Propositi:

Capire cos'è Vue.js e i suoi componenti principali

Capire cos'è React.js e i suoi componenti principali

Differenze tra angular e i precedenti elementi

Cos'è Vue?#

è un framework JS per la creazione di interfacce utente. Si basa su HTML, CSS e JSstandard e fornisce un modello di programmazione dichiarativo e basato su componenti che aiuta a sviluppare in modo efficiente interfacce utente, siano esse semplici o complesse.

Vue.js ci consente di estendere l' HTML tramite l'utilizzo di specifici attributi HTML chiamati direttive

Le direttive di Vue.js ci offrono diverse funzionalità adattabili alle nostre applicazioni HTML

Inoltre come molti altri Vue.js fornisce nativamente sia direttive integrate che direttive definite dall'utente

itconsulting
Consulenza & Formazione

Un esempio:

```
1. import { createApp } from 'vue'
2.
3. createApp({
4.   data() {
5.     return {
6.       count: 0
7.     }
8.   }
9. }).mount('#app')
10.
11. <div id="app">
12.   <button @click="count++">
13.     la data è: {{ count }}
14.   </button>
15. </div>
```

Un esempio:

L'esempio sopra mostra le due caratteristiche principali di Vue:

R*endering dichiarativo* : Vue estende l'HTML standard con una sintassi del modello che ci consente di descrivere in modo dichiarativo l'output HTML in base allo stato JavaScript.

R*eattività* : Vue traccia automaticamente le modifiche allo stato di JavaScript e aggiorna in modo efficiente il DOM quando si verificano

itconsulting
Consulenza & Formazione

Componenti a file singolo

Nella maggior parte dei progetti Vue abilitati per lo strumento di compilazione, creiamo componenti Vue utilizzando un formato di file simile a HTML chiamato Single-File Component (noto anche come *.vuefile, abbreviato come SFC).
Un Vue SFC, come suggerisce il nome, incapsula la logica del componente (JavaScript), il modello (HTML) e gli stili (CSS) in un singolo file.
Ecco l'esempio precedente, scritto in formato SFC:

```
<script>
export default {
  data() {
    return {
      count: 0
    } }   }
</script>
<template>
  <button @click="count++">Count is: {{ count }}</button>
</template>
<style scoped>
button {
  font-weight: bold;}
</style>
```

itconsulting
Consulenza & Formazione

Componenti a file singolo

Nella maggior parte dei progetti Vue abilitati per lo strumento di compilazione, creiamo componenti Vue utilizzando un formato di file simile a HTML chiamato Single-File Component (noto anche come *.vuefile, abbreviato come SFC).
Un Vue SFC, come suggerisce il nome, incapsula la logica del componente (JavaScript), il modello (HTML) e gli stili (CSS) in un singolo file.
Ecco l'esempio precedente, scritto in formato SFC:

```
<script>
export default {
  data() {
    return {
      count: 0
    } }   }
</script>
<template>
  <button @click="count++">Count is: {{ count }}</button>
</template>
<style scoped>
button {
  font-weight: bold;}
</style>
```

itconsulting
Consulenza & Formazione

Componenti a file singolo

API delle opzioni#

Con l'API delle opzioni, definiamo la logica di un componente utilizzando un oggetto di opzioni come data, methodse mounted. Le proprietà definite dalle opzioni sono esposte nelle funzioni interne, che puntano all'istanza del componente:

```
<script>
export default {
  // Properties returned from data() become reactive state
  // and will be exposed on `this`.
  data() {
    return {
      count: 0  }  },
  // Methods are functions that mutate state and trigger updates.
  // They can be bound as event listeners in templates.
  methods: {
    increment() {
      this.count++  } },
  // Lifecycle hooks are called at different stages
  // of a component's lifecycle.
  // This function will be called when the component is mounted.
  mounted() {
    console.log(`The initial count is ${this.count}.`)}}
</script>
<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

itconsulting
Consulenza & Formazione

Direttive Vue.js

Vue.js utilizza doppie parentesi graffe { {} } come placeholder di delimitazione per i dati.
Le direttive Vue.js sono attributi HTML con il prefisso v-
Nell'esempio seguente, creiamo un nuovo oggetto Vue con new Vue() .
La proprietà el: lega il nuovo oggetto Vue all'elemento HTML con id="app" .

