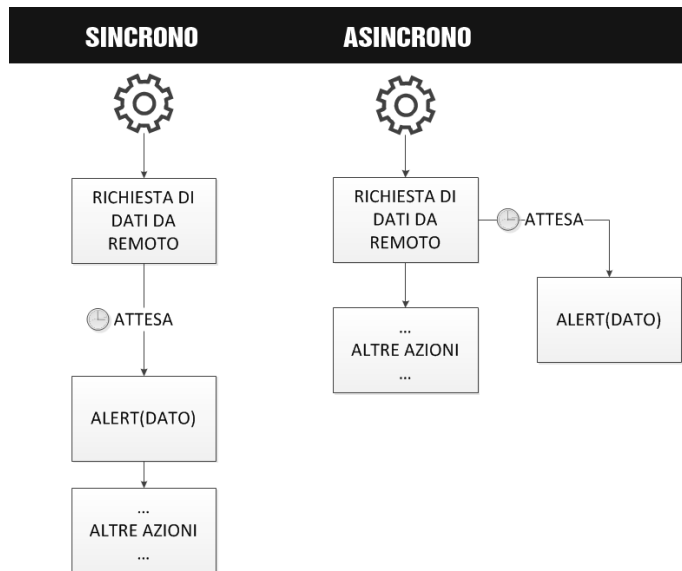
Il modello event driven o “programmazione ad eventi”, si basa su un concetto piuttosto semplice, si lancia una azione quando accade qualcosa. Ogni azione quindi risulta asincrona a differenza dei pattern di programmazione più comune in cui un’azione succede ad un’altra solo dopo che essa è stata completata.

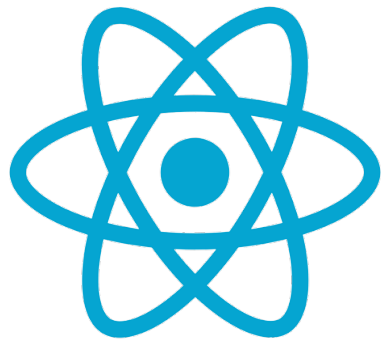
---



# Approccio Asincrono

Grazie al comportamento asincrono durante le attese di una certa azione il runtime può gestire qualcos'altro che ha a che fare con la logica applicativa, ad esempio





---

# Installazione Node.js

Per installare localmente Node.js bisogna andare sul sito ufficiale [www.nodejs.org](http://www.nodejs.org) e scaricare il pacchetto per il nostro sistema operativo

Node.js® è un runtime JavaScript costruito sul motore JavaScript V8 di Chrome.

Join us at OpenJS World, a free virtual event on June 2-3, 2021

Download per Windows (x64)

14.17.0 LTS

Consigliata

16.2.0 Corrente

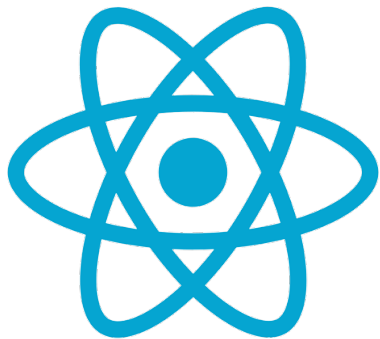
Ultime funzionalità

[Altri download](#) | [Changelog](#) | [Documentazione API](#)

[Altri download](#) | [Changelog](#) | [Documentazione API](#)

Dai un'occhiata alla [tabella di marcia LTS](#).

