

Corso front-end BASE

Mirko Campari

4 capitoli e 4 test

Cosa affronteremo in questo capitolo

Angular e JS

- Architettura e installazione
- Introduzione a Typescript

Node JS

- Cos'è Node.js
- NodeJS Command Line Interface

Test teorici

- Introduttivo e di conoscenza base
- Di fine capitolo

Pratica

- Primi script con TypeScript, JS e java
- Test pratico di gruppo

4capitoli totali con 4 test pratici e teorici

Cosa sono back end e front end

I termini inglesi **front end** e **back end** in informatica denotano, rispettivamente, la parte visibile all'utente di un programma e con cui egli può interagire, nella maggior parte dei casi, un'interfaccia utente, e la parte che permette l'effettivo funzionamento di queste interazioni e limitando le possibilità la dove necessario.

Il **front end**, nella sua accezione più generale, è responsabile dell'acquisizione dei dati di ingresso e della loro elaborazione con modalità conformi a specifiche predefinite e invarianti, tali da renderli utilizzabili dal **back end**.

Il collegamento del **front end** al **back end** è un caso particolare di interfaccia.

Cosa sono back end e front end prt 2

Dopo la pura teoria ora andiamo a confrontare le differenze tra lo sviluppo back-end e quello front-end, le differenze di solito si presentano nelle seguenti categorie oltreché nelle specifiche tecniche del progetto:

- **Le aree di cui si occupano questi due tipi di sviluppo.**
- **La tecnologia e gli strumenti utilizzati da ciascuno.**
- **I linguaggi utilizzati nel backend rispetto al frontend.**
- **Requisiti e competenze professionali.**

Cos'è *HTML*?

In informatica l'HyperText Markup Language, cioè linguaggio a marcatori per ipertesti, comunemente noto con l'acronimo HTML, è un linguaggio di markup.

Nato per la formattazione e impaginazione di documenti ipertestuali disponibili nel web 1.0, oggi è utilizzato principalmente per il disaccoppiamento della struttura logica di una pagina web definita appunto dal markup e la sua rappresentazione, gestita tramite gli stili CSS per adattarsi alle nuove esigenze di comunicazione e pubblicazione all'interno di Internet.

L'HTML è un linguaggio di pubblico dominio, la cui sintassi è stabilita dal World Wide Web Consortium (W3C). È derivato dall'SGML, un metalinguaggio finalizzato alla definizione di linguaggi utilizzabili per la stesura di documenti destinati alla trasmissione in formato elettronico.

La versione attuale, la quinta, è stata rilasciata dal W3C nell'ottobre 2014.

Un esempio di HTML

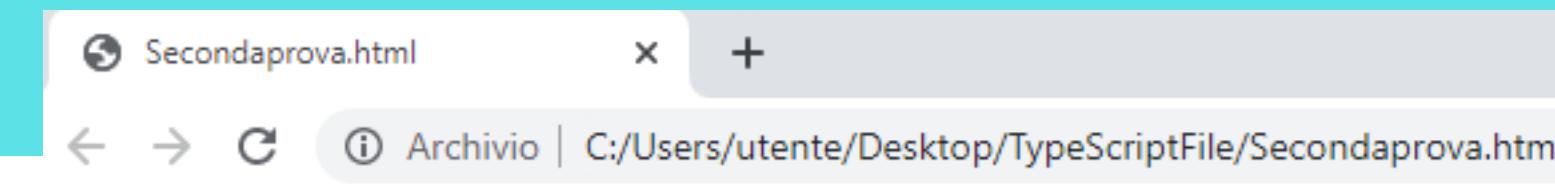
Andiamo ora a vedere un esempio di HTML

```
<!DOCTYPE html>
<html><body>

<h1 style="background-color:DodgerBlue;">Salve Aula!</h1>

<p style="background-color:Tomato;">
Lorem ipsum </p>

</body></html>
```



Quello che vedremo accadere dev'essere
qualcosa di simile a questo

----->

Salve Aula!

-----> **Lorem ipsum**

Cos'è il CSS?

Il CSS, sigla per Cascading Style Sheets, in italiano fogli di stile a cascata; è un linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML, ad esempio i siti web e relative pagine web. Le regole per comporre il CSS sono contenute in un insieme di direttive emanate da W3C.

L'introduzione del CSS si è resa necessaria per separare i contenuti delle pagine HTML dalla loro formattazione o layout e permettere una programmazione più chiara e facile da utilizzare, sia per gli autori delle pagine stesse sia per gli utenti, garantendo contemporaneamente anche il riutilizzo di codice ed una sua più facile manutenzione.

Cos'è Java Script?

In informatica JavaScript è un linguaggio di programmazione multi paradigma orientato agli eventi, comunemente utilizzato nella programmazione Web lato client (esteso poi anche al lato server) per la creazione, in siti web e applicazioni web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso.

Originariamente sviluppato da Brendan Eich della Netscape Communications con il nome di Mochan e successivamente di LiveScript, e poi stato rinominato "JavaScript" ed è stato formalizzato con una sintassi più vicina a quella del linguaggio Java

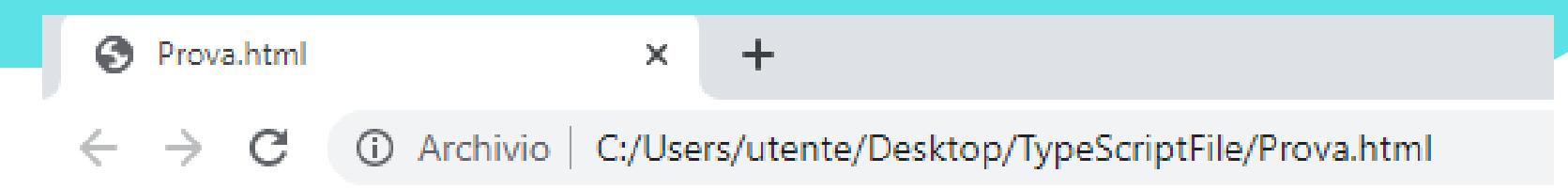
Un piccolo esempio di JS

Naturalmente useremo anche HTML in combinazione con JS

```
<!DOCTYPE html>
<html><body>
<h2>La nostra prima app</h2>

<button type="button"
onclick="document.getElementById('demo').innerHTML = Date()"
Clicca per vedere ora e data.</button>

<p id="demo"></p>
</body> </html>
```



Quello che vedremo accadere dev'essere
qualcosa di simile a questo



La nostra prima app

[Clicca per vedere ora e data.](#)

Mon Jan 09 2023 12:37:18 GMT+0100 (Ora standard dell'Europa centrale)

Cos'è AngularJS? e perchè partiamo da lui

AngularJS è un framework per applicazioni web open source, sviluppato da Google e dalla comunità di sviluppatori individuali, al fine di migliorare la quality of life dello sviluppo di applicazioni su singola pagina.

Per semplificare lo sviluppo e i test di queste applicazioni fornendo un framework lato client con architettura MVC (Model View Controller) e Model-view-viewmodel (MVVM) insieme a componenti usate nelle applicazioni RIA(Rich Internet application).

Il framework lavora leggendo prima la pagina HTML, che ha encapsulati degli attributi personalizzati addizionali, interagendo con questi attributi come delle direttive per legare le parti di ingresso e uscita della pagina al modello che è rappresentato da variabili standard JavaScript. Il valore di queste variabili può essere impostato manualmente nel codice o recuperato da risorse JSON

Cos'è Angular?

Angular è un framework open source per lo sviluppo di applicazioni web con licenza MIT, evoluzione del già citato AngularJS.

Sviluppato principalmente da Google, la sua prima release è avvenuta il 14 settembre 2016.

Perchè Angular?

Angular è stato completamente riscritto rispetto a AngularJS e le due versioni non sono compatibili.

Il linguaggio di programmazione usato per AngularJS è JavaScript mentre quello di Angular è TypeScript.

Recap post PRIMA lezione:

JS: è un linguaggio di programmazione multi paradigma orientato agli eventi

HTML: l'HyperText Markup Language, noto come HTML, è un linguaggio di markup

CSS: CSS ovvero Cascading Style Sheets è un linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML

AngularJS: è un framework per applicazioni web open source per lo sviluppo semplificato di applicazioni su singola pagina.

Recap Concettuale:

La differenza tra il back-end è il front-end può essere semplificata nel concetti di cosa viene mostrato all'utente e le interazioni che esso può eseguire FRONT e invece il BACK è il cosa, nella pratica, rende FUNZIONANTI le sue interazioni e che conserva dati e strutture.

Abbiamo visto la differenza tra AngularJS E ANGULAR, ovvero, la differenza nella programmazione del framework, le prestazioni e l'incompatibilità tra i due

Caratteristiche di Angular

Le applicazioni sviluppate in Angular vengono eseguite interamente dal web browser dopo essere state scaricate dal web server (elaborazione lato client). Questo comporta il risparmio di dover spedire indietro la pagina web al web-server ogni volta che c'è una richiesta di azione da parte dell'utente. Il codice generato da Angular gira su tutti i principali web browser moderni quali ad esempio Chrome, Microsoft Edge, Opera, Firefox, Safari ed altri.

Angular è stato progettato per fornire uno strumento facile per sviluppare app che girano su qualunque piattaforma .

Le applicazioni web in Angular in combinazione con il toolkit open source Bootstrap diventano responsive, ossia il design del sito si adatta in funzione alle dimensioni del dispositivo utilizzato.

È in corso di sviluppo un altro toolkit di design responsivo, Flex Layout, più semplice da usare rispetto a Bootstrap e concepito appositamente per Angular.

Altro toolkit che facilita la progettazione in Angular è Angular Material, una serie di componenti che permette di creare una pagina web molto velocemente: con l'utilizzo combinato di Flex Layout ed Angular Material si possono creare siti e app web responsive basate su Angular.

