REACT.JS

Sviluppare applicazioni web

Ing. Massimo Nannini

Mi presento

Formazione:

- Laurea in ingegneria Elettronica ind. Automazione
- Master in costruzione di macchine automatiche

Attività lavorativa:

- Consulente informatico e project manager
- Formatore
- Sviluppatore software
- Software architect

Conosciamoci ….

Di cosa parleremo ?

- Modulo 1
  - Cos’e React
  - Perchè utilizzare React
  - Vantaggi e Limitazioni
  - Installazione e configurazione ambiente di sviluppo ReactJS
  - Utilizzo di NPM e Package.json file
  - Yarn
  - Text editor e plugins
  - create-react-app
  - Creazione di una applicazione ReactJS
  - JSX
  - Render elements
  - Component
  - Style & CSS

Di cosa parleremo ?

- Modulo 1
  - React Props
  - Props Validation con Data Types
  - Component State
  - Conditional rendering
  - Ciclo di vita
  - “ref”
  - Liste

Di cosa parleremo ?

- Modulo 2
  - React Native
  - Components
  - Styles
  - Architecture
  - Data
  - Set up emulator (IOS – Android)
  - Debug on device

Introduzione

Il web oggi è diverso, come è diverso il modo in cui si programma per il web.

A fronte delle difficoltà che si incontrano nel trattare il codice imperativo (di ardua manutenzione) prodotto per esempio da **jQuery**, occorre trovare nuovi modi per gestire la complessità delle moderne interfacce utente.

Introduzione

Serve dunque una nuova libreria che aiuti a realizzare applicazioni front-end:

- dichiarative
- modulari
- veloci
- scalari

Utilizzando JavaScript

React.js

React.js è una libreria JavaScript

sviluppata da Facebook che introduce interessanti idee interessanti su come:

- lavorare con il DOM,
- organizzare il flusso dei dati dell'applicazione
- pensare gli elementi dell'interfaccia come singoli componenti.

React.js

React è una **libreria** JavaScript **dichiarativa** basata su **componenti** utilizzata per la creazione di interfacce utente

React.js per cosa si usa ?

- React serve a realizzare UI
- React è usata per realizzare single page application
- React permette di creare componenti UI riusabili

Single Page Application

Chi programma il web lato server dalle origini, sa che è necessario scaricare ogni pagina web (dal server al browser) per poterla utilizzare.

Ogni eventuale aggiornamento ai contenuti mostrati sarebbe possibile solo al prezzo di ricaricare la pagina, il tutto con grande fastidio per l'utente e accurato lavoro di coordinamento dello sviluppatore.

Single Page Application - Ajax

La tecnologia **Ajax** ha cambiato lo scenario permettendo al codice JavaScript di aggiornare porzioni della pagina web in ***background*** senza richiederne un ricaricamento integrale da server.

Single Page Application

L’estremizzazione di questa pratica, con il favore delle librerie che l’hanno reso sempre più agevole (pensiamo a jQuery e tutti i suoi simili e derivati), ha portato a delle applicazioni web che non hanno bisogno di ricaricarsi mai: le **Single Page Application**, spesso dette SPA.

Single Page Application

In pratica, la pagina viene scaricata nel browser non appena invocata e lì continua a dialogare con servizi Web o a funzionare autonomamente, godendo dei benefici del web con la comodità di un'applicazione desktop.

Le SPA sono ormai la struttura più comune per le web app e la loro natura richiede un'architettura interna articolata, propria più di un'applicazione completa che di una semplice interfaccia per interagire con qualche altro servizio.

SPA – funzionamento

Una SPA è composta da una pagina HTML che scarica i file JavaScript necessari all'applicazione.

La pagina HTML contiene un tag (tipicamente un div) che diventa il contenitore dell'intera applicazione.

A questo punto entrano in azione il codice JavaScript che scarica dal server un frammento di HTML corrispondente alla home page della nostra SPA.

Il frammento di HTML viene inserito nel div che agisce come contenitore quindi l'utente vede la sua home page (ovviamente il frammento di HTML scaricato non è un'intera pagina ma solo la parte visuale dell'HTML).

SPA - funzionamento

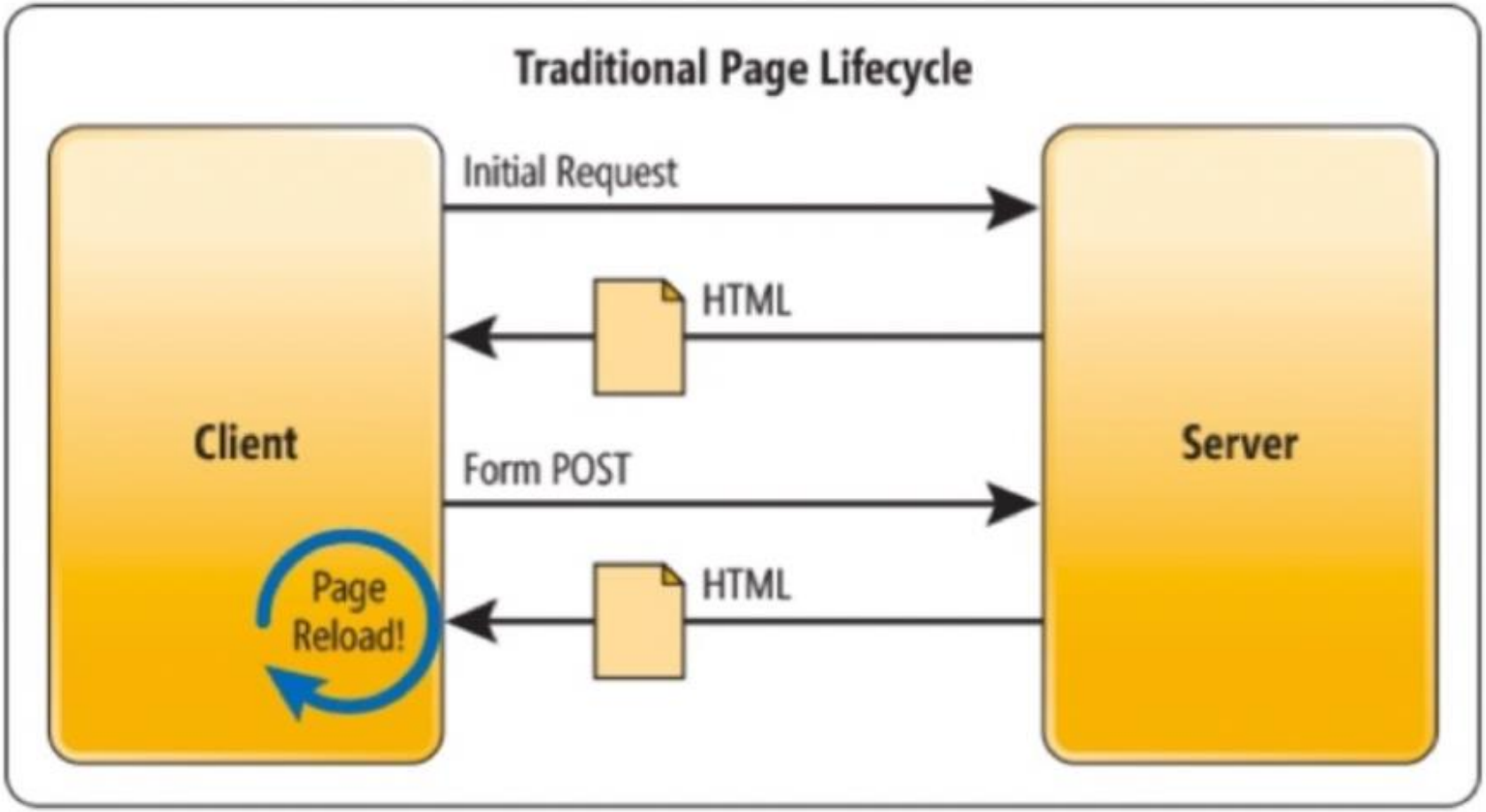
Nel momento in cui l'utente naviga verso un'altra pagina, la navigazione viene intercettata dal JavaScript che si occupa di recuperare il frammento di HTML e il codice JavaScript della pagina a cui l'utente sta navigando.