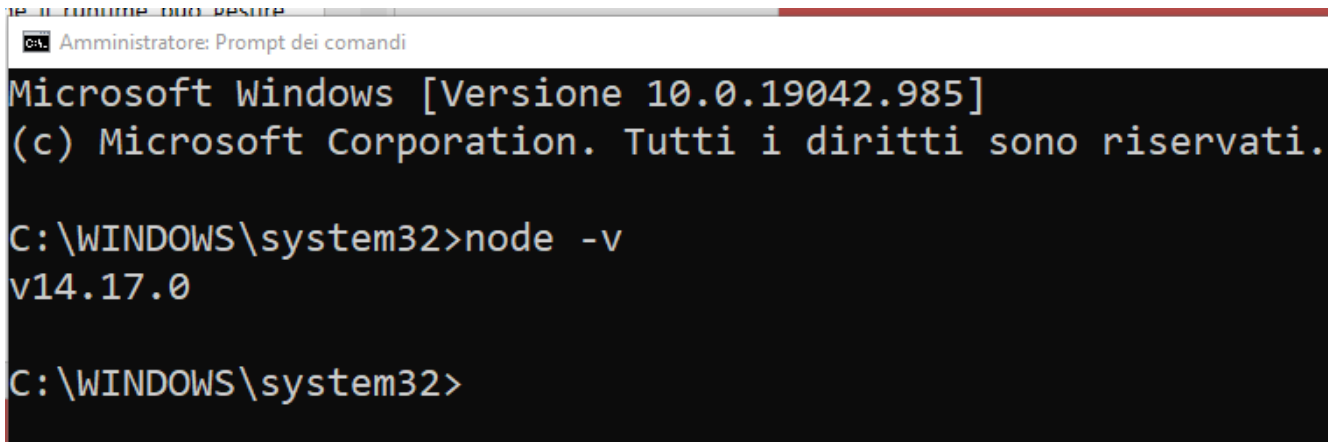
---



---

# Installazione Node.js

Node.js va utilizzato da riga di comando. Verifichiamo che l'installazione è andata a buon fine aprendo il terminale dos e digitare il comando `node v`

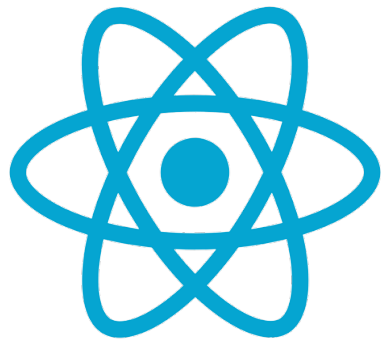
A screenshot of a Windows Command Prompt window. The title bar reads "Amministratore: Prompt dei comandi". The window content shows the following text: "Microsoft Windows [Versione 10.0.19042.985]", "(c) Microsoft Corporation. Tutti i diritti sono riservati.", "C:\WINDOWS\system32>node -v", "v14.17.0", and "C:\WINDOWS\system32>".

```
Microsoft Windows [Versione 10.0.19042.985]
(c) Microsoft Corporation. Tutti i diritti sono riservati.

C:\WINDOWS\system32>node -v
v14.17.0

C:\WINDOWS\system32>
```





---

# Il nostro primo script Node

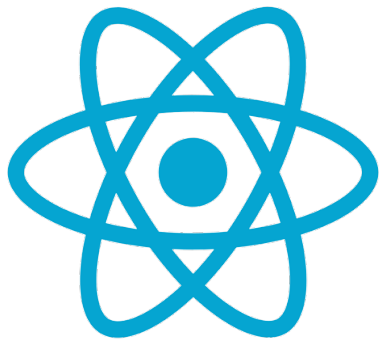
Per prima cosa apriamo il nostro terminale e digitiamo semplicemente il comando `node`. In questo modo entriamo in una modalità interattiva per poter fare dei semplici test.

Proviamo a scrivere la seguente riga di codice:

```
'node.js'.toUpperCase()
```

e poi diamo invio. Possiamo vedere il risultato a video. Per uscire dalla modalità `node` premiamo i tasti `CTRL+D`.