Tipizzazione cenni tecnici

Debole: *In informatica e programmazione, la tipizzazione dinamica è la politica di tipizzazione, ovvero di assegnazione di tipi alle variabili, in cui il controllo del tipo della variabile è effettuato a runtime piuttosto che in fase di compilazione.*

Forte: *In informatica, e in particolare in programmazione, la tipizzazione forte (strong typing) può essere usata per caratterizzare il tipo di regole che un determinato linguaggio di programmazione impone, a livello sintattico o semantico, circa la tipizzazione dei dati e all'uso dei dati in relazione al loro tipo. L'opposto della tipizzazione forte è la tipizzazione debole.*

L'implementazione della tipizzazione statica cambia molto da linguaggio a linguaggio, ma rimangono costanti alcune caratteristiche. Il programma mantiene in una tabella dei valori tutte le variabili dichiarate dal programmatore, insieme al loro tipo e al valore corrente. Grazie a ciò, un tentativo di assegnamento di un valore di tipo diverso da quello di una variabile causa un errore terminale.

I linguaggi a tipizzazione statica più comuni sono Java e C/C++.

Tipizzazione FORTE cenni tecnici

In un linguaggio fortemente tipizzato, il programmatore è tenuto a specificare il tipo di ogni elemento sintattico che durante l'esecuzione denota un valore (per esempio un valore costante, una variabile o un'espressione), e il linguaggio garantisce che tale valore sia utilizzato in modo coerente con il tipo specificato: per esempio, non è possibile eseguire una somma aritmetica su dati di tipo stringa. Questo concetto generale può applicarsi con diverse sfumature; a seconda del contesto. Dunque, l'espressione linguaggio fortemente tipizzato può riferirsi a cose leggermente diverse:

- **tipizzazione statica: tutti i controlli sull'uso corretto dei valori rispetto al loro tipo vengono eseguiti durante la compilazione;**
- **sicurezza rispetto ai tipi (type safety): è garantito che vengano fatti controlli esaustivi sull'uso dei valori rispetto al loro tipo, ma non necessariamente durante la compilazione (un programma potrebbe fallire durante l'esecuzione a causa di violazioni di tipo)**
- **impossibilità di eseguire conversioni di tipo;**
- **impossibilità di eseguire conversioni di tipo implicite;**

Tipizzazione FORTE cenni tecnici prt 2

Un esempio limite di linguaggio debolmente tipizzato è il linguaggio macchina, in cui un'area di memoria, rappresentata nel codice da un indirizzo di memoria, può essere usata indifferentemente per contenere valori di qualunque tipo, numeri interi, numeri con la virgola, caratteri, e così via. I linguaggi di programmazione ad alto livello tendono ad avere sistemi dei tipi più sicuri e quindi ad avvicinarsi all'ideale della forte tipizzazione, ma in genere rimangono disponibili meccanismi per trattare dati in modo flessibile che si prestano a essere utilizzati per violare il sistema dei tipi. Per esempio, il linguaggio C fornisce almeno tre meccanismi che lo qualificano come linguaggio debolmente tipizzato:

- **le operazioni di casting, che consentono di forzare l'interpretazione di un qualunque valore secondo un qualunque tipo (anche un tipo diverso da quello a cui il valore è stato precedentemente associato);**
 - **i puntatori a void, che godono di conversione di tipo implicita verso qualunque altro tipo puntatore;**
 - **le unioni, che consentono di interpretare una collezione di dati correlati secondo diverse attribuzioni di tipo indipendenti.**
-

Esempi pratici di tipizzazione statica

Java

```
public class EsempioTipizzazione {public static void main( String[] args ) {  
  
    int a; //dichiara la variabile intera a  
    a = 3.5; //ERRORE! Il valore 3.5 è numerico a virgola mobile, la variabile a invece è intera }  
}
```

C

```
int main() {  
    char c; /* dichiara una variabile di tipo carattere */  
    c = "Esempio";  
    /* ERRORE! "Esempio" è una stringa, non può essere assegnata a una variabile carattere */  
}
```

Esempi pratici di tipizzazione debole

Python

x = 12 # int || Integer || NUMERO INTERO

x = "parola alfanumerica" # String || Stringa || può contenere qualsiasi combinazione di lettere e numeri all'interno degli apici --> ""

x = False # Bool || Booleani || può contenere solo True o False

x = 123.23434 # Numero complesso FLOAT può contenere numeri con al virgola fino ad una determinata soglia

Print(x) // andrà in errore?

Esercitazione 1: Html e JS

Scrivere un documento HTML contenente una form contenente i seguenti campi:

- lavoro(casella di testo a scelta multipla)
- nome (casella di testo editabile lunga 30 caratteri) •
- email (casella di testo editabile lunga 30 caratteri)
- bottone di invio e bottone di reset
- una casella di testo per le note

Aggiungere al documento HTML JavaScript che esegua i seguenti controlli:

- verifica che il campo nome e l'email non siano vuoti ne troppo lunghi;
- verifica che la selezione del lavoro sia corretta.

Inoltre, fare si che le funzioni JS vengano eseguite solo quando l'utente invia la form.

Cos'è Type Script?

TypeScript è JavaScript con sintassi aggiunta per i tipi. Possiamo anche definirlo un superset sintattico di JavaScript che aggiunge tipizzazione statica, ciò significa sostanzialmente che TypeScript aggiunge la sintassi a JavaScript, consentendo agli sviluppatori di aggiungere tipi.
Quindi essere un "Superset sintattico" significa che condivide la stessa sintassi di base di JavaScript, ma aggiunge qualcosa ad esso.

JavaScript è un linguaggio vagamente tipizzato e è spesso molto difficile capire quali tipi di dati vengono trasmessi in JavaScript, i parametri e le variabili delle funzioni non hanno alcuna informazione utile alla tipizzazione.

TypeScript consente di specificare i tipi di dati che vengono passati all'interno del codice e ha la capacità di segnalare errori quando i tipi non corrispondono. Ad esempio, TypeScript segnalerà un errore quando si passa una stringa in una funzione che prevede un numero. JavaScript no.

TypeScript utilizza il controllo del tipo in fase di compilazione. Ciò significa che controlla se i tipi specificati corrispondono prima di eseguire il codice, non durante l'esecuzione del codice.

Come si usa TypeScript?

Un modo comune per utilizzare TypeScript consiste nell'utilizzare il compilatore TypeScript ufficiale, che transpila il codice TypeScript in JavaScript.

La sezione successiva mostra come ottenere la configurazione del compilatore per un progetto locale.

Alcuni popolari editor di codice, come Visual Studio Code, hanno il supporto TypeScript integrato e possono mostrare errori mentre scrivi il codice!

Come si usa TypeScript?

Per installare Type Script avremo bisogno del nostro NPM, esegui il seguente comando per installare il compilatore:

npm install typescript --save-dev

Che dovrebbe darti un output simile a:

```
□ Prompt dei comandi
Microsoft Windows [Versione 10.0.19045.2364]
(c) Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\utente>npm install typescript --save-dev
added 1 package, and audited 4 packages in 13s

  found 0 vulnerabilities
  npm notice
  npm notice New major version of npm available! 8.19.2 -> 9.2.0
  npm notice Changelog: https://github.com/npm/cli/releases/tag/v9.2.0
  npm notice Run npm install -g npm@9.2.0 to update!
  npm notice

C:\Users\utente>
C:\Users\utente>
```

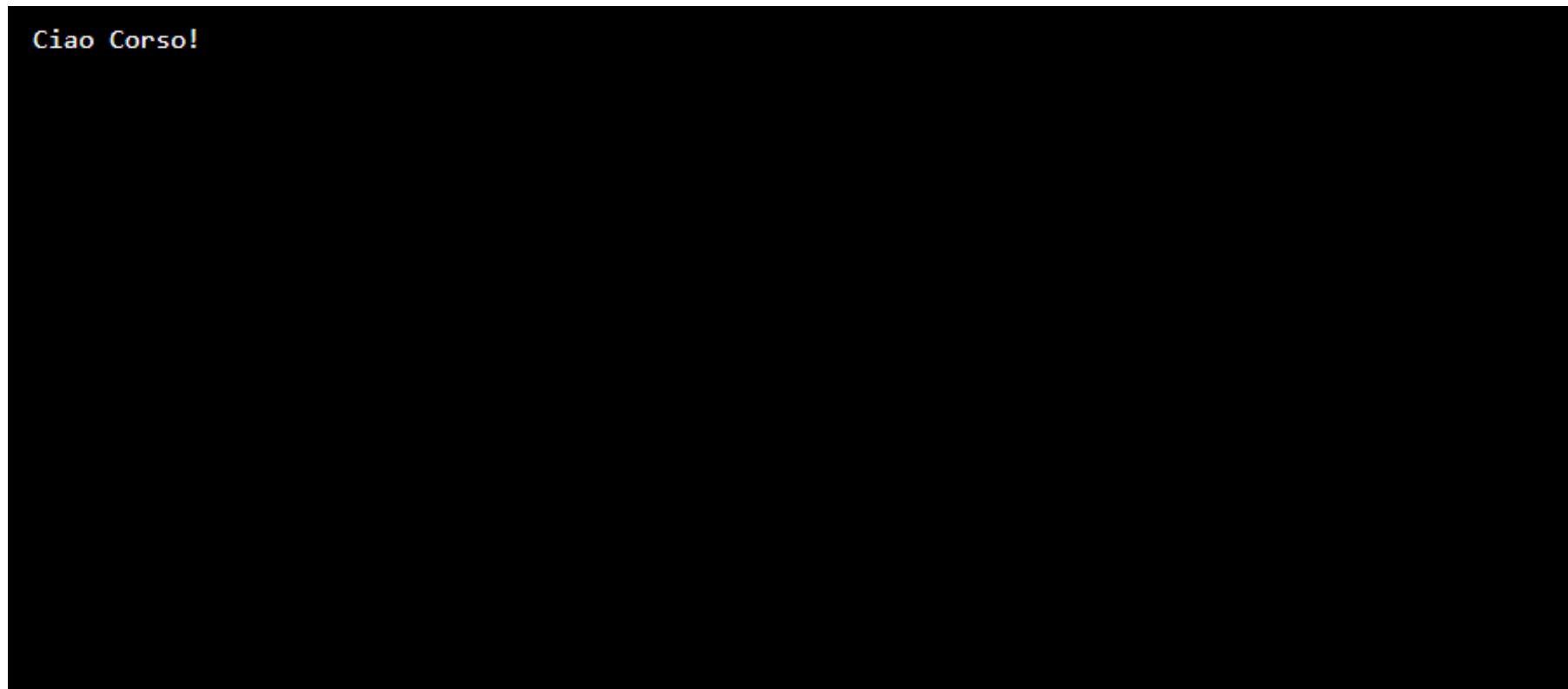
Il primo script in TypeScript

Andiamo ora a vedere il cosiddetto "hello word", ovvero il primo script che faremo girare, utilizzando type-script, faremo stampare una variabile tramite un semplice comando

```
let hellostring : string = "Ciao Corso!"; # tipizzazione esplicita
```

```
let hellostring = "Ciao Corso!";
console.log(hellostring); #tipizzazione vaga o implicita
```

**Dovrebbe darvi un output simile a questo
che vedete nell'immagine**



Ma cos'è NPM?