Successivamente, il codice HTML contenuto nel frammento scaricato viene inserito all'interno del div contenitore rimpiazzando quello della home page.

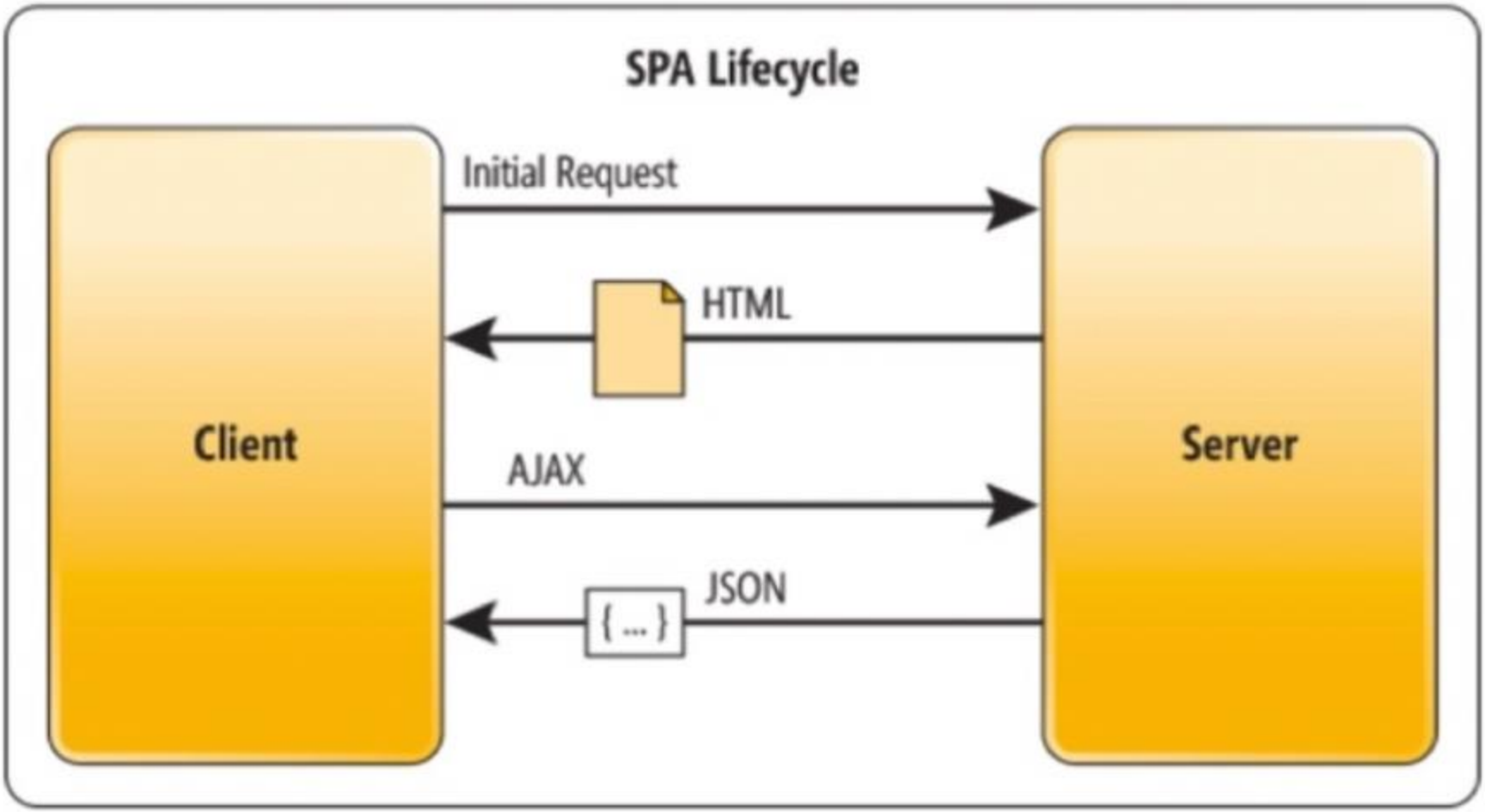
SPA - funzionamento

Essendo questa navigazione interamente gestita da codice JavaScript, la pagina sul browser **non viene mai completamente ricaricata** ed è questo il motivo per cui questo tipo di applicazione viene definita **Single Page Application**.