---



---

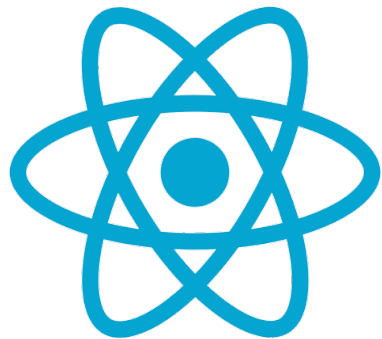
# Il nostro primo script Node

```
C:\WINDOWS\system32>node
Welcome to Node.js v14.17.0.
Type ".help" for more information.
> 'node.js'.toUpperCase()
'NODE.JS'
>

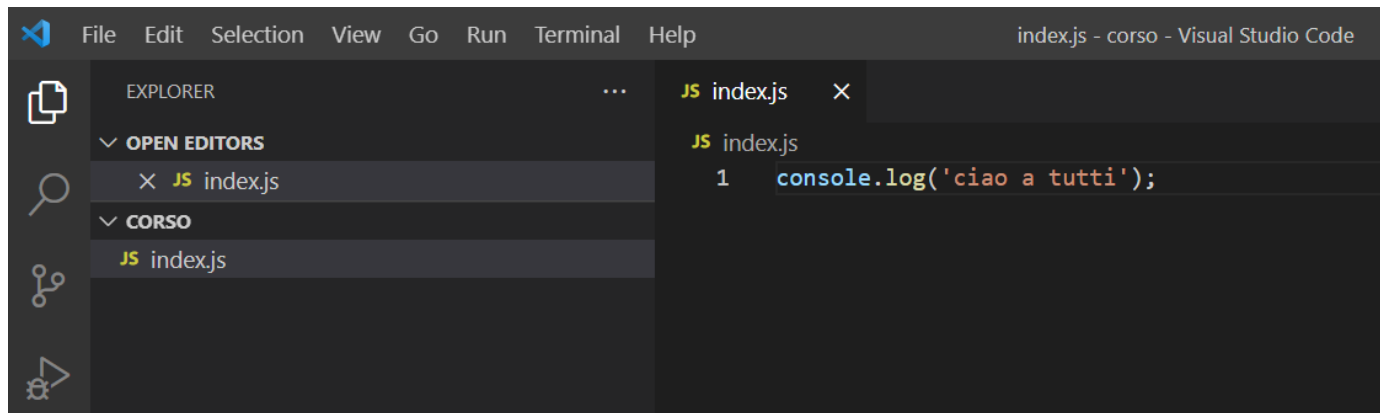
C:\WINDOWS\system32>
```

Per fare un esempio meno banale andiamo a creare una cartella sul desktop e la chiamiamo «corso» e sempre tramite la riga di comando ci andremo a posizionare all'interno di essa e creeremo un file chiamato index.js (usando un editor di testo come ad esempio Visual Studio Code)

---



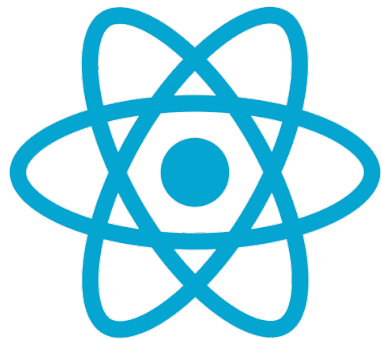
# Il nostro primo script Node



```
C:\Users\Rick\Desktop>cd corso
```

```
C:\Users\Rick\Desktop\corso>node index.js  
ciao a tutti
```

```
C:\Users\Rick\Desktop\corso>
```



---

# Il registro NPM

NPM rappresenta un repository di librerie scritte apposta per poter essere utilizzate con Node.js

Per installare un pacchetto basta un semplice:

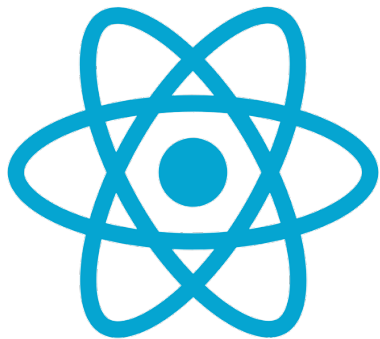
**npm install nome pacchetto**

Per vedere un esempio concreto consideriamo l'installazione del modulo socket.io. Questo modulo permette di gestire le connessioni via socket a basso livello.