```
1. <!DOCTYPE html>
2. <html>
3. <script src="https://cdn.jsdelivr.net/npm/vue@2.6.14/dist/vue.js"></script>
4. <body>
5. <div id="app">
6.   <h1>{{ message }}</h1>
7. </div>
8. <script>
9. var myObject = new Vue({
10.   el: '#app',
11.   data: {message: 'Hello Vue!'}
12. })
13. </script>
14. </body>  </html>
```

itconsulting
Consulenza & Formazione

Associazione in Vue.js generica

Quando un oggetto Vue è associato a un elemento HTML, l'elemento HTML cambierà quando l'oggetto Vue cambia in maniera diretta:

```
<div id="app">
{{ message }}
</div>

<script>
var myObject = new Vue({
   el: '#app',
   data: {message: 'Hello Vue!'}
})

function myFunction() {
   myObject.message = "John Doe";
}
</script>
```

itconsulting
Consulenza & Formazione

Associazione in Vue.js

Quando un oggetto Vue è associato a un elemento HTML, l'elemento HTML cambierà quando l'oggetto Vue cambia in maniera diretta:

```
<!DOCTYPE html>
<html>
<script src="https://cdn.jsdelivr.net/npm/vue@2.6.14/dist/vue.js"></script>
<body>
<h2>Vue.js</h2>
<div id="app">
  {{ message }}
</div>
<p>
<button onclick="myFunction()">SONO UN BOTTONE</button>
</p>
<script>
var myObject = new Vue({
  el: '#app',
  data: {message: 'Hello Vue!'}  })
function myFunction() {
    myObject.message = "John Doe";}
</script>
</body> </html>
```

itconsulting
Consulenza & Formazione

Associazione a due vie

La v-model la direttiva lega il valore degli elementi HTML ai dati dell'applicazione.

Questo è chiamato binding a due vie:

```
<div id="app">
  <p>{{ message }}</p>
  <p><input v-model="message"></p>
</div>

<script>
var myObject = new Vue({
  el: '#app',
  data: {message: 'Hello Vue!'}
})
</script>

```

itconsulting
Consulenza & Formazione

Binding in Vue.js

Utilizzo della direttiva v-for serve per associare un array di oggetti Vue a un "array" di elementi HTML:

```
<!DOCTYPE html>
<html>
<script src="https://cdn.jsdelivr.net/npm/vue@2.6.14/dist/vue.js"></script>
<body>
<h2>Vue.js</h2>
<div id="app">
  <ul>
    <li v-for="x in todos">
      {{ x.text }}
    </li>
  </ul>
</div>
<script>
myObject = new Vue({
  el: '#app',
  data: {
    todos: [
      { text: 'Learn JavaScript' },
      { text: 'Learn Vue.js' },
      { text: 'Build Something Awesome' } ] } })
</script>  </body>  </html>
```

itconsulting
Consulenza & Formazione

API di composizione

Con l'API Composition, definiamo la logica di un componente utilizzando le funzioni API importate. Negli SFC, l'API di composizione viene in genere utilizzata con <script setup>. L' setupattributo è un suggerimento che consente a Vue di eseguire trasformazioni in fase di compilazione che ci consentono di utilizzare l'API Composition con meno boilerplate. Ad esempio, le importazioni e le variabili/funzioni di primo livello dichiarate in <script setup>sono direttamente utilizzabili nel modello. Ecco lo stesso componente, con lo stesso identico modello, ma utilizzando l'API di composizione e <script setup>invece:

```
<script setup>
import { ref, onMounted } from 'vue'
// reactive state
const count = ref(0)
// functions that mutate state and trigger updates
function increment() {
  count.value++ }
// lifecycle hooks
onMounted(() => {
  console.log(`The initial count is ${count.value}.`) })
</script>
<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

Cenni tecnici

Entrambi gli stili API sono pienamente in grado di coprire casi d'uso comuni. Sono interfacce diverse alimentate dallo stesso identico sistema sottostante. In effetti, l'API delle opzioni è implementata sopra l'API della composizione! I concetti e le conoscenze fondamentali su Vue sono condivisi tra i due stili.

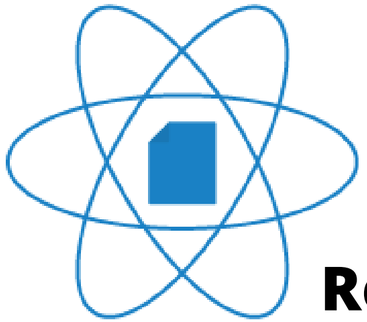
L'API delle opzioni è incentrata sul concetto di "istanza del componente" (thiscome mostrato nell'esempio), che in genere si allinea meglio con un modello mentale basato sulla classe per gli utenti che provengono da un background linguistico OOP. È anche più adatto ai principianti astraendo i dettagli di reattività e applicando l'organizzazione del codice tramite gruppi di opzioni.