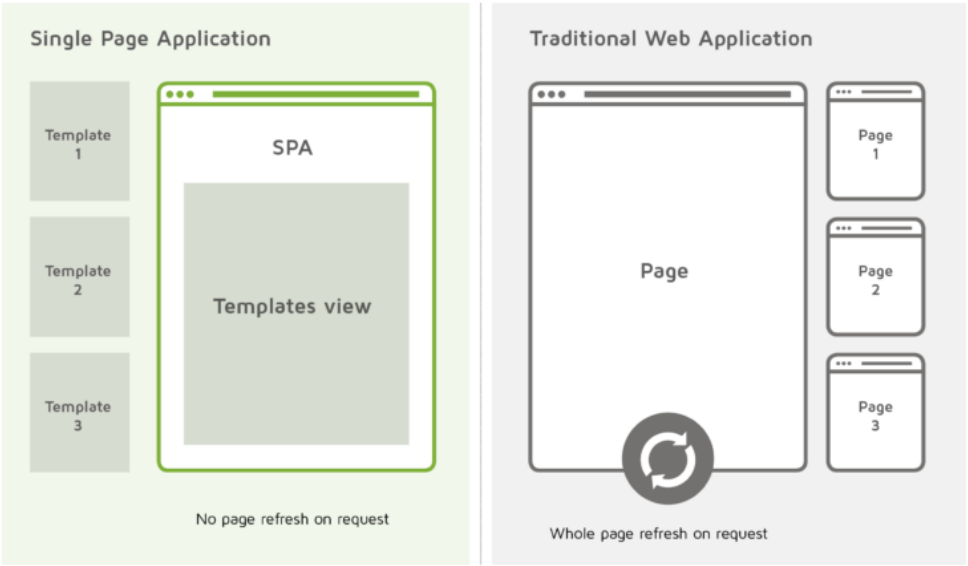
SPA - funzionamento

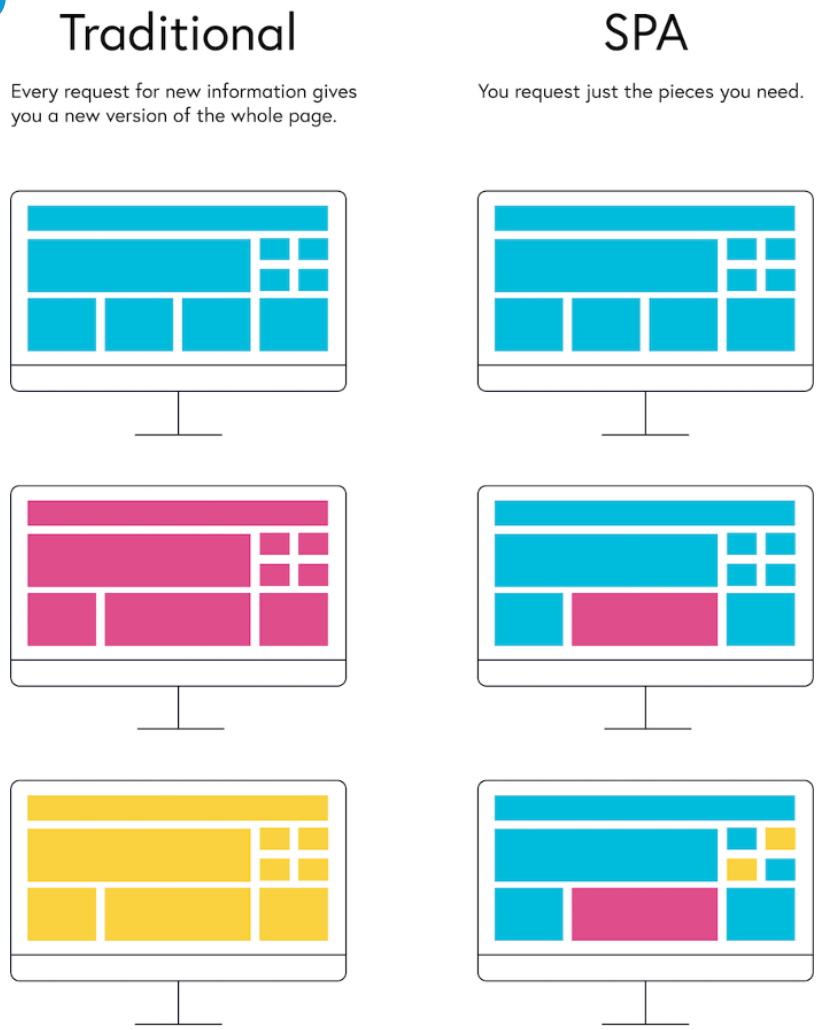


SPA - funzionamento



SPA - funzionamento





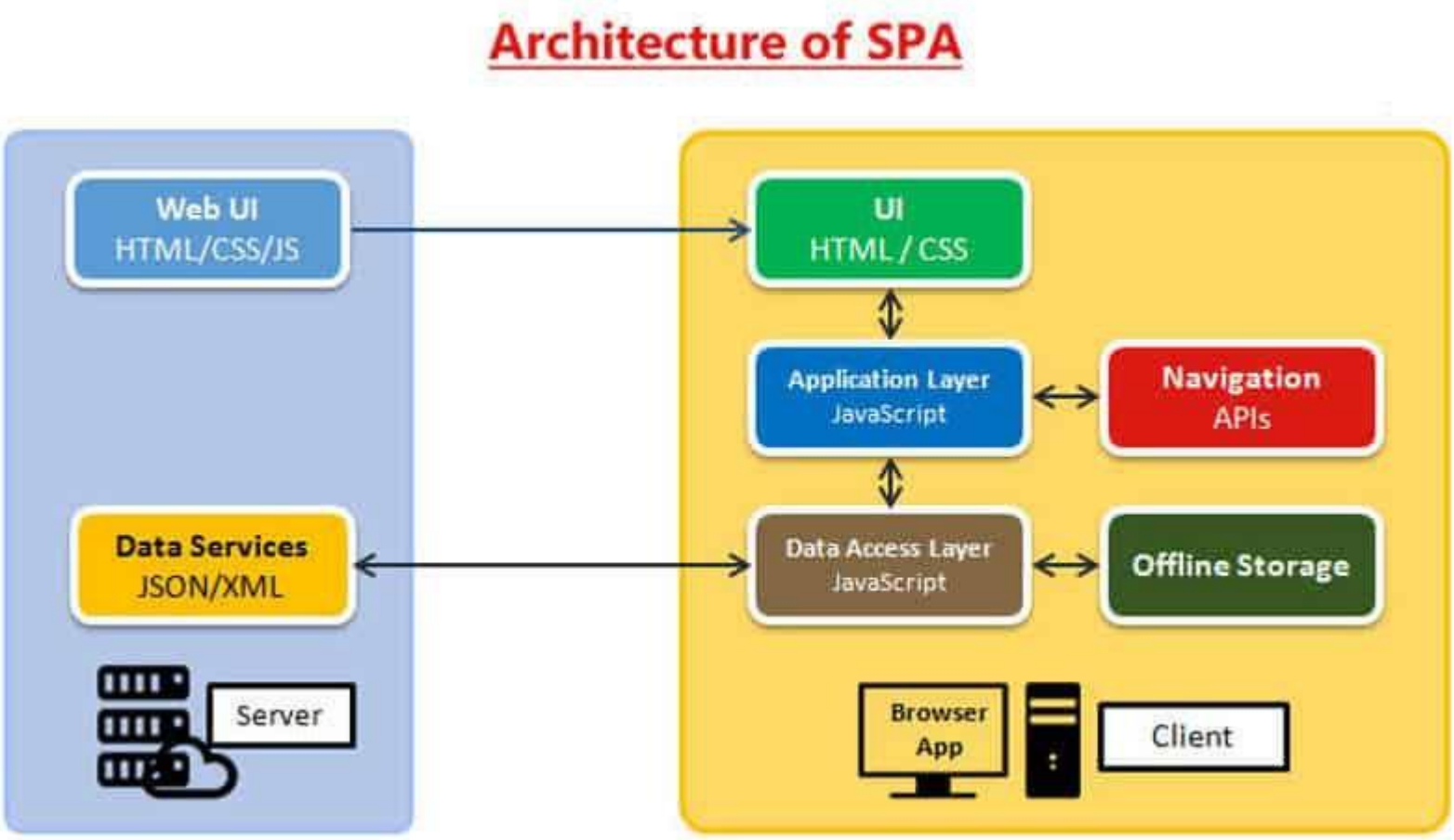
SPA - funzionamento

Come si può capire il codice da scrivere per creare una SPA da zero è notevole in quanto ci dobbiamo occupare della gestione della navigazione, del cambiare il contenuto del div dopo ogni navigazione, dello scaricare i file HTML e JavaScript necessari e molto altro ancora.