- **npm è la libreria software (registro) più grande del mondo**
- **npm è anche un software Package Manager e Installer**

Il registro contiene oltre 800.000 pacchetti di codice .

Il nome completo è Node Package Manager.

Gli sviluppatori open source usano npm per condividere il software.

Molte organizzazioni usano anche npm per gestire lo sviluppo privato.

L'uso di npm è gratuito

Puoi scaricare tutti i pacchetti software pubblici di npm senza alcuna registrazione o accesso proprio per via della sua vocazione open source.

Ma cos'è NPM?

npm è installato con Node.js

Ciò significa che dovremo installare Node.js per installare npm sul tuo computer.

Scarica Node.js dal sito web ufficiale di Node.js: <https://nodejs.org>

Il nome npm (Node Package Manager) deriva da quando npm è stato creato per la prima volta come gestore di pacchetti per Node.js.

Tutti i pacchetti npm sono definiti in file denominati package.json .

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

Nel file di definizione devono essere presenti almeno due campi: name e version .

Cos'è Node.js?

Node.js è un runtime system open source multipiattaforma orientato agli eventi per l'esecuzione di codice JavaScript, costruito sul motore JavaScript V8 di Google Chrome.

Molti dei suoi moduli base sono scritti in JS, e gli sviluppatori possono scrivere nuovi moduli nello stesso linguaggio di programmazione. In origine, JS veniva utilizzato soprattutto 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 direttamente all'interno di un web browser.

Node.js consente invece di utilizzare JS 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 in questo modo permette di implementare il paradigma "JavaScript everywhere", unificando lo sviluppo di applicazioni Web intorno a un unico linguaggio.

Node.js ha un'architettura orientata agli eventi che rende possibile l'I/O asincrono.

Questo design punta ad ottimizzare il throughput e la scalabilità nelle applicazioni web con molte operazioni di input/output, è inoltre ottimo per applicazioni web sistema real-time (ad esempio programmi di comunicazione in tempo reale o browser game).

Come controllare node.js

Per controllare se Node.js è installato, digitare node -v nel terminale.

**Se è tutto ok dovrebbe darvi
un output simile a questo che
vedete nell'immagine**

□ Prompt dei comandi

```
Microsoft Windows [Versione 10.0.19045.2364]
(c) Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\utente>node -v
v18.12.1

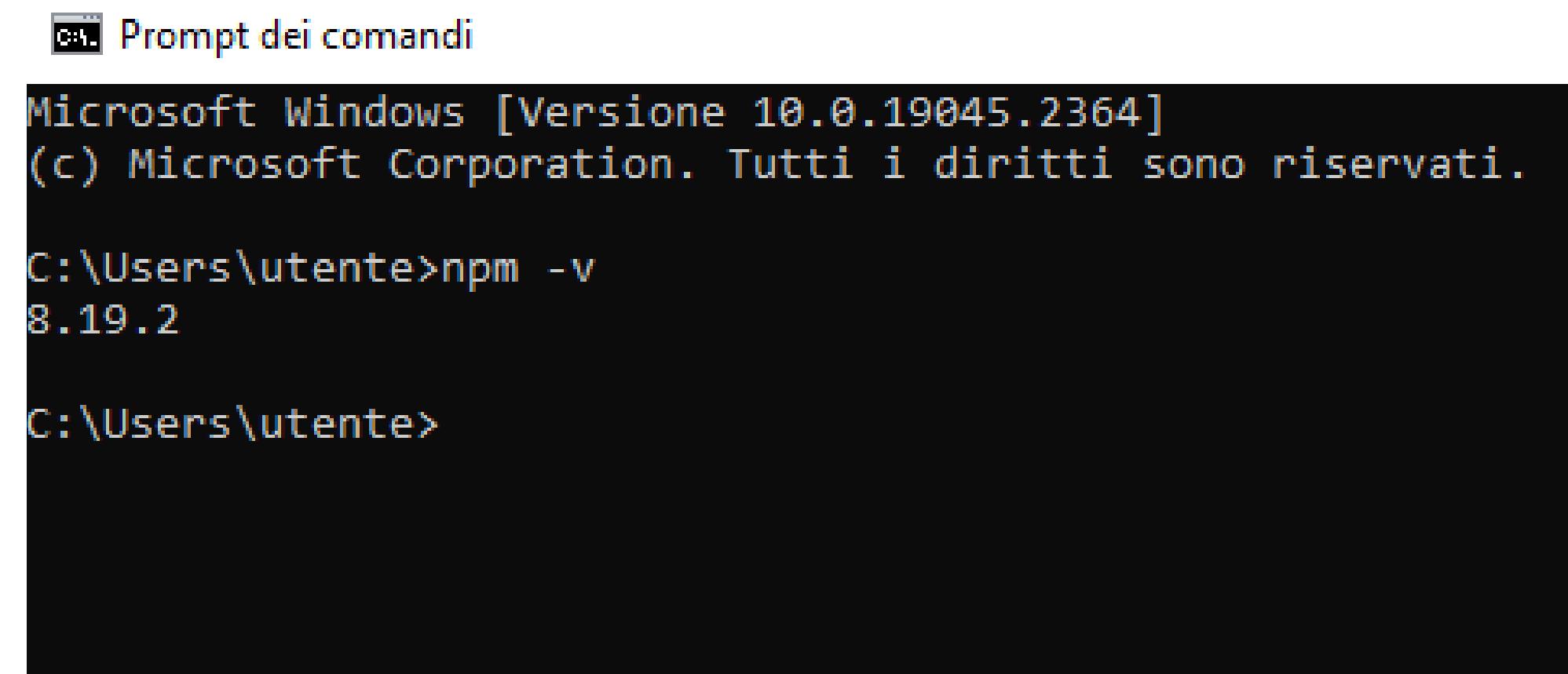
C:\Users\utente>
```

Come controllare NPM

Per controllare se NPM è installato, digitare npm -v nel terminale.

Se NPM è installato correttamente dovreste vedere il numero della versione

**Se è tutto ok dovrebbe darvi
un output simile a questo che
vedete nell'immagine**



A screenshot of a Windows command prompt window titled "Prompt dei comandi". The window shows the following text:
Microsoft Windows [Versione 10.0.19045.2364]
(c) Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\utente>npm -v
8.19.2

C:\Users\utente>

Hello world in Node.js

Nel seguente esempio "hello world", molte connessioni possono essere gestite contemporaneamente. Ad ogni connessione, la richiamata viene attivata, ma se non c'è lavoro da fare, Node.js andrà in sospensione.

```
const http = require('http');
const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Cenni tecnici basilari di Node.js

Gli elementi principali in NODE.JS sono i moduli e i nodi

Che cos'è un modulo in Node.js? Considera i moduli come le stesse librerie JavaScript o più tecnicamente potremmo anche dire l'insieme di tutte le funzioni che desideri includere nella tua applicazione.

Oltre tutto i moduli possono essere di diversi tipi

Moduli integrati

Node.js ha una serie di moduli integrati che puoi utilizzare senza ulteriori installazioni che consentono a n.js nativamente di funzionare.

https://www.w3schools.com/nodejs/ref_modules.asp

Cenni tecnici basilari di Node.js prt 2

**Per includere un modulo, bisogna utilizzare la KEYWORD require() funzione
che dovrà essere accompagnata con il nome del modulo:**

```
var http = require('http');
```

**Ora la nostra applicazione avrà accesso al modulo HTTP ed è in grado di
creare il server che poi noi andremo ad istanziare:**

```
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.end('Hello World!');  
}).listen(8080);
```

Cenni tecnici basilari di Node.js prt 3

Possiamo andare anche a creare dei moduli personali e includerli facilmente nelle tue applicazioni; Creiamo un modulo che restituisca la data e l'ora

```
exports.myDateTime = function () {  
    return Date();  
};
```

Utilizziamo la KEYWORD exports per rendere disponibili proprietà e metodi all'esterno del file del modulo.

Salviamo ora il codice sopra un file chiamato "primomodulo.js"

Cenni tecnici basilari di Node.js prt 4

Ora possiamo includere e utilizzare il modulo in uno qualsiasi dei nostri file Node.js. Usiamo il modulo "primomodulo.js" in un file Node:

```
var http = require('http');
var dt = require('./primomodulo');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write("La data e l'ora al momento sono: " + dt.myDateTime());
  res.end();
}).listen(8080);
```

Si noti che utilizziamo ./ per individuare il modulo, ciò significa che il modulo si trova nella stessa cartella del file Node.js

Cenni tecnici basilari di Node.js prt 5

Salviamo ora il nostro codice sopra un file chiamato "modulo_demo.js" e avvia il file Avvia demo_module.js:

C:\Users\Your Name> node modulo_demo.js

SIAMO APPENA ANDATI AD ESEGUIRE UN NODO LOGICO TRAMITE I MODULI, O LE SU FUNZIONI, CHE ABBIAMO DECISO DI IMPORTARE AL SUO INTERNO

Esercitazione 2: HTML,js, CSS

HTML: Andare a creare una form con 4 parametri in ingresso dei tipi: Numerico, booleano, string e check type

si può accedere alla form solo dopo un check password che sarà: "AULA Front-end"

due bottoni invio dati e reset

Js: Controllo che tutti i campi siano correttamente tipizzati e che abbiano compilazione adeguata, dopo di che far si che vengano stampati a video sulla pagina con

CSS: due background color diversi, centrati e con grandezze diverse.

Cenni tecnici basilari di Node.js prt 5

HTTP è il modulo che consente a Node.js di trasferire dati tramite Hyper Text Transfer Protocol (HTTP).

Il modulo HTTP può creare un server HTTP che ascolta le porte del server e restituisce una risposta al client andiamo ora ad utilizzare il createServer() è un metodo che viene usato per creare un server HTTP:

```
var http = require('http');

http.createServer(function (req, res) {
  res.write('Serve c'è'); //Scriviamo quella che sarà la risposta del server
  res.end(); //Specifichiamo che il sistema da qui non avrà altri input client
}).listen(8080);
```

Cenni tecnici basilari di Node.js prt 6

Possiamo anche dividere una stringa di query e per farlo esistono moduli integrati per suddividere facilmente la stringa di query in parti leggibili, come il modulo URL.

```
var http = require('http');
var url = require('url');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  var q = url.parse(req.url, true).query;
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(8080);
```

Cenni tecnici basilari di Node.js prt 7

**Ora andiamo a vedere il modulo FILE SYSTEM il 'suo uso comune passa da:
Leggere i file, Creare file, Aggiornare i file, Cancellare file e Rinominare i file**

```
var fs = require('fs');
```

Il *fs.readFile()* viene utilizzato per leggere i file sul tuo computer, creiamo ora un file HTML ,nella stessa cartella di Node.js:

```
<html>
<body>
<h1>My Header</h1>
<p>My paragraph.</p>
</body>
</html>
```

Cenni tecnici basilari di Node.js prt 7

Creiamo un file Node.js che legge il file HTML

```
var http = require('http');
var fs = require('fs');

http.createServer(function (req, res) {
  fs.readFile('demofile1.html', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);
```

Cenni tecnici basilari di Node.js prt 8

Ora creiamo un vero e proprio nuovo file usando il metodo fs.appendFile():

```
var fs = require('fs');

fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

Il modulo File System dispone di diversi metodi per la creazione di nuovi file:

fs.appendFile() fs.open() fs.writeFile()

Cenni tecnici basilari di Node.js prt 8

Il fs.appendFile() aggiunge il contenuto specificato a un file.

Se il file non esiste, il file verrà creato:

```
var fs = require('fs');

fs.appendFile('mynewfile1.txt', 'ciao a tutti!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

Cenni tecnici basilari di Node.js prt 8

Il fs.open() accetta un "flag" come secondo argomento, se il flag è "w" per "scrittura", il file specificato viene aperto per la scrittura.

Se il file non esiste, viene creato un file vuoto:

```
var fs = require('fs');

fs.open('mynewfile2.txt', 'w', function (err, file) {
  if (err) throw err;
  console.log('Saved!');
});
```

Cenni tecnici basilari di Node.js prt 8

Il fs.writeFile() sostituisce il file e il contenuto specificati se esiste.

Se il file non esiste, verrà creato un nuovo file, contenente il contenuto specificato:

```
var fs = require('fs');

fs.writeFile('mynewfile3.txt', 'Hello content!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

Cenni tecnici basilari di Node.js prt 8

Il modulo File System dispone di due metodi specifici per l'aggiornamento file:

fs.appendFile() fs.writeFile()

Il fs.appendFile() nello specifico aggiunge al file il contenuto specificato

```
var fs = require('fs');

fs.appendFile('mynewfile1.txt', ' This is my text.', function (err) {
  if (err) throw err;
  console.log('Updated!');
});
```

Cenni tecnici basilari di Node.js prt 8

**Il modulo File System dispone di un metodo per eliminare un file il fs.unlink()
elimina il file specificato:**

```
var fs = require('fs');

fs.unlink('mynewfile1.txt', function (err) {
  if (err) throw err;
  console.log('File deleted!');
});
```

Cenni tecnici basilari di Node.js prt 8

Il modulo File System per rinominare un file utilizza il fs.rename() che rinomina il file specificato:

```
var fs = require('fs');

fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function (err) {
  if (err) throw err;
  console.log('File Renamed!');
});
```

Cenni tecnici basilari di Node.js prt 9

C:\Users\Your Name>npm install upper-case

Una volta installato, il pacchetto è pronto per l'uso. Includi il pacchetto "maiuscolo" nello stesso modo in cui includi qualsiasi altro modulo:

```
var http = require('http');
var uc = require('upper-case');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(uc.upperCase("sono gigante!"));
  res.end();
}).listen(8080);
```

Esercitazione 3: un po di Node.js

Andiamo a creare un sistema CRUD(Create Rename Update Delate) che inserisca e rimuova elementi a vostra scelta, per tipo e valori, usando il modulo FS e facendolo stampare su un file di html Step1.5 css diviso per tipologia di azione

Dovrà farlo solo quando viene richiamato tramite http server

step2: far si che possa venir stampato partendo da un input da bottone e che sempre tramite un pulsante si possa salvare

Cenni tecnici basilari di Node.js prt 10

Rispetto ad altri linguaggi, node.js ha la caratteristica di essere in Input / Output non bloccante ad eventi come modello .

Nel modello I/O bloccante tipico degli altri linguaggi le risorse rimangono impegnate fintanto che le risposte da parte del server non giungono.

Nel modello non bloccante ad eventi, il singolo thread non attende la risposta dal server, prima di processare nuove richieste, ma durante la fase di elaborazione richiesta, genera una call back e va a processare altre richieste.

Quando una risposta da parte del server arriva viene generato un evento. Questo permette maggiore scalabilità e minori richieste hardware laddove viene fatto un grande uso della rete, mentre diventa sconsigliato e poco efficiente quando le applicazioni necessitano di grande utilizzo di CPU, tipo elaborazioni grafiche .

Cenni tecnici basilari di Node.js prt 10

Ogni azione su un computer è un evento. Come quando viene stabilita una connessione o viene aperto un file.

Gli oggetti in Node.js possono attivare eventi, come l'oggetto readStream attiva eventi durante l'apertura e la chiusura di un file:

```
var fs = require('fs');
var rs = fs.createReadStream('./placeholder.txt');

rs.on('open', function () {
  console.log('Il file dovrebbe essersi aperto'); });
```

Cenni tecnici basilari di Node.js prt 10

Node.js è le sue applicazioni girano su singolo processo, ma supporta simultaneamente anche eventi e callback, quindi le prestazioni migliroano.

Ogni API Node.js è asincrono e viene eseguito come un thread separato, utilizzando chiamate di funzione asincrone, questa è la concorrenza.

Node.js sostanzialmente è tutto il meccanismo di un evento che viene implementato utilizzando la modalità osservatore e/o la modalità di progettazione.

Node.js è simile a entrare in un unico file while (true) fino a quando non vi è alcun osservatore o evento di uscita, ogni evento asincrono genera un osservatore, se si verifica un evento che chiama la funzione di callback.

Cenni tecnici basilari di Node.js prt 10

Node.js ha un modulo integrato, chiamato "events", in cui puoi creare, attivare e ascoltare i tuoi eventi.

Per includere il modulo integrato Events utilizziamo il require() .

Inoltre, tutte le proprietà dei metodi dell'evento sono un'istanza di un oggetto EventEmitter.

Per poter accedere a queste proprietà e metodi, crea un oggetto Event Emitter:

```
var events = require('events');  
var eventEmitter = new events.EventEmitter()
```

Cenni tecnici basilari di Node.js prt 10

Possiamo assegnare gestori di eventi ai tuoi eventi con l'oggetto EventEmitter.

Andiamo ora a creare una funzione che verrà eseguita quando viene attivato un evento "suono".

Per attivare un evento, utilizzare emit()

```
var events = require('events');
var eventEmitter = new events.EventEmitter();
var myEventHandler = function () {
  console.log('sto sentendo un suono');
}
eventEmitter.on('suono!', myEventHandler);
eventEmitter.emit('suono');
```

Esercitazione 3: Node.js ad eventi

**Andiamo a creare un server http un modulo personale che
importeremo con al interno due due (nome, punteggio)
facciamo si che un evento al suo interno vada a far stampare
questi dati oppure in caso di errore un messaggio di gestione**

**Vrs 2:Far si che come url debbano essere immessi i dati che
preferiamo (password)**

**VR3: far si che l'input derivi da una text box dove inseriamo la
password e un bottone per confermare**

Cenni tecnici basilari di Node.js prt 10

Esiste un ottimo modulo per lavorare con i caricamenti di file, chiamato "formidable" andiamo a scaricarlo e installarlo utilizzando NPM:

C:\Users\Your Name>npm install formidable

Dopo aver scaricato il modulo Formidable, puoi includere il modulo in qualsiasi applicazione:

var formidable = require('formidable');

Possiamo ora gestire più agevolmente il caricamento dei dati, sia in nel campo delle prestazioni sia nel campo della mera richiesta di risorse

Cenni tecnici basilari di Node.js prt 10

Ora possiamo creare una pagina web in Node.js che consenta all'utente di caricare file , come prima cosa dobbiamo creare un modulo di caricamento

Creiamo un file Node.js che scrive un modulo HTML, con un campo di caricamento

```
1. var http = require('http');
2.
3. http.createServer(function (req, res) {
4.   res.writeHead(200, {'Content-Type': 'text/html'});
5.   res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');
6.   res.write('<input type="file" name="filetoupload"><br>');
7.   res.write('<input type="submit">');
8.   res.write('</form>');
9.   return res.end(); }).listen(8080);
```

Cenni tecnici basilari di Node.js prt 10

Con il modulo Formidable possiamo analizzare il file caricato quando raggiunge il server.