Ogni applicazione Vue inizia creando una nuova istanza dell'applicazione con la createAppfunzione:

```
import { createApp } from 'vue'

import App from './App.vue'

const app = createApp(App)
```

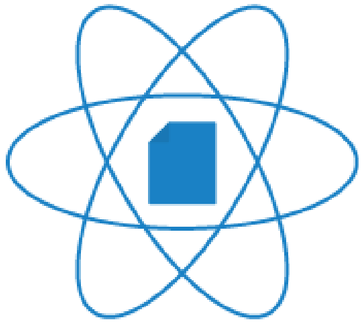


React è una libreria JavaScript per la creazione di interfacce utente.

React viene utilizzato per creare applicazioni a pagina singola anche dette single page application.

React ci consente di creare i componenti dell'interfaccia utente e di renderli generalizzati ovvero riutilizzabili.

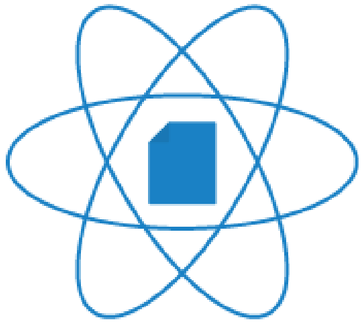
A cosa serve React?



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

itconsulting
Consulenza & Formazione

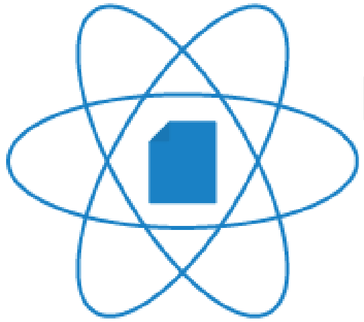


Per imparare e testare React, dobbiamo configurare un ambiente React sul computer.
Questo esempio utilizza il formato create-react-app.
Lo create-react-app è un modo ufficialmente supportato per creare applicazioni React.
Node.js è necessario per utilizzare create-react-app.

https://nodejs.org/it/download/

Alla fine del processo controlliamo la versione tramite

node -v



Apri il tuo terminale nella directory in cui desideri creare la tua applicazione.
Esegui questo comando per creare un'applicazione React denominata my-react-app

```
npx create-react-app my-react-app
```

create-react-appimposterà tutto il necessario per eseguire un'applicazione React.

Nota: se in precedenza hai installato create-react-appa livello globale, ti consigliamo di disinstallare il pacchetto per assicurarti che npx utilizzi sempre la versione più recente di create-react-app. Per disinstallare: npm uninstall -g create-react-app.

Cenni iniziali

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

Questa strana sintassi con i tag non è né una stringa né HTML.
È chiamata JSX, ed è un'estensione della sintassi JavaScript. Ti consiglio di utilizzarla con React per descrivere l'aspetto che dovrebbe avere la UI (User Interface, o interfaccia utente).

JSX ti potrebbe ricordare un linguaggio di template, ma usufruisce di tutta la potenza del JavaScript.
JSX produce "elementi React".

Studieremo il modo in cui gli elementi vengono renderizzati nel DOM nella prossima sezione.

Qui sotto troverai le nozioni fondamentali di JSX, sufficienti per iniziare ad utilizzarlo.

Cenni iniziali

Perché JSX? React riconosce il fatto che la logica di renderizzazione è per sua stessa natura accoppiata con le altre logiche che governano la UI: la gestione degli eventi, il cambiamento dello stato nel tempo, la preparazione dei dati per la visualizzazione.

Invece di separare artificialmente le tecnologie inserendo il codice di markup e la logica in file separati, React separa le responsabilità utilizzando unità debolmente accoppiate chiamate “componenti” che contengono entrambi.

React non obbliga ad utilizzare JSX, ma la maggior parte delle persone lo trovano utile come aiuto visuale quando lavorano con la UI all’interno del codice JavaScript. Inoltre, JSX consente a React di mostrare messaggi di errore e di avvertimento più efficaci.

itconsulting
Consulenza & Formazione

Cenni iniziali - Incorporare espressioni in JSX

Nell'esempio in basso, dichiariamo una variabile chiamata name e poi la utilizziamo all'interno di JSX racchiudendola in parentesi graffe:

```
const name = 'Giuseppe Verdi';
const element = <h1>Hello, {name}</h1>;
```

Puoi inserire qualsiasi espressione JavaScript all'interno delle parentesi graffe in JSX. includiamo il risultato della chiamata ad una funzione JavaScript, formatName(user), in un elemento <h1>.

```
1. function formatName(user) {
2.   return user.firstName + ' ' + user.lastName;}
3. const user = {
4.   firstName: 'Giuseppe',
5.   lastName: 'Verdi' };
6. const element = (
7.   <h1>
8.     Hello, {formatName(user)}!
9. </h1>);
```