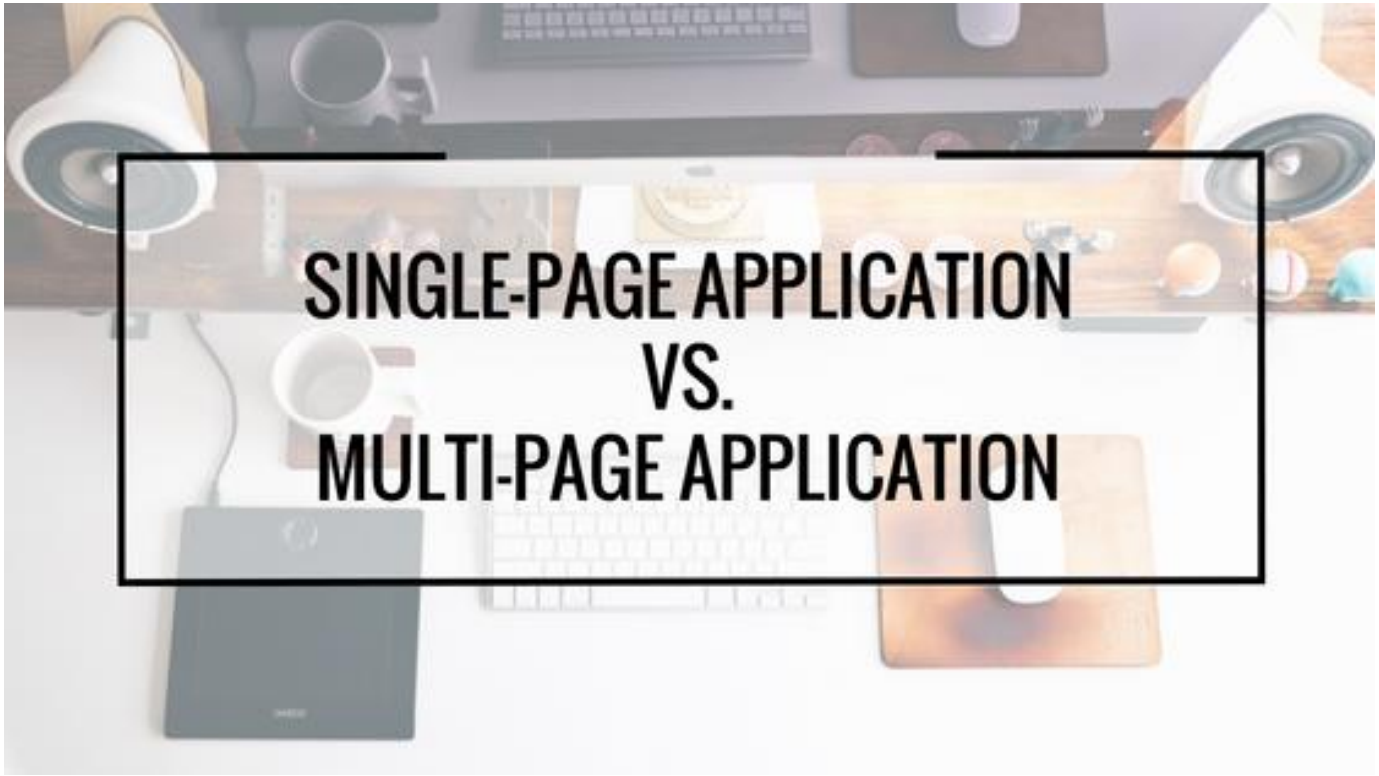
Le librerie/framework si occupano di questi aspetti lasciando al programmatore solo la gestione degli aspetti business propri dell'applicazione.

SPA - funzionamento

SPA - architettura



SPA – Vantaggi e svantaggi



SPA – Vantaggi e svantaggi

I più grossi vantaggi consistono nell'elevata usabilità e nelle elevate performance dell'applicazione.

Poiché la maggior parte del codice gira sul client e la pagina non viene mai ricaricata grazie al fatto che tutte le richieste al server sono AJAX (ovviamente eccetto la prima), l'utente ha la sensazione di lavorare quasi con un'applicazione Windows.

Inoltre, sempre poiché il codice gira in gran parte sul client, il server viene sollevato da molti compiti e quindi può evadere più richieste.

SPA – Vantaggi e svantaggi

Il grosso svantaggio delle SPA consiste nell'enorme quantità di codice JavaScript da scrivere per mantenerle.

Sebbene con un po' di organizzazione si possa scrivere codice JavaScript ordinato e manutenibile, lo sviluppo in JavaScript è complicato e soggetto a errori.

Per chi non ha buone conoscenze di JavaScript, la curva di apprendimento per lo sviluppo di una SPA può essere molto elevata.

Evoluzione della programmazione

Dal punto di vista storico, l'architettura e lo sviluppo del software sono passati dalla programmazione strutturata imperativa per poi approdare all'attuale modello OOP (Object-Oriented Programming).

Linguaggi quali Java, C#, C++, ecc, sono stati realizzati in conformità con al paradigma OOP.

Esiste tuttavia un «ribelle» che ha abbracciato la programmazione funzionale e ha seguito un proprio percorso evolutivo, si tratta di JavaScript.

React.js è una manifestazione di questo modo di pensare:

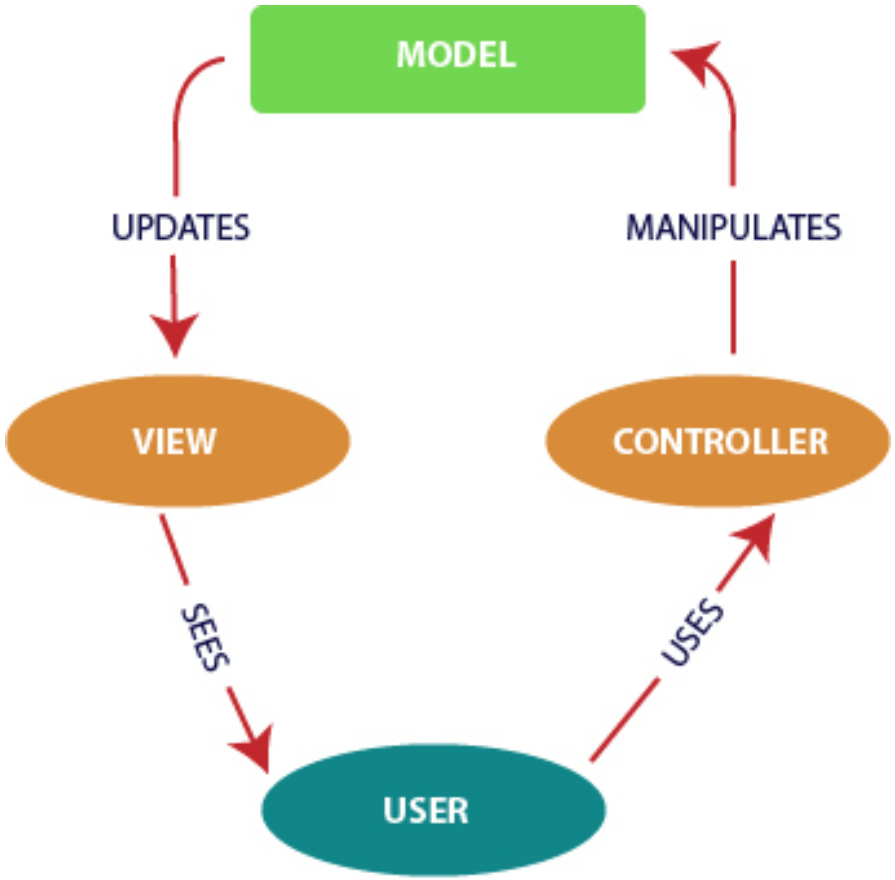
React.js filosofia

L’interfaccia utente è una funzione dello stato

React.js è dunque una libreria che utilizza uno stile dichiarativo di programmazione per descrivere lo stato dell’interfaccia utente.