Quando il file viene caricato e analizzato, viene inserito in una cartella temporanea sul tuo computer.

```
1. var http = require('http');
2. var formidable = require('formidable');
3. http.createServer(function (req, res) {
4.   if (req.url == '/fileupload') {
5.     var form = new formidable.IncomingForm();
6.     form.parse(req, function (err, fields, files) {
7.       res.write('File uploaded');
8.       res.end(); });
9.   } else {
10.     res.writeHead(200, {'Content-Type': 'text/html'});
11.     res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');
12.     res.write('<input type="file" name="filetoupload"><br>');
13.     res.write('<input type="submit">');
14.     res.write('</form>');
15.     return res.end(); }
16. }).listen(8080);
```

Cenni tecnici basilari di Node.js prt 10

Quando un file viene caricato correttamente sul server, viene inserito in una cartella temporanea.

Il percorso di questa directory può essere trovato nell'oggetto "files", passato come terzo argomento nella parse() funzione di callback del metodo.

Per spostare il file nella cartella di tua scelta, usa il modulo File System e rinomina il file

```
1. var http = require('http');
2. var formidable = require('formidable');
3. var fs = require('fs');
4. http.createServer(function (req, res) {
5.   if (req.url == '/fileupload') {
6.     var form = new formidable.IncomingForm();
7.     form.parse(req, function (err, fields, files) {
8.       var oldpath = files.fileuploadfilepath;
9.       var newpath = 'C:/Users/Your Name/' + files.fileupload.originalFilename;
10.      fs.rename(oldpath, newpath, function (err) {
11.        if (err) throw err;
12.        res.write('File uploaded and moved!');
13.        res.end();  });
14.    });
15.  } else {
16.    res.writeHead(200, {'Content-Type': 'text/html'});
17.    res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');
18.    res.write('<input type="file" name="fileupload"><br>');
19.    res.write('<input type="submit">');
20.    res.write('</form>');
21.    return res.end();  });
22.  }).listen(8080);
```

Esercitazione 3: Node.js, Html e JS

Cenni tecnici basilari di Node.js prt 11

Il modulo Nodemailer semplifica l'invio di e-mail dal tuo computer o nodo centrale verso altri dispositivi. Andiamo ad installare il modulo Nodemailer utilizzando npm:

```
C:\Users\Your Name>npm install nodemailer
```

Dopo aver scaricato il modulo Nodemailer, puoi includere il modulo in qualsiasi applicazione:

```
var nodemailer = require('nodemailer');
```

Dovremmo sempre utilizzare un nome utente ed una password del provider di posta elettronica selezionato per inviare un'e-mail.

Cenni tecnici basilari di Node.js prt 11

```
1. var nodemailer = require('nodemailer');
2. var transporter = nodemailer.createTransport({
3.   service: 'gmail',
4.   auth: {
5.     user: 'youremail@gmail.com',
6.     pass: 'yourpassword'} });
7. var mailOptions = {
8.   from: 'youremail@gmail.com',
9.   to: 'myfriend@yahoo.com', //campari.mirko@gmail.com
10.  subject: 'Sending Email using Node.js',
11.  text: 'That was easy!'};
12.
13. transporter.sendMail(mailOptions, function(error, info){
14.  if (error) {
15.    console.log(error);
16.  } else {
17.    console.log('Email sent: ' + info.response);}
18.});
```

Cenni tecnici basilari di Node.js prt 11

I Ricevitori multipli servono per inviare un'e-mail a più di un destinatario, aggiungili alla proprietà "to" dell'oggetto mailOptions, separati da virgole:

```
var mailOptions = {  
  from: 'youremail@gmail.com',  
  to: 'myfriend@yahoo.com, myotherfriend@yahoo.com',  
  subject: 'Sending Email using Node.js',  
  text: 'That was easy!' }
```

Per invece inviare del testo in formato HTML nella tua email, usa la proprietà "html" invece della proprietà "text":

```
var mailOptions = {  
  from: 'youremail@gmail.com',  
  to: 'myfriend@yahoo.com',  
  subject: 'Sending Email using Node.js',  
  html: '<h1>Welcome</h1><p>That was easy!</p>'}
```

Cenni tecnici basilari di Node.js prt 12

Node.js può essere utilizzato nelle applicazioni di database.

Uno dei database più popolari è MySQL.

Per poter sperimentare gli esempi di codice che vedremo assieme, dovremo avere MySQL installato sul tuo computer è possibile scaricare un database MySQL gratuito all'indirizzo

[https://www.mysql.com/downloads/ .](https://www.mysql.com/downloads/)

Per installare il modulo "mysql", apri il terminale di comando ed esegui quanto segue:

C:\Users\Your Name>npm install mysql

Node.js può utilizzare questo modulo per manipolare il database MySQL:

var mysql = require('mysql');

Cenni tecnici basilari di Node.js prt 12

Iniziamo creando una connessione al database.

Usiamo il nome utente e la password del nostro database MySQL.

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword" } );

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!"); } );
```

Salviamo il codice sopra in un file chiamato "demo_db_connection.js" ed esegui il file:

C:\Users\Your Name>node demo_db_connection.js

Cenni tecnici basilari di Node.js prt 12

Usa le istruzioni SQL per leggere da o scrivere su un database MySQL.

Questo è anche chiamato "interrogare" il database.

L'oggetto connessione ha un metodo nativo per interrogare il database:

```
con.connect(function(err) {  
  if (err) throw err;  
  console.log("Connected!");  
  con.query(sql, function (err, result) {  
    if (err) throw err;  
    console.log("Result: " + result); })  
});
```

Il metodo query accetta un'istruzione sql come parametro e restituisce il risultato tramite la tipizzazione di ritorno.

Cenni tecnici basilari di Node.js prt 12

Per creare un database in MySQL, usa l'istruzione "CREATE DATABASE":

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword" });

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  con.query("CREATE DATABASE mydb", function (err, result) {
    if (err) throw err;
    console.log("Database created");  });
}); // ALLA FINE C:\Users\Your Name>node NOMEVOSTROFILE.js
```

Cenni tecnici basilari di Node.js prt 12

Per creare un database in MySQL, usa l'istruzione "CREATE DATABASE":

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb" });

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table created"); });
}); // ALLA FINE C:\Users\Your Name>node NOMEVOSTROFILE.js
```

Esercitazione FINALE: Node.js, Html e JS

Registrazione:

Creare una form html che prenda almeno 4 parametri e che li salvi (nome, pass, email, data di nascita)

Login:

per accedere ai propri dati bisognerà conoscere almeno due parametri (password e nome) e gestire l'errore

Action:

se l'accesso va a buon fine far caricare un file (formidable)

Ivl 2: utilizzare la metodologia ad oggetti isolati tramite un crud (anche non completo)

Ivl 3: far andare a salvare richiamare i dati sul nostro DB (mysql)

Cenni tecnici basilari di TypeScript

Esistono tre primitive principali in JavaScript e TypeScript.

- **boolean**- **valori veri o falsi**
- **number**- **numeri interi e valori in virgola mobile**
- **string**- **valori di testo come "TypeScript Rocks"**

Quando si crea una variabile, ci sono due modi principali di assegnazione:

- **Esplicito**

```
let Nome: string = "Mirko";
```

- **Implicito**

```
let firstName = "Dylan";
```

Cenni tecnici basilari di TypeScript

Oltre ai tipi primitivi TypeScript permette di usare dei "tipi speciali" che potrebbero non fare riferimento a nessun tipo specifico di dati.

***any* è un tipo che disabilita il controllo del type safe e consente effettivamente l'utilizzo di tutti i tipi.**

- **let m: any = true;**
- **m = "string"; // no error**
- **Math.round(m); // no error**

Cenni tecnici basilari di TypeScript

Any è molto potente e può essere un modo utile per superare gli errori poiché disabilita il controllo del tipo, ma TypeScript non sarà in grado di fornire la sicurezza del tipo e gli strumenti che si basano sui dati del tipo, come il completamento automatico, non funzioneranno.

Un altro "tipo speciale" è unknown che infatti è un'alternativa simile, ma più sicura a any. TypeScript impedirà unknown l'utilizzo dei tipi

**unknown può essere utilizzato al meglio quando non si conosce il tipo di dati
Per aggiungere un tipo in un secondo momento, il casting è quando usiamo la parola chiave "as" per dire che la proprietà o la variabile è del tipo cast.**

Cenni tecnici basilari di TypeScript

```
let w: unknown = 1;  
w = "stringa"; // nessun errore  
w = { runANonExistentMethod: () => {  
    console.log("Credo quindi di essere un ");}  
} as{ runANonExistentMethod: () => void}
```

Possiamo evitare l'errore per il codice utilizzando il nostro nuovo tipo speciale

```
if(typeof w === 'oggetto' && w !== null) {  
    (w as { runANonExistentMethod: Function }). runANonExistentMethod();}
```

Anche se lo dobbiamo lanciare più volte possiamo fare un controllo nel if per garantire il nostro tipo e avere un casting più sicuro

Cenni tecnici basilari di Typescript

TypeScript ha una sintassi specifica per digitare gli array.