Non preoccupiamoci di capire in dettaglio a cosa serve al momento lo useremo solo come esempio di procedura di installazione di un nuovo componente

**npm install bootstrap**

---



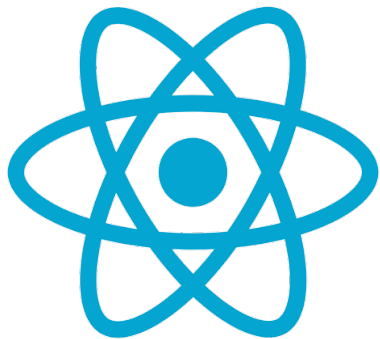
---

# Come creare una React App

Per creare la prima app è sufficiente digitare il comando `npx` che non è altro che un esecutore di pacchetti seguito dal pacchetto `create-react-app` e poi il nome dell'applicazione. Per fare un primissimo esempio creiamo sul desktop una cartella `ciaomondo` e tramite terminale digitiamo:

```
npx create-react-app prima-app
```

---



---

# Come creare una React App

Success! Created ciao at C:\Users\Rick\Desktop\ciao  
Inside that directory, you can run several commands:

`npm start`

Starts the development server.

`npm run build`

Bundles the app into static files for production.

`npm test`

Starts the test runner.

`npm run eject`

Removes this tool and copies build dependencies, configuration files and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

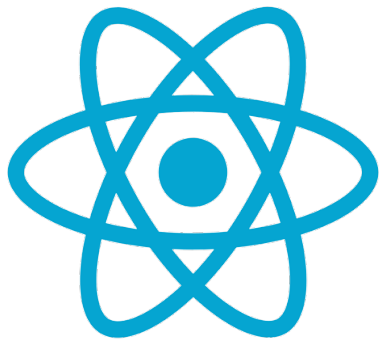
`cd ciao`

`npm start`

Happy hacking!

C:\Users\Rick\Desktop>

---



---

# Come creare una React App

Appena lanciamo il comando partirà la configurazione della nostra nuova app, ci vuole qualche secondo Una volta terminata l'installazione di tutte le librerie necessarie possiamo lanciare il nostro server locale digitando da riga di comando

`npm start`

```
Compiled successfully!
```

```
You can now view ciao in the browser.
```

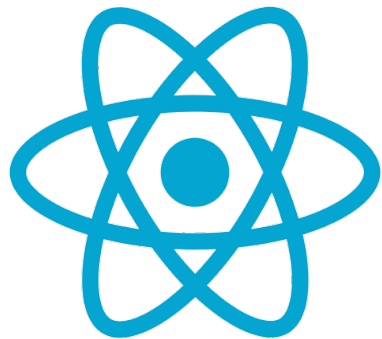
```
Local: http://localhost:3000
```

```
On Your Network: http://172.18.144.1:3000
```

```
Note that the development build is not optimized.
```

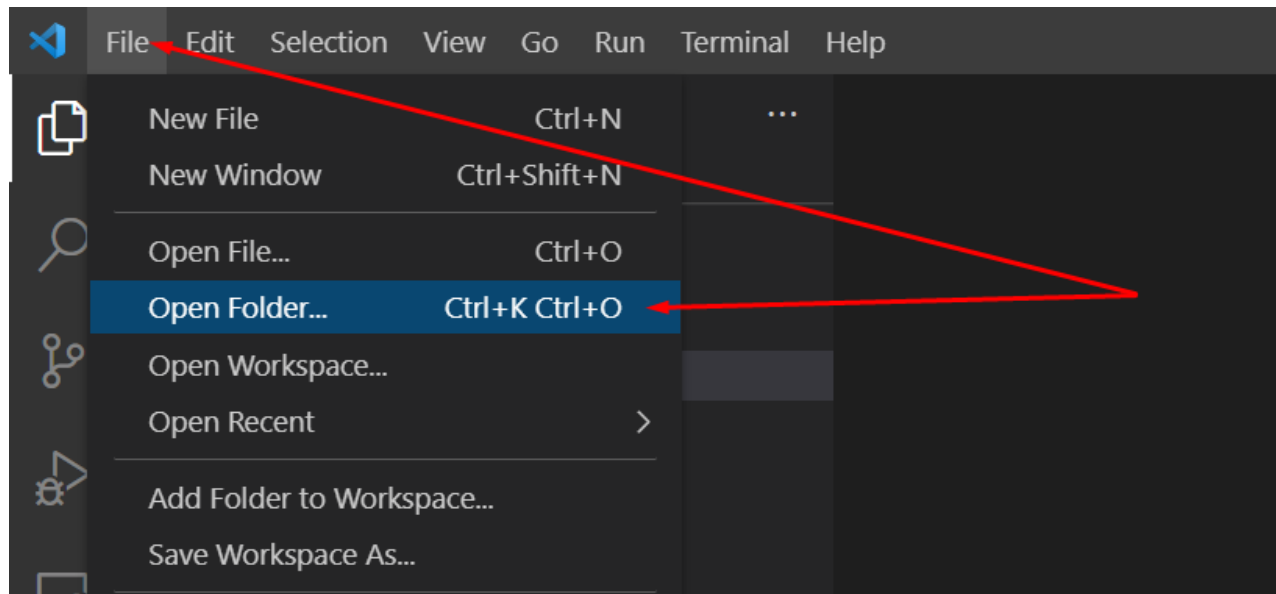
```
To create a production build, use npm run build.
```