Rappresenta la **V** dell’architettura MVC

MVC

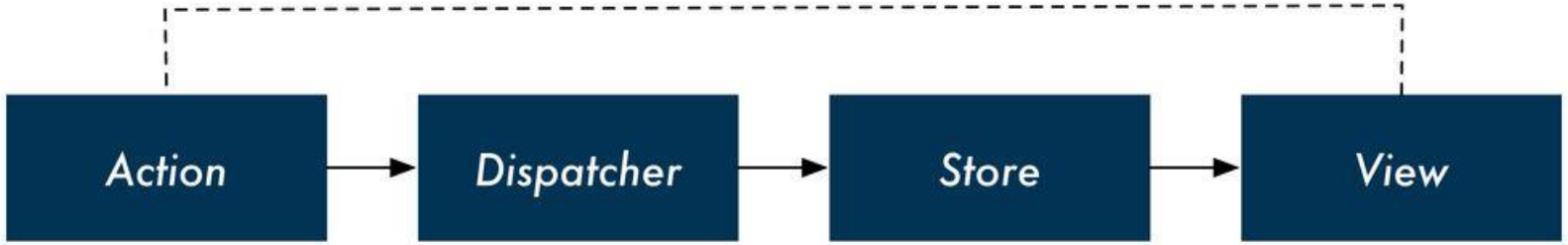


MVC Architecture

FLUX - REDUX

Per ottenere il modello MVC utilizzando React.js lo si utilizza in combinazione con pattern **FLUX** e la libreria **REDUX**.

FLUX



React.js Installazione o Setup

React.js è una libreria di JavaScript contenuta in un singolo file

react-<version>.js

che può essere incluso in ogni pagina HTML.
Comunemente viene anche installata la libreria

react-dom-<version>.js

insieme alla libreria principale

Basic Inclusion in HTML file

```
<!DOCTYPE html>
<html>
<head></head>
   <body>
      <script type="text/javascript"
      src="/path/to/react.js"></script>
      <script type="text/javascript" src="/path/to/react-
      dom.js"></script>
      <script type="text/javascript">
            // Use react JavaScript code here or in a
            separate file
      </script>
   </body>
</html>
```

JSX cosa è ?

JSX è una estensione creata da Facebook per aggiungere la sintassi XML a JavaScript

Come si usa ?

Per potere utilizzare la sintassi JSX all'interno degli script JavaScript è necessario includere libreria Babel:

```
<script
      src="https://npmcdn.com/babel-core@5.8.38/browser.min.js">
</script>
```

e cambiare:

```
<script type="text/javascript">
```

in

```
<script type="text/babel">
```

Come si usa ?

```
<!DOCTYPE html>
<html>
<head></head>
   <body>
      <script type="text/javascript"
      src="/path/to/react.js"></script>
      <script type="text/javascript" src="/path/to/react-
      dom.js"></script>
      <script src="https://npmcdn.com/babel-
      core@5.8.38/browser.min.js"></script>
      <script type="text/babel">
            // Use react JSX code here or in a separate
      file
      </script>
   </body>
</html>
```

Hello world – with no JSX

```
<body>
   <div id="example"></div>
   <script type="text/javascript">
      // create a React element rElement
      var rElement = React.createElement('h1', null,
      'Hello, world!');
      // dElement is a DOM container
      var dElement = document.getElementById('example');
      // render the React element in the DOM container
      ReactDOM.render(rElement, dElement);
   </script>
</body>
```

Hello world – with JSX

```
<body>
   <div id="example"></div>
   <script type="text/babel">
      // create a React element rElement using JSX
      var rElement = <h1>Hello, world!</h1>;
      // dElement is a DOM container
      var dElement = document.getElementById('example');
      // render the React element in the DOM container
      ReactDOM.render(rElement, dElement);
   </script>
</body>
```

Babel

Babel è un **transpiler** JavaScript.

Babel converte il codice ES6 in codice Es5.

Questa conversione viene appunto chiamata **transpiling**.

In questo modo si possono utilizzare le caratteristiche di ES6 convertite in ES5 cosicché possiamo essere sicuri che il nostro codice funzioni correttamente anche nei browsers che supportano solo ES5.

Babel

Un’altra caratteristica importante è che Babel «capisce» JSX.

Babel converte il codice JSX in ES5 JS così il browser è in grado di interpretarlo e di eseguirlo.

E’ necessario solamente indicare al browser che si intende usare Babel per eseguire il codice JavaScript.

```
<script type="text/babel">
```

Prepocessore JSX

Questo approccio va bene per l'apprendimento e la creazione di demo semplici.

Tuttavia, rallenta il sito Web e non è dunque adatto alla produzione.

Per questi motivi è meglio utilizzare il **preprocessore** JSX che richiede esclusivamente la presenza sul computer di **Node.js**

DOM

Per prima cosa, **DOM** sta per "**D**ocument **O**bject **M**odel".

Il DOM in parole semplici rappresenta l'interfaccia utente della applicazione.

Ogni volta che si verifica una modifica nello stato dell'interfaccia utente dell'applicazione, il DOM viene aggiornato per rappresentare tale modifica.

La frequente manipolazione del DOM influisce sulle prestazioni dell’applicazione rallentandola.

Che cosa rende la manipolazione lenta ?

Il DOM è rappresentato come una struttura di dati ad albero, per questo motivo, le modifiche e gli aggiornamenti al DOM sono rapidi.

Ma dopo la modifica, dell'elemento aggiornato e i relativi figli devono essere sottoposti ad un nuovo rendering per aggiornare l'interfaccia utente dell'applicazione.

Il rendering è ciò che la rallenta.

Pertanto, maggiore è il numero di componenti dell'interfaccia utente, più costosi sono gli aggiornamenti del DOM.

Virtual DOM

Il DOM virtuale è solo una rappresentazione virtuale del DOM.

Ogni volta che cambia lo stato della nostra applicazione, viene aggiornato il DOM virtuale anziché il DOM reale.

Dunque, il DOM virtuale fa la stessa cosa del vero DOM, sembra un doppio lavoro?

Come può essere più veloce del semplice aggiornamento del vero DOM?

Perché il Virtual DOM è più veloce?

Quando vengono aggiunti nuovi elementi o modificati elementi dell'interfaccia utente, viene creato un DOM virtuale, rappresentato come un albero.

Ogni elemento è un nodo di questo albero.

Se lo stato di uno di questi elementi cambia, viene creato un nuovo albero DOM virtuale.

Questo albero viene quindi confrontato con il precedente albero DOM virtuale.