```
const names: string[] = [];
names.push("Mikk"); // no error
// names.push(3);    // Error: Argument of type 'number'
```

La "readonly" invece è la parola chiave che può impedire la modifica degli array.

```
const names: readonly string[] = ["mikk"];
names.push("Jack"); // Error: Property 'push' does not exist on type 'readonly'
```

Tramite il inferenza del tipo TypeScript può dedurre il tipo di un array se ha valori.

```
const numbers = [1, 2, 3]; // inferred to type number[]
numbers.push(4); // no error
numbers.push("2"); // Error:
let head: number = numbers[0]; // no error
```

Cenni tecnici basilari di Typescript

Una tupla è un array tipizzato con una lunghezza e tipi predefiniti per ogni indice.

Le tuple sono molto usate perché consentono a ciascun elemento dell'array di essere un tipizzato tramite di valore noto.

Per definire una tupla, specificare il tipo di ciascun elemento nell'array:

```
// Andiamo a inizializzare la nostra Tupla  
let ourTuple: [number, boolean, string];  
  
// Andiamo a valorizzarla secondo il type pattern corretto  
ourTuple = [50, false, 'qui è tutto ok'];  
  
// Andiamo a valorizzarla secondo il type pattern sbagliato  
ourTuple = [false, 'qui non è tutto ok', 5];
```

L'ordine all'interno dell'inserimento dei dati rispetto al pattern è fondamentale

Cenni tecnici basilari di Typescript

Una buona pratica è creare quando possibile sempre una "tupla readonly" .

Le tuple hanno solo tipi fortemente definiti per i valori iniziali:

```
// definiamo la tupla come una cost readonly  
const our_READONLYTuple: readonly [number, boolean, string] = [5, true, 'The Real  
Coding God'];
```

```
// se proviamo a pushare al suo interno genererà un errore  
our_READONLYTuple.push('Coding God took a day off');
```

Cenni tecnici basilari di Typescript

Un'istruzione if può includere una o più espressioni che restituiscono un valore booleano.

Se l'espressione booleana restituisce true, viene eseguito un insieme di istruzioni.

```
if (true) {  
    console.log('questo verrà eseguito sempre.');//  
}if (false) {  
    console.log('questo non verrà eseguito mai.');//}
```

includiamo ora la tipizzazione e più espressioni booleane nella condizione if

```
let x: number = 10, y = 20;  
if (x < y) {console.log('x è meno di y');//}
```

L'espressione della condizione if $x < y$ viene valutata true e quindi l'app esegue l'istruzione all'interno delle parentesi graffe {}.

Esercitazione di partenza su typescript

Creare un sistema di inserimento in input che inserisca gli elementi in degli array (3 : string, numerici, booleani)

Far sì che quando il numero di inserimenti X sia raggiunto i dati siano divisi per tipizzazione corretta rispetto agli array.

Tipi di speciale in Type Script

Un enum è una "classe" speciale che rappresenta un gruppo di costanti (variabili immutabili).

Gli enum sono disponibili in due versioni stringe numeric. Cominciamo con il numerico.

Per impostazione predefinita, enum inizializzerà il primo valore 0 e aggiungerà 1 a ogni valore aggiuntivo:

```
enum CardinalDirections {  
    North,  
    East,  
    South,  
    West };
```

```
let currentDirection = CardinalDirections.North;  
// North è il primo valore quindi stamperà '0'  
console.log(currentDirection);
```

Tipi di speciale in Type Script

Enum numerici:

Puoi impostare il valore del primo enum numerico e farlo aumentare automaticamente da quello:

```
enum CardinalDirections {  
    North = 1,  
    East,  
    South,  
    West  
}  
// logs 1  
console.log(CardinalDirections.North);  
// logs 4  
console.log(CardinalDirections.West);
```

Tipi di speciale in Type Script

Enum numerici istanziamento e valorizzazione completa :

```
enum StatusCodes {  
    NotFound = 404,  
    Success = 200,  
    Accepted = 202,  
    BadRequest = 400  
};  
  
// logs 404  
console.log(StatusCodes.NotFound);  
  
// logs 200  
console.log(StatusCodes.Success);
```

Tipi di speciale in Type Script

Gli enum possono anche contenere strings. Questo è più comune delle enumerazioni numeriche, a causa della loro leggibilità e finalità.

```
enum CardinalDirections {  
    North = 'North',  
    East = "East",  
    South = "South",  
    West = "West"  
};  
  
// logs "North"  
console.log(CardinalDirections.North);  
  
// logs "West"  
console.log(CardinalDirections.West);
```

Tipi di oggetto in Type Script

TypeScript ha una sintassi specifica per digitare SEMANTICAMENTE gli oggetti.

```
const car: { type: string, model: string, year: number } = {  
    type: "FIAT",  
    model: "500",  
    year: 2011  
};
```

Tipi di oggetto come questo possono essere scritti divisi e riutilizzati

```
const car = {  
    type: "Fiat",};  
  
car.type = "Ford"; // no error  
car.type = 12;     // Error: Type 'number' is not assignable to type 'string'.
```

Tipi di oggetto in Type Script

Le proprietà facoltative sono attributi che non devono essere definite nella definizione dell'oggetto.

Esempio senza una proprietà facoltativa

```
const car: { type: string, mileage: number } = { // Error: Property 'mileage' is missing  
  type: "FIAT" };  
  
car.mileage = 2000;
```

Esempio con una proprietà facoltativa

```
const car: { type: string, mileage?: number } = { // no error  
  type: "FIAT" };  
  
car.mileage = 2000
```

Tipi di oggetto in Type Script

TypeScript consente di definire i tipi separatamente dalle variabili che li utilizzano.

Gli alias e le interfacce consentono di condividere facilmente i tipi tra diverse variabili/oggetti.

Gli alias di tipo consentono di definire i tipi con un nome personalizzato (un alias).

Gli alias di tipo possono essere utilizzati per stringtipi primitivi o più complessi come objectse array

```
1.  
2. type CarYear = number;  
3. type CarType = string;  
4. type CarModel = string;  
5. type Car = {  
6.   year: CarYear,  
7.   type: CarType,  
8.   model: CarModel};  
9. const carYear: CarYear = 2001  
10. const carType: CarType = "Toyota"  
11. const carModel: CarModel = "Corolla"  
12. const car: Car = {  
13.   year: carYear,  
14.   type: carType,  
15.   model: carModel};  
16. console.log(car);
```

Tipi di oggetto in Type Script

Le interfacce sono simili agli alias di tipo, tranne per il fatto che si applicano solo object ai tipi.

```
1. interface Rectangle {  
2.   height: number,  
3.   width: number  
4. };  
5.  
6. const rectangle: Rectangle = {  
7.   height: 20,  
8.   width: 10  
9. };  
10.  
11. console.log(rectangle);
```

Tipi di oggetto in Type Script

Estensione delle interfacce, le interfacce possono estendere la definizione reciproca.
Estendere un'interfaccia significa creare una nuova interfaccia con le stesse proprietà dell'originale, più qualcosa di nuovo.

```
1.interface Rectangle {  
2.   height: number,  
3.   width: number}  
4.interface ColoredRectangle extends Rectangle {  
5.   color: string}  
6.const coloredRectangle: ColoredRectangle = {  
7.   height: 20,  
8.   width: 10,  
9.   color: "red"};  
10.console.log(coloredRectangle);
```

Tipi di oggetto in Type Script

I tipi di unione vengono utilizzati quando un valore può essere più di un singolo tipo. Ad esempio quando una proprietà sarebbe stringo number. Unione | (O)

```
1.function printStatusCode(code: string | number) {  
2.  console.log(`My status code is ${code}.`)  
3.}  
4.  
5.printStatusCode(404);  
6.printStatusCode('404');
```

devi sempre sapere qual è il tuo tipo quando vengono utilizzati i tipi di unione per evitare errori di tipizzazione

Tipi di oggetto in Type Script

Nell'esempio stiamo riscontrando un problema nell'invocare toUpperCase() come string metod e number perché non abbiamo accesso ad esso.

```
function printStatusCode(code: string | number) {  
  console.log(`My status code is ${code.toUpperCase()}.`) // error:  
  Property 'toUpperCase' does not exist on type 'string | number'.  
  Property 'toUpperCase' does not exist on type 'number'  
}
```

Tipi di oggetto in Type Script

Nell'esempio stiamo riscontrando un problema nell'invocare toUpperCase() come string metod e number perché non abbiamo accesso ad esso.

```
function printStatusCode(code: string | number) {  
  console.log(`My status code is ${code.toUpperCase()}.`) // error:  
  Property 'toUpperCase' does not exist on type 'string | number'.  
  Property 'toUpperCase' does not exist on type 'number'  
}
```

Andiamo a creare 3 tipologie di oggetto richiamabili da un menu (fatto con un Enum) che dopo la selezione e l'inserimento della corretta password crei l'oggetto e ti richieda di inserire una serie di informazioni in sequenza (età, email, sei sposato? t/f), è obbligatoriamente in quella sequenza (in una tupla), e che solo nel caso in cui tutti i dati siano inseriti correttamente fa stampare a schermo/console l'insieme del dati e torni all'inizio chiedendoti se ripartire

Prt2: Integrare il richiamo a sistemi node.js(html)

Prt3: Alla fine di far inserire i dati in un file tramite il modulo fs

Funzioni in Type Script

TypeScript ha una sintassi specifica per digitare i parametri della funzione e i valori restituiti ovvero il Tipo di ritorno

Il tipo del valore restituito dalla funzione può essere definito in modo esplicito.

```
function getTime(): number {  
    return new Date().getTime();  
}  
  
console.log(getTime());
```

Se non viene definito alcun tipo restituito, TypeScript tenterà di inferirlo attraverso i tipi delle variabili o delle espressioni restituite.

Funzioni in Type Script

Tipo di reso nullo ovvero Il tipo void che può essere utilizzato per indicare che una funzione non restituisce alcun valore.