Cenni iniziali - JSX come Espressione

Dopo la compilazione, le espressioni JSX diventano normali chiamate a funzioni JavaScript che producono oggetti JavaScript.
Questo significa che puoi utilizzare JSX all'interno di istruzioni if e cicli for, assegnarlo a variabili, utilizzarlo come argomento di una funzione e restituirlo come risultato di una funzione:

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

Specificare gli Attributi con JSX

Puoi utilizzare le virgolette per valorizzare gli attributi con una stringa:

```
const element = <a href="https://www.reactjs.org"> link </a>;
```

Puoi anche utilizzare le parentesi graffe per includere un'espressione JavaScript in un attributo:

```
const element = <img src={user.avatarUrl}></img>;
```

Non aggiungere le virgolette attorno alle parentesi graffe quando includi un'espressione JavaScript in un attributo. Dovresti utilizzare o le virgolette (per le stringhe) o le parentesi graffe (per le espressioni), ma mai entrambe nello stesso attributo.

itconsulting
Consulenza & Formazione

Esempio React

Il nostro strumento "Show React" semplifica la dimostrazione di React.
Mostra sia il codice che il risultato.

```
1. import React from "react";
2. import ReactDOM from "react-dom/client";
3.
4. function Hello(props) {
5.   return <h1>Hello World!</h1>;
6. }
7.
8. const root = ReactDOM.createRoot(document.getElementById("root"));
9. root.render(<Hello />);
```

Renderizzare Elementi

Gli elementi sono i più piccoli tra i vari mattoni costituenti apps scritte in React.
Un elemento descrive cosa vuoi vedere sullo schermo:

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

Contrariamente agli elementi DOM del browser, gli elementi React sono oggetti semplici e per questo più veloci da creare. Il DOM di React tiene cura di aggiornare il DOM del browser per essere consistente con gli elementi React.

Renderizzare un Elemento nel DOM

Supponiamo di avere un da qualche parte nel tuo file HTML:

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

Lo chiameremo nodo DOM “radice” (o root) in quanto ogni cosa al suo interno verrà gestita dal DOM di React.Applicazioni costruite solo con React di solito hanno un solo nodo DOM radice. Se stai integrando React all’interno di apps esistenti, potresti avere più elementi DOM radice isolati, dipende dai casi.
Per renderizzare un elemento React, passa l’elemento DOM a ReactDOM.createRoot(), successivamente passa l’elemento React a root.render():

```
const root = ReactDOM.createRoot(
  document.getElementById('root'));const element = <h1>Hello, world</h1>;
root.render(element);
```

Aggiornare un Elemento Renderizzato

Gli elementi React sono immutabili. Una volta creato un elemento, non puoi cambiare i suoi figli o attributi. Un elemento è come un singolo fotogramma di un film: rappresenta la UI (interfaccia utente) ad un certo punto nel tempo.
Con la conoscenza che abbiamo fino a questo punto, l'unico modo per aggiornare l'UI è quello di creare un nuovo elemento e di passarlo a root.render().
Prendiamo in considerazione il prossimo esempio, nel quale abbiamo un orologio:

```
1. const root = ReactDOM.createRoot(
2.   document.getElementById('root'));
3. function tick() {
4.   const element = (
5.     <div>
6.       <h1>Hello, world!</h1>
7.       <h2>It is {new Date().toLocaleTimeString()}.</h2>
8.     </div>);
9.   root.render(element);}
10. setInterval(tick, 1000);
```

Componenti e Props

I Componenti ti permettono di suddividere la UI (User Interface, o interfaccia utente) in parti indipendenti, riutilizzabili e di pensare ad ognuna di esse in modo isolato. Questa pagina offre una introduzione al concetto dei componenti. Puoi trovare invece informazioni dettagliate nella API di riferimento dei componenti.

Concettualmente, i componenti sono come funzioni JavaScript: accettano in input dati arbitrari (sotto il nome di "props") e ritornano elementi React che descrivono cosa dovrebbe apparire sullo schermo.

Componenti e props

Il modo più semplice di definire un componente è quello di scrivere una funzione JavaScript:

```
function Ciao(props) {
  return <h1>Ciao, {props.nome}</h1>;
}
```

Questa funzione è un componente React valido in quanto accetta un oggetto parametro contenente dati sotto forma di una singola “props” (che prende il nome da “properties” in inglese, ossia “proprietà”) che è un oggetto parametro avente dati al suo interno e ritorna un elemento React. Chiameremo questo tipo di componenti “componenti funzione” perché sono letteralmente funzioni JavaScript.

Puoi anche usare una classe ES6 per definire un componente:

```
class Ciao extends React.Component {
  render() {
    return <h1>Ciao, {this.props.nome}</h1>;
 } }
```