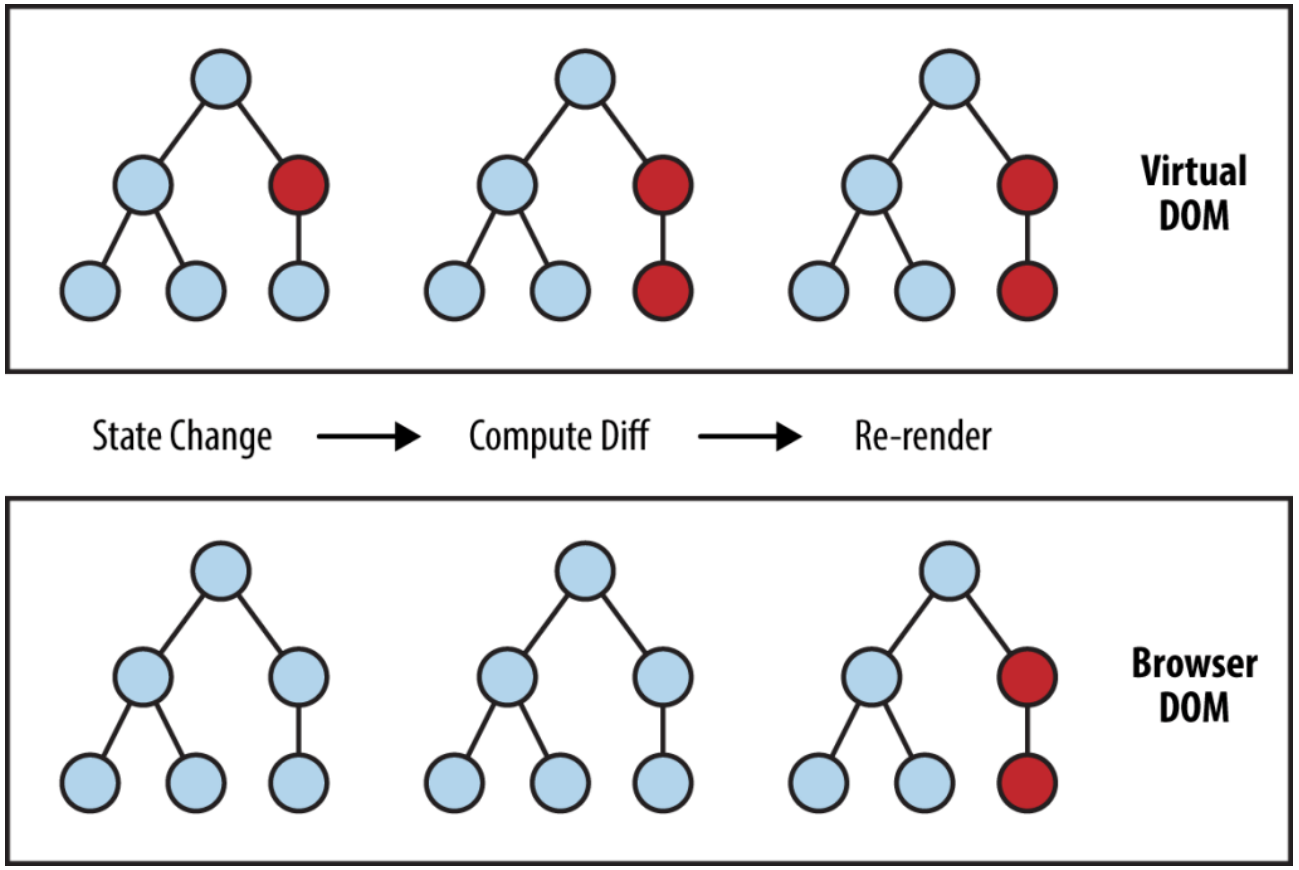
Perché il Virtual DOM è più veloce?

Una volta fatto ciò, il DOM virtuale calcola il miglior metodo possibile per apportare queste modifiche al DOM reale.

Questo assicura che vengano effettuate le minime indispensabili operazioni sul vero DOM.

Riducendo così i costi di elaborazione per l'aggiornamento del DOM reale.

Perché il Virtual DOM è più veloce?



Perché il Virtual DOM è più veloce?

I nodi rossi rappresentano i nodi che sono stati modificati.

Questi nodi rappresentano gli elementi dell'interfaccia utente a cui è stato modificato il loro stato.

Viene quindi calcolata la differenza tra la versione precedente dell'albero DOM virtuale e l'albero DOM virtuale corrente.

L'intero sottostruttura principale viene renderizzata per fornire l'interfaccia utente aggiornata.

Questo albero aggiornato viene quindi riportato in modo batch sul vero DOM.

Come React usa il DOM virtuale?

In React ogni elemento dell'interfaccia utente è un componente e ogni componente ha uno stato.

React segue il **pattern observable** e ascolta i cambiamenti di stato.

Quando lo stato di un componente cambia stato, React aggiorna l'albero DOM virtuale.

Dopo aver aggiornato il DOM virtuale, React confronta la versione corrente del DOM virtuale con la versione precedente del DOM virtuale.

Questo processo si chiama "**diffing**".

Come React usa il DOM virtuale?

Tutti questi dettagli vengono «nascosti» allo sviluppatore e dunque non si deve preoccupare di ciò che deve essere effettivamente sottoposto di nuovo a rendering.

React, infatti, consente di scrivere il codice come se si stesse rieseguendo il rendering dell'intero DOM ogni volta che lo stato dell'applicazione cambia, anche se questo non è vero per quanto detto in precedenza.