---

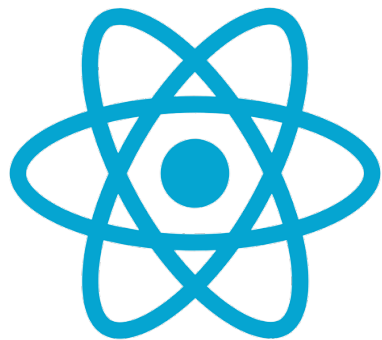


# Struttura di un progetto React

Andiamo ora ad aprire la nostra applicazione attraverso Visual Studio Code

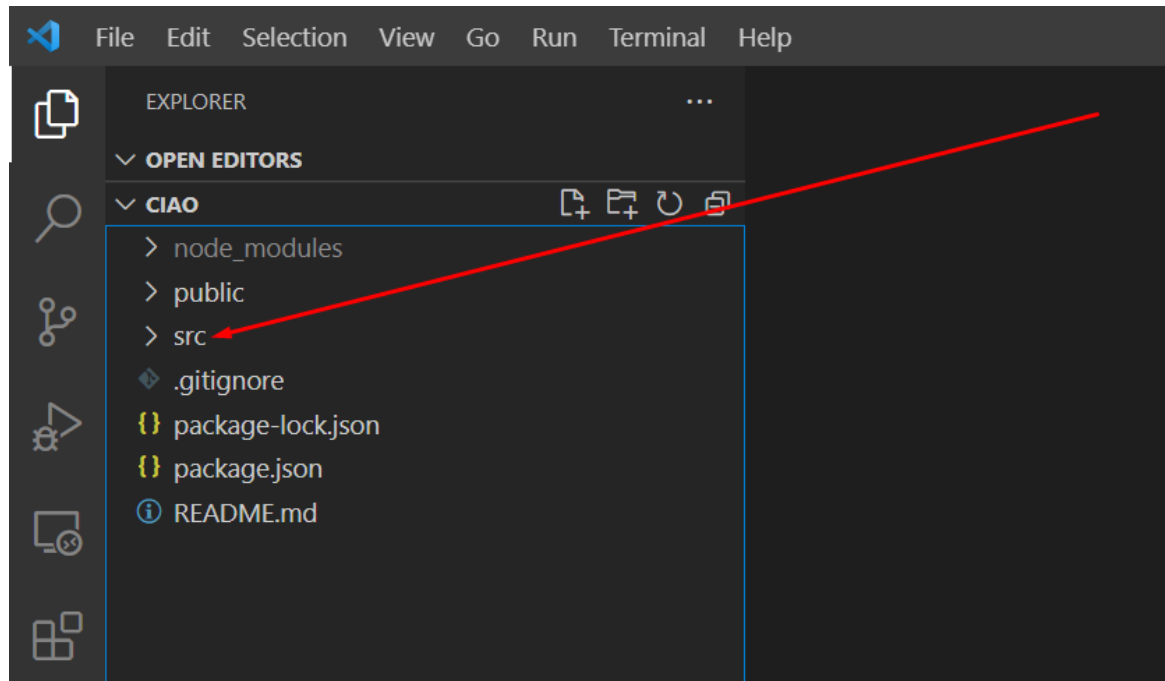


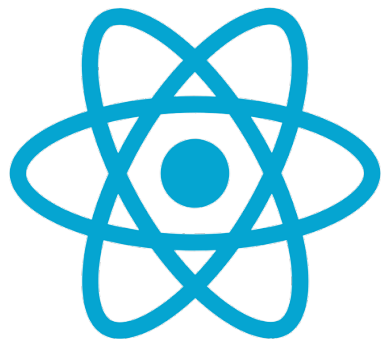




# Struttura di un progetto React

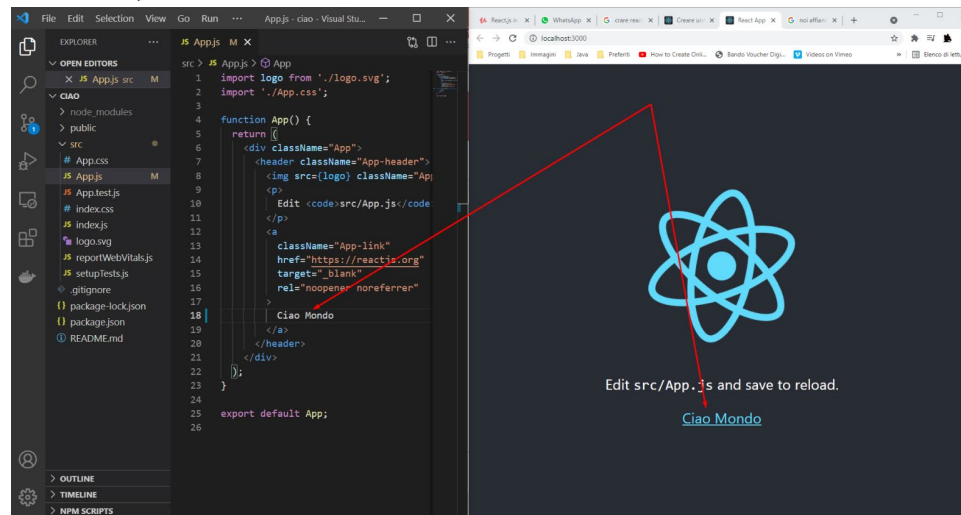
Andiamo ora ad aprire la nostra applicazione attraverso Visual Studio Code

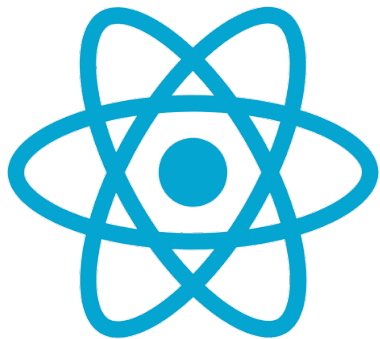




# Struttura di un progetto React

Per fare una prima prova apriamo il file App.js che troviamo all'interno della nostra cartella src e proviamo a modificare il testo all'interno del codice html ad esempio sostituiamo la scritta « Learn React » con «Ciao Mondo», salviamo e vediamo cosa succede (per vederlo affianchiamo le due finestre browser e visual studio code)