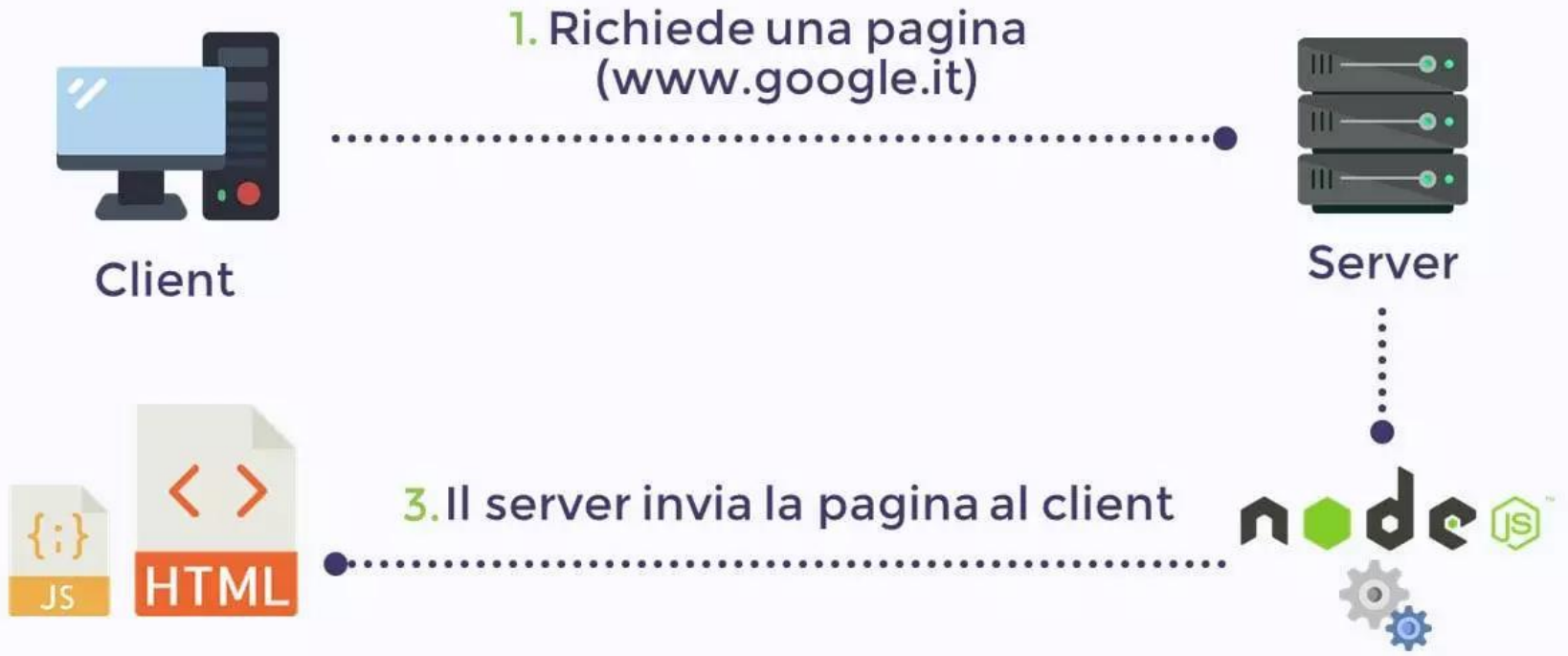
Introduzione a Node.js

Node.js consente di utilizzare il linguaggio Javascript sul server, permettendo così di scrivere Javascript al di fuori del browser (lato client)!

Ha tutto il potere di JavaScript e ci offre un modo completamente nuovo di sviluppare siti web dinamici.

Javascript lato server con Node.js

node academy



Che cosa ha di differente Node.js

La principale differenza sta nel fatto che Node.js, come Javascript, **è basato sugli eventi**, ed è proprio da qui che trae tutta la sua potenza e velocità.

Questo “nuovo” approccio ha cambiato l’intero modo di scrivere le applicazioni web!

Cosa possiamo fare con Node.js

Con Node.js, è possibile creare applicazioni veloci, come ad esempio:

- Un chat server
- Un sistema di upload performante
- Più in generale:

ogni applicazione che deve rispondere a numerose richieste in modo rapido ed efficiente,

in tempo reale

Cosa è allora Node.js ?

Node.js ***non nè un framework nè un linguaggio****.*

Ma allora cos'è?

È una tecnologia, basata su Javascript, e ci permette di ***eseguire delle funzioni sul server****.*

Se volessimo fare un paragone potremmo dire che per certi versi è più simile a C, in quanto sono entrambi ***di basso livello****, che a PHP o Java.*

Node.js® is a JavaScript runtime built on

Chrome's V8 JavaScript engine

V8 Engine

La piattaforma è basata sul **JavaScript Engine V8**, che è il runtime di Google utilizzato anche da Chrome e disponibile sulle principali piattaforme, anche se maggiormente performante su sistemi operativi UNIX-like.

Difficile da padroneggiare ?

*In realtà gli sviluppatori Node.js hanno creato degli appositi strumenti per **facilitare lo sviluppo web**. Parliamo di framework come **Express, Hapi, Sails**.*

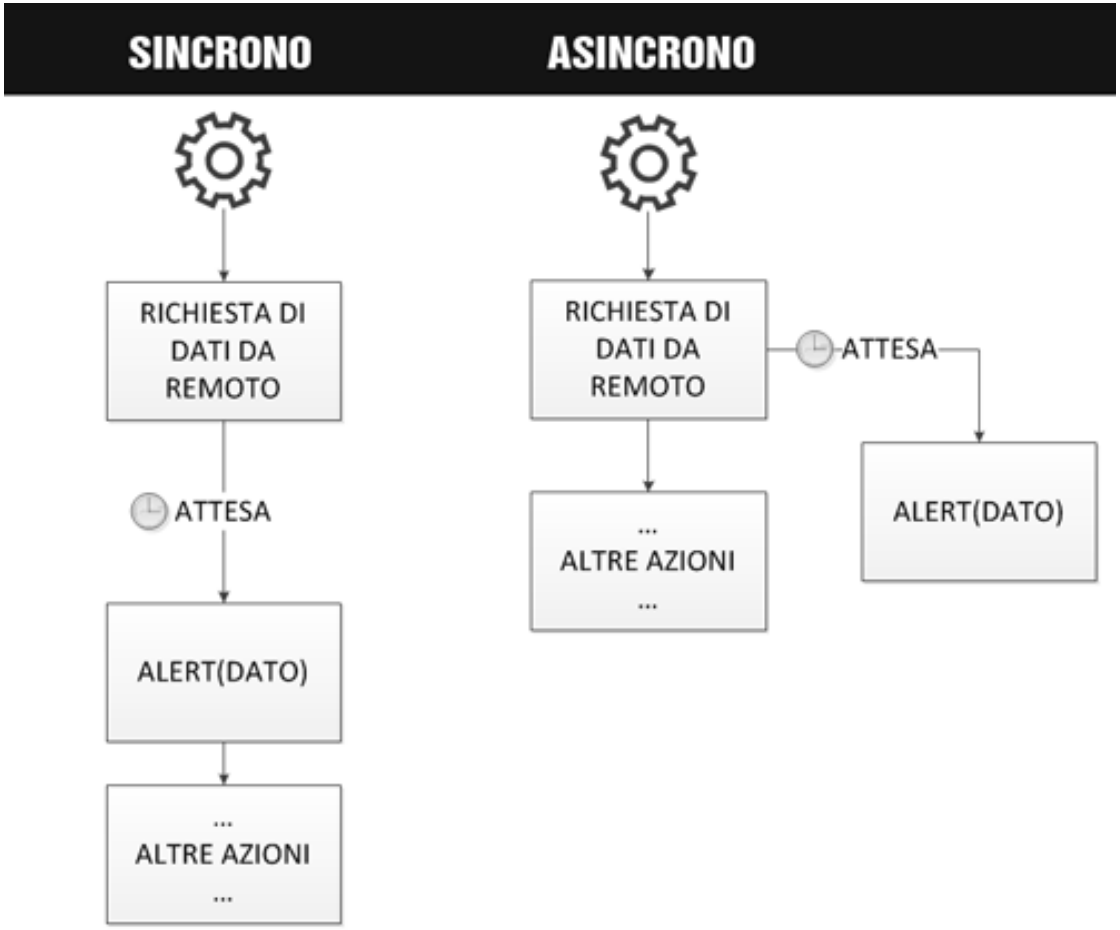
Questi framework permettono di evitare compiti ripetitivi, imposti dalla natura a basso livello di Node.js

Node.JS Approccio asincrono

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 comuni in cui una azione succede ad un’altra solo dopo che essa è stata completata.

Approccio asincrono



Approccio sincrono - esempio

Per capire ancora meglio ecco un breve snippet di pseudo-codice che mette in risalto la differenza tra i due approcci:

```
/** approccio sincrono (classico) **/
var dato = ottieniDatoDaRemoto(url);
alert(dato);
```

In questo caso l'esecuzione aspetta la ricezione dei dati prima di effettuare l'alert.

Approccio asincrono - esempio

```
/** approccio ad eventi (asincrono) **/
ottieniDatoDaRemoto(url, function(dato) {
alert(dato);
}); //la funzione ritorna subito
```

In questo caso invece l'azione da effettuare una volta ottenute le informazioni richieste non è stata scritta direttamente dopo la prima, ma è stata passata come parametro alla funzione ottieniDatoDaRemoto. sotto forma di funzione callback.

Installazione di Node.js

L'**installazione di Node.js** è abbastanza semplice.

Dal sito ufficiale è possibile scaricare l'installer di Node.js per Windows e per Mac e il codice sorgente.

Node.js è comunque disponibile nei principali repository per sistemi operativi Linux.

Installazione di Node.js

Una volta installato avremo a disposizione nuovi comandi:

- **node** che permette di eseguire un'applicazione contenuta in un file passato come parametro
- **node-waf**, un tool per facilitare la creazione di nuovi moduli per Node.js
- **npm**, il package manager, utilissimo per scaricare e installare numerose applicazioni e plugin.

NPM (Node Package Manager)

NPM è quindi un package manager (*Node.js Package Manager*).

Lo strumento che permette di includere, rimuovere e aggiornare le librerie all'interno di un proprio progetto attingendo dal più esteso repository del mondo detto «Software Registry».

NPN Registry contiene più di 800,000 code package.

Tutti i moduli sono pubblicati sul sito http://npmjs.org

NPM (Node Package Manager)

Tutti i pacchetti NPM sono definiti all'interno di files chiamati **package.json.**

Il contenuto di package.json deve essere scritto in JSON.

Due campi sono obbligatori:

- name
- version

NPM (Node Package Manager)

Esempio:

```
{
        "name" : "foo",
        "version" : "1.2.3",
        "description" : "A package for fooing things",
        "main" : "foo.js",
        "keywords" : ["foo", "fool", "foolish"],
        "author" : "John Doe",
        "licence" : "ISC"
}
```

NPM (Node Package Manager)

Gestione delle dipendenze dei moduli:

NPM gestisce le dipendenze tra i moduli definendole all'interno del file package.json.

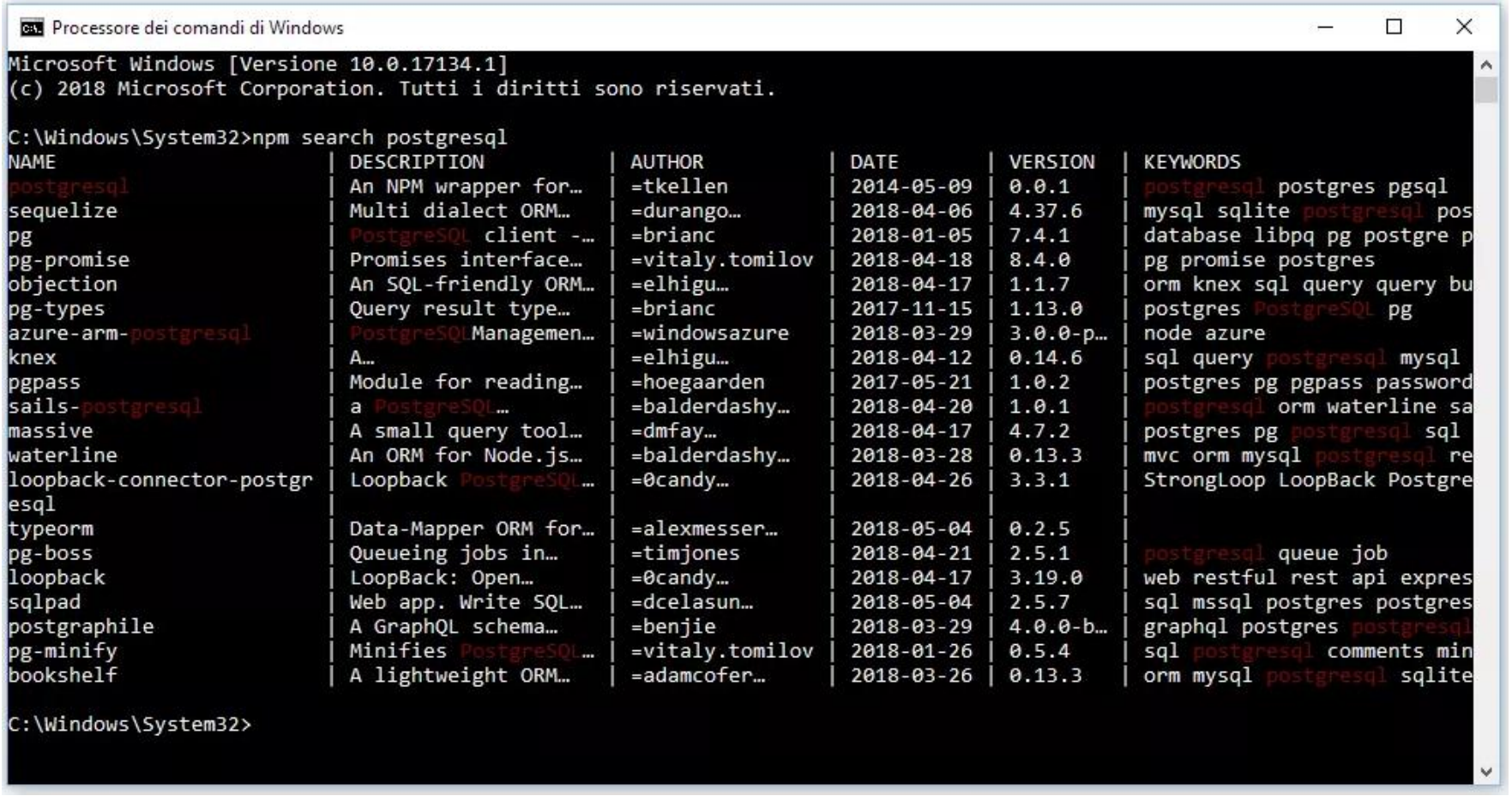
Cercare un modulo:

E' possibile effettuare una ricerca dalla console in questo modo:

```
npm search postgresql
```

Avvierà una ricerca per tutti i moduli che hanno qualcosa a che fare con il database PostgreSQL

NPM (Node Package Manager)



NPM (Node Package Manager)

Installazione di un modulo:

Un modulo è facilmente installabile, basta posizionarsi nel terminale all'interno della cartella del progetto e scrivere:

```
npm install nomemodulo
```

Il modulo verrà installato localmente, ovvero soltanto all'interno del progetto specifico.

Se ci sono più progetti, si dovrà rilanciare il comando per ciascuno. Questo consente di **utilizzare diverse versioni dello stesso modulo a seconda dei progetti**.

NPM (Node Package Manager)

Installazione di un modulo globale o locale:

NPM installa i moduli localmente per ogni progetto, motivo per cui crea
sottocartelle *node_modules* all'interno del progetto.

Tuttavia, va detto che NPM può essere utilizzato anche per **installare moduli globalmente**.

Questo è utile nei **rari casi** in cui il modulo fornisce file eseguibili (e non solo .js).

```
npm install -g create-react-app
```

Creazione di una React application

Installazione di un modulo globale:

```
npm install -g create-react-app
```

Creazione del dell'applicazione React :

```
npx create-react-app myfirstreact
```

Il modulo `create-react-app` crea tutto quanto necessario per l'esecuzione dell'applicazione `myfirstreact`

Esecuzione di una React application

Posizionamento nella cartella:

```
cd myfirstreact
```

Run:

```
npm start
```

Una finestra conterrà la nuova applicazione appena creata:

Esecuzione di una React application