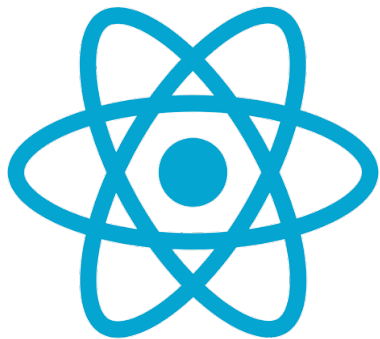




# Struttura di un progetto React

A screenshot of the Visual Studio Code editor interface. The Explorer sidebar on the left shows a project structure for a React application. The file explorer is expanded to show the 'src' directory, which contains files like App.css, App.js, App.test.js, index.css, index.js, logo.svg, reportWebVitals.js, setupTests.js, .gitignore, package-lock.json, package.json, and README.md. The package.json file is selected and its content is displayed in the main editor area. A red line is drawn across the package.json content, starting from the 'scripts' section and extending towards the top right. The package.json content is as follows:

```
{
  "name": "ciao",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.13.0",
    "@testing-library/react": "^11.2.7",
    "@testing-library/user-event": "^12.8.3",
    "react": "^17.0.2",
    "react-dom": "^17.0.2",
    "react-scripts": "4.0.3",
    "web-vitals": "^1.1.2"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  }
}
```



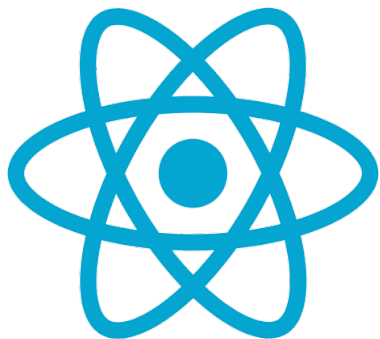
---

# Struttura di un progetto React

Questo è il primo file del nostro progetto che dobbiamo conoscere `package.json` dove troviamo la nostra configurazione e tutte le librerie che sono state installate automaticamente quando abbiamo creato il progetto

Fisicamente tutte le librerie scaricate possiamo vederle all'interno della cartella `node_modules`

---



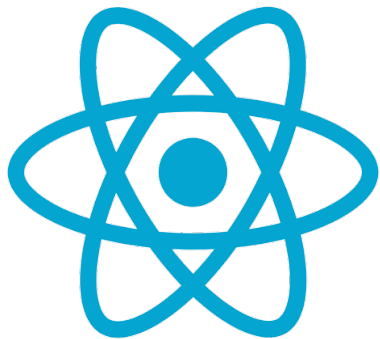
# Struttura di un progetto React

All'interno della cartella public troviamo il file index.html che è il file che effettivamente viene caricato quando facciamo partire la nostra applicazione.

Tutto quello che scriveremo all'interno del nostro progetto, quindi nella cartella src verrà visualizzato all'interno del div con id root.

A screenshot of the Visual Studio Code editor interface. The Explorer sidebar on the left shows the project structure: 'public' folder containing 'index.html', 'logo192.png', 'logo512.png', 'manifest.json', and 'robots.txt'; 'src' folder containing '.gitignore', 'package-lock.json', 'package.json', and 'README.md'. The 'index.html' file is selected and its content is displayed in the main editor. The code is a standard HTML template for a web application, including a head section with meta tags for charset, viewport, theme-color, and description, and a body section with a title and a root div. Red arrows point from the 'index.html' file in the Explorer to the corresponding line in the code editor, and from the 'div id="root"' in the code to the text in the paragraph above.

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8" />
5     <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
6     <meta name="viewport" content="width=device-width, initial-scale=1" />
7     <meta name="theme-color" content="#000000" />
8     <meta
9       name="description"
10      content="Web site created using create-react-app"
11    />
12     <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
13   </head>
14   <!--
15     manifest.json provides metadata used when your web app is installed on a
16     user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
17   -->
18   <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
19   <!--
20     Notice the use of %PUBLIC_URL% in the tags above.
21     It will be replaced with the URL of the 'public' folder during the build.
22     Only files inside the 'public' folder can be referenced from the HTML.
23
24     Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
25     work correctly both with client-side routing and a non-root public URL.
26     Learn how to configure a non-root public URL by running 'npm run build'.
27   -->
28   <title>React App</title>
29 </head>
30 <body>
31   <noscript>You need to enable JavaScript to run this app.</noscript>
32   <div id="root"></div>
```



---

# Esercizio

Creare un progetto react e creare una pagina di presentazione personalizzata sostituendo l'html presente di default nella pagina App.js

Importare bootstrap o con il semplice link html o scaricando la libreria con npm

---