```
function printHello(): void {  
    console.log('Hello!');}  
printHello();
```

Funzioni in Type Script

I parametri di funzione sono digitati con una sintassi simile a quella delle dichiarazioni di variabile.

```
function multiply(a: number, b: number) {  
    return a * b;  
}
```

```
console.log(multiply(2,5))
```

Se non è definito alcun tipo di parametro, TS userà per impostazione predefinita any, a meno che non siano disponibili informazioni sul tipo

Funzioni in Type Script

Parametri predefiniti Per i parametri con valori predefiniti, il valore predefinito va dopo l'annotazione del tipo:

```
function pow(value: number, exponent: number = 10) {  
    return value ** exponent;  
}  
  
console.log(pow(10));
```

Funzioni in Type Script

Parametri facoltativi Per impostazione predefinita, TypeScript presupporrà che tutti i parametri siano obbligatori, ma possono essere esplicitamente contrassegnati come facoltativi.

```
function add(a: number, b: number, c?: number) {  
    return a + b + (c || 0);  
}  
  
console.log(add(2,5))
```

TypeScript può anche dedurre il tipo dal valore predefinito.

Funzioni in Type Script

Parametri con nome La digitazione dei parametri denominati segue lo stesso schema della digitazione dei parametri normali.

```
function divide({ dividend, divisor }: { dividend: number, divisor: number })
{
    return dividend / divisor;
}

console.log(divide({dividend: 10, divisor: 2}));
```

Funzioni in Type Script

Parametri di REST I parametri rest possono essere digitati come parametri normali, ma il tipo deve essere un array poiché i parametri rest sono sempre array.

```
function add(a: number, b: number, ...rest: number[]) {  
    return a + b + rest.reduce((p, c) => p + c, 0);  
}  
  
console.log(add(10,10,10,10,10));
```

Funzioni in Type Script

Digitare Alias I tipi di funzione possono essere specificati separatamente dalle funzioni con alias di tipo.

```
type Negate = (value: number) => number;  
const negateFunction: Negate = (value) => value * -1;
```

```
console.log(negateFunction(10));
```

Casting in Type Script

Ci sono momenti in cui si lavora con i tipi in cui è necessario sovrascrivere il tipo di una variabile, ad esempio quando i tipi non corretti vengono forniti da una libreria. Il casting è il processo di override di un tipo. Un modo semplice per lanciare una variabile è usare la asparola chiave, che cambierà direttamente il tipo della variabile data.

```
let x: unknown = 'hello';
```

```
console.log((x as string).length);
```

Casting in Type Script

Il cast in realtà non modifica il tipo di dati all'interno della variabile, ad esempio il codice seguente non funziona come previsto poiché la variabile x contiene ancora un numero

```
let x: unknown = 4;  
  
console.log((x as string).length); //
```

TypeScript tenterà comunque di controllare i cast per evitare cast che non sembrano corretti, ad esempio quanto segue genererà un errore di tipo poiché TypeScript sa che il casting di una stringa su un numero non ha senso senza convertire i dati:

```
console.log((4 as string).length);
```

Casting in Type Script

L'uso di <> funziona come il casting con as.

```
let x: unknown = 'hello';
console.log((<string>x).length);
```

Per eseguire l'override degli errori di tipo che TypeScript può generare durante il cast, eseguire prima il cast su unknown, quindi sul tipo di destinazione.

```
let x = 'hello';
console.log(((x as unknown) as number).length);
```

Typing in Type Script

TypeScript viene fornito con un gran numero di tipi che possono aiutare con alcune manipolazioni di tipo comuni, solitamente indicate come tipi di utilità.

Partial modifica tutte le proprietà di un oggetto in modo che siano facoltative.

```
interface Point {  
    x: number;  
    y: number;}  
let pointPart: Partial<Point> = {} // `Partial` allows x and y to be optional  
pointPart.x = 10;  
console.log(pointPart);
```

Typing in Type Script

Required cambia tutte le proprietà di un oggetto in obbligatorie.

```
interface Car {  
  make: string;  
  model: string;  
  mileage?: number; }
```

```
let myCar: Required<Car> = {  
  make: 'Ford',  
  model: 'Focus',  
  mileage: 12000 // `Required` forces mileage to be defined };
```

Comporre una struttura di interfacce utente che permettano di:

Fare login o iscriversi per due tipi di utente (utente normale , admin [admin dovrà avere x metodo speciale per confermare la creazione es: password])

Permettere dopo il login di, scrivere un file /(fs module) o modificarne uno se c'è ne già creati (la modifica è liberamente ottimizzabile, dal contenuto al nome del file singolo vanno bene uguali tutte le soluzioni nel mezzo)

Permettere di modificare i propri dati utente aggiungendo elementi che non sono OBBLIGATORI durante l'iscrizione

Permettere di fare logout e ripetere il ciclo

Obblighi: voglio che usiate I tipizzazione e che la creazione file sia gestita o internamente a ts oppure tramite node e gli oggetti per definire i due tipi di utenti possibili

PRT2: aggiungere formidable alle azioni after login e permettere il caricamento di un file

PRT3: far sì che si possa cercare per nome gli utenti già esistenti come oggetto se si è admin

Un piccolo pezzo di OOP. Cosa e perchè?

Ereditarietà è una caratteristica che permette ad una classe di estendere le proprietà di altre classi.

Polimorfismo è la possibilità di richiamare su vari oggetti uno stesso metodo che agisce in modo diverso in base al tipo di oggetto su cui è richiamato.

Incapsulamento può essere usato per riferirsi a due concetti, collegati tra loro ma distinti :

- un meccanismo del linguaggio di programmazione atto a limitare l'accesso diretto agli elementi dell'oggetto;
 - un costrutto del linguaggio di programmazione che favorisce l'integrazione dei metodi (o di altre funzioni) propri della classe all'interno della classe stessa.
-

Un piccolo pezzo di OOP. Cosa e perchè?

La programmazione ad oggetti in JS o TS si basa sul modello prototipale che si differenzia da quello di altri linguaggi come Java o PHP.

In Javascript non esiste il concetto di classe. Vengono sfruttate le funzioni (Function constructor) per simulare il modello di programmazione a oggetti classico. Javascript supporta il meccanismo dell'ereditarietà attraverso l'uso degli oggetti prototype.

Ogni oggetto ha al suo interno una proprietà prototype che punta a un altro oggetto il quale contiene a sua volta una proprietà prototype e così via fino a raggiungere un oggetto base. In questo modo si viene a creare quella che prende il nome di Prototype Chain (catena di prototipi)..

Esempio di Ereditarietà in JS

```
1. class Animal {  
2.   constructor(name) {  
3.     this.speed = 0;  
4.     this.nome= nome;  
5.   }  
6.   run(speed) {  
7.     this.speed = speed;  
8.     alert(`${this.nome} corre alla velocità di ${this.speed}.`);  
9.   }  
10.  stop() {  
11.    this.speed = 0;  
12.    alert(`${this.nome} sta fermo.`);  
13.  }  
14.}  
15.let animal = new Animal("cane bau");
```

Esempio di Ereditarietà in JS

Andiamo ora a creare un'altra classe ovvero **Coniglio**, la classe Coniglio sarà quindi essere basata, o più tecnicamente estesa basandosi, su **Animal**, avendo accesso a tutti i metodi di **Animal**, in questo modo Rabbit può assumere tutti i comportamenti di base di un **Animal**. La sintassi utilizzate per estendere un'altra classe è: class Child extends Parent.

```
1.class Coniglio extends Animal {  
2.    hide() {  
3.        alert(`${this.name} Si nasconde!`);  
4.    }  
5.}  
6.  
7.let rabbit = new Coniglio("White Rabbit");  
8.  
9.Coniglio.run(5); // Il nostro Coniglio correrà con speed 5.  
10.  
11.Coniglio.hide(); // il nostro coniglio userà il suo metodo unico
```

Esercitazione di partenza java

**Creare un sistema crud su un oggetto Utente
che possa essere richiamato da un punto
centrale e che stampi in console i dati dell
utente se richiesto**

Cos'è una Command Line Interface (CLI).

Una Command Line Interface (CLI) è un'interfaccia a riga di comando, ovvero un'interfaccia basata su comandi testuali per eseguire azioni su un sistema informatico. Esistono molti tipi diversi di interfacce a riga di comando, ma le due più popolari sono quelle dei sistemi operativi Windows (DOS) e Linux e OS (shell bash).

Agli inizi dell'informatica, la CLI era l'interfaccia standard delle interazioni utente / computer. Con il successivo sviluppo tecnologico, le CLI vennero sostituite per lo più da GUI, interfacce grafiche che sostituiscono i comandi testuali con elementi grafici come puntatori, pulsanti e icone.

Istalliamo angular!

Angular CLI (Command Line Interface). Tramite questi comandi, potremmo creare singoli elementi del framework Angular senza dover scrivere decine di righe di codice per ognuno. Per dare un'occhiata alla versione attuale e a eventuali operazioni da fare per aggiornare eventuali progetti Angular datati, collegati al link ufficiale su github qui: [Angular-CLI](#).

Per installare Angular CLI, dovrai scrivere questa riga all'interno della finestra terminale:

```
npm install -g @angular/cli
```

```
C:\Users\videocorsi>node -v  
v6.9.1
```

```
C:\Users\videocorsi>npm -v  
4.1.1
```

```
C:\Users\videocorsi>npm install -g angular-cli
```

Istalliamo angular!

Per poter confezionare l'insieme dei file necessari ad entrare nel vivo dello sviluppo della tua prima applicazione, possiamo sfruttare il comando qui sotto, che ci permetterà di creare un nuovo progetto con un nome specifico, nel nostro caso "miaprima-app". Chiaramente sarà creata nella stessa cartella in cui sei posizionato attualmente, quindi per cambiarla dovrai usare i classici comandi DOS, cd nomecartella per entrare in una sottocartella, e cd.. per spostarti nella cartella padre.

ng new miaprime-app

Come nome dell'applicazione, devi sempre partire con lettere minuscole, non inserire spazi, caratteri alfanumeri, simbolo di underscore (trattino basso)

Nell'esempio qui sotto la cartella testAngular nel mio desktop, mi sono spostato all'interno, e poi ho creato il primo progetto Angular dal nome miaprime-app:

```
C:\Users\videocorsi>cd Desktop
```

```
C:\Users\videocorsi\Desktop>cd testAngular
```

```
C:\Users\videocorsi\Desktop\testAngular>ng new miaprime-app
```

Istalliamo angular!

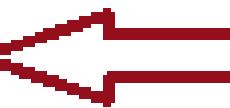
Per eseguire un'applicazione Angular nel browser, devi prima crearla.

Puoi creare ed eseguire applicazioni utilizzando il comando Angular CLI o il comando NPM. È possibile utilizzare il terminale/finestra di comando per eseguire questi comandi. Tuttavia, se stai utilizzando VS Code, puoi eseguire i comandi dal suo terminale. Utilizzare il comando CLI angolare *ng serve -o* per creare un'applicazione.

Il -o indica di aprirlo automaticamente nel browser predefinito.

Utilizzare il comando NPM *npm start* per creare un'applicazione. Internamente, utilizza solo il comando *ng serve*. Aprire un browser e passare a <http://localhost:4200> per visualizzare la home page dell'applicazione. <http://localhost:4200> è l'URL predefinito di un'applicazione Angular creata utilizzando Angular CLI.

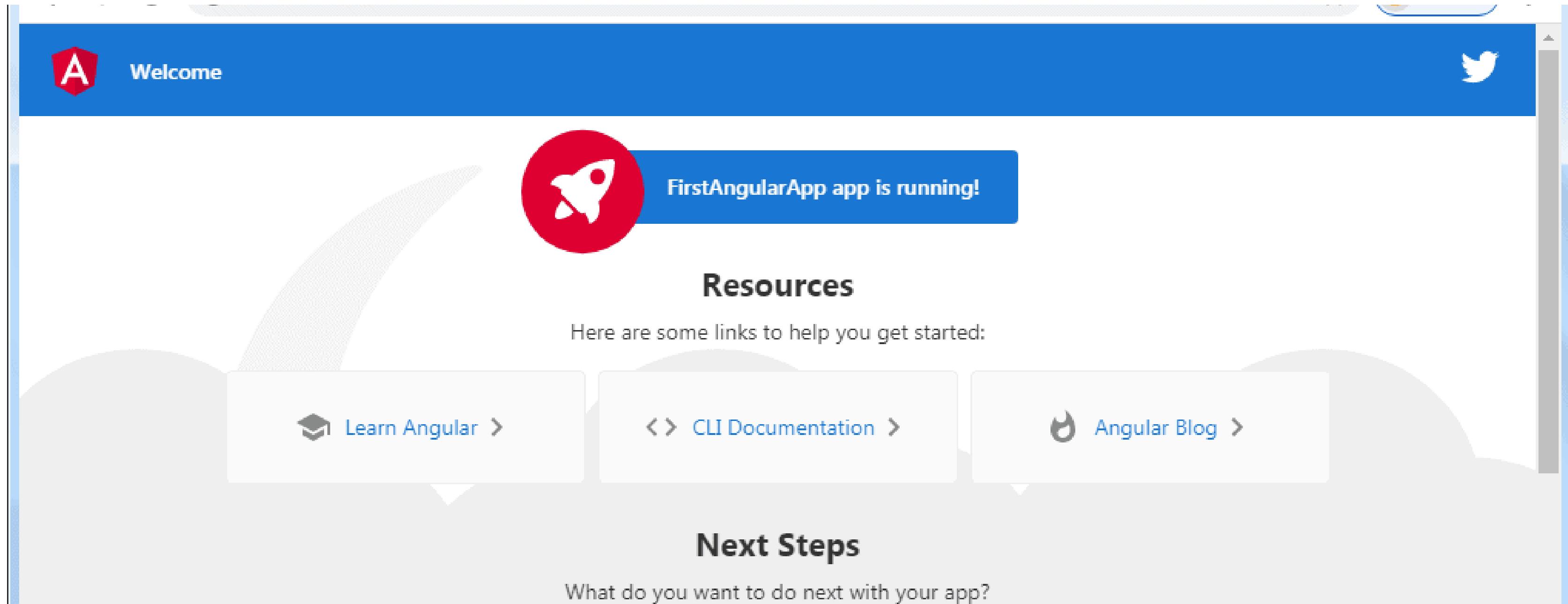
Apri il terminale in VS Code dal menu Terminale -> Nuovo terminale e digita *ng serve -o* command e premi Invio, come mostrato di seguito.

D:\AngularApps\FirstAngularApp>*ng serve -o* 

```
chunk {main} main.js, main.js.map (main) 60.6 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 141 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
```

Istalliamo angular!

Il ng serve comando crea l'app, avvia il server di sviluppo, controlla i file di origine e ricostruisce l'app mentre apporti modifiche a quei file. Aprirà la tua applicazione Angular nel browser predefinito, come mostrato di seguito.



Istalliamo angular!

Il ng serve comando continua a guardare i file sorgente, quindi se apporti modifiche a qualsiasi file del progetto, lo ricostruirà e aggiornerà automaticamente il browser per riflettere le modifiche.

Per interrompere il processo di compilazione automatica, premere Ctrl + c nel terminale di VS Code.

Pertanto, puoi creare uno spazio di lavoro e un progetto angolare iniziale ed eseguirlo utilizzando VS Code.

Angular e i suoi Componenti

Qui imparerai a conoscere il componente Angular e come creare un componente personalizzato utilizzando Angular CLI.

Angular è un framework SPA e una vista è composta da uno o più componenti.

Un componente angolare rappresenta una parte di una vista.

Generalmente, una pagina Web interattiva è composta da HTML, CSS e JavaScript. Il componente ANGULAR non è diverso.

Componente angolare =

Modello HTML + Classe componente + Metadati componente

Angular e i suoi Componenti

Classe: In sostanza, una classe componente è una classe TypeScript che include proprietà e metodi. Le proprietà memorizzano dati e metodi includono la logica per il componente. Alla fine, questa classe verrà compilata in JavaScript.

Nota: TypeScript è un linguaggio open source orientato agli oggetti sviluppato e gestito da Microsoft. È un superset tipizzato di JavaScript che viene compilato in semplice JavaScript.

Metadati: I metadati sono alcuni dati extra per un componente utilizzato dall'API Angular per eseguire il componente, come la posizione dei file HTML e CSS del componente, selettore, fornitori, ecc.

Angular e i suoi Componenti

Puoi creare file per un componente manualmente o utilizzando il comando Angular CLI. La CLI angolare riduce i tempi di sviluppo. Quindi, usiamo Angular CLI per creare un componente. Utilizzare il comando CLI per generare un componente.

ng genera il componente <nome componente>

Tutti i comandi Angular CLI iniziano con ng, generate detto anche g è un comando, component è un argomento e quindi il nome del componente è l'ultimo elemento. Quanto segue esegue il ng comando per generare il componente greet in VS Code.

```
D:\TestProjects\angularapp>ng g component greet ←  
CREATE src/app/greet/greet.component.html (20 bytes)  
CREATE src/app/greet/greet.component.spec.ts (621 bytes)  
CREATE src/app/greet/greet.component.ts (271 bytes)  
CREATE src/app/greet/greet.component.css (0 bytes)
```

Angular e i suoi Componenti

Il comando precedente creerà una nuova cartella "greet" e una cartella dell'app con dentro quattro file, come mostrato di seguito.



Sopra, greet. component.css è un file CSS per il componente, greet.html è un file HTML per il componente in cui scriveremo HTML per un componente, greet .spec.ts è un file di test in cui possiamo scrivere unit test per un componente ed greet.component.ts è il file di classe per un componente.

Angular e i suoi Componenti

Andiamo a vedere la composizione di un component,

```
greet.component.ts
```

Import → import { Component, OnInit } from '@angular/core';

Metadata {
 @Component({
 selector: 'app-greet', ← Component Tag
 templateUrl: './greet.component.html', ← HTML Template File Name and Location
 styleUrls: ['./greet.component.css'] ← CSS File Name and Location
 })
 © TutorialsTeacher.com

Component Class {
 export class GreetComponent implements OnInit {

 constructor() { }

 ngOnInit(): void {
 }
 }
}

Angular e i suoi Componenti

Component Class: GreetComponent è la classe del componente. Contiene proprietà e metodi per interagire con la vista tramite un'API Angular. Implementa l' OnInit interfaccia, che è un hook del ciclo di vita.

Metadati del componente: è @Componentun decoratore utilizzato per specificare i metadati per la classe del componente definita immediatamente sotto di esso. È una funzione e può includere diverse configurazioni per il componente. Indica ad Angular dove ottenere i file richiesti per il componente, creare e rendere il componente. Tutti i componenti Angular devono avere un @Componentdecoratore sopra la classe del componente.

L'istruzione import ottiene la funzionalità richiesta da Angular o da altre librerie. L'importazione ci consente di utilizzare membri esportati da moduli esterni. Ad esempio, @Componentdecoratore e OnInitinterfaccia sono contenuti nella @angular/corelibreria. Quindi, possiamo usarli dopo averlo importato.

Angular e i suoi Componenti

Ora aggiungiamo una proprietà e un metodo nella classe del componente, come mostrato di seguito.

```
constructor() { }
```

```
ngOnInit(): void {  
}
```

```
name: string = "Steve";
```

```
greet(): void {  
    alert("Hello " + this.name);  
};  
}
```

Angular e i suoi Componenti

Sopra, abbiamo aggiunto la name proprietà e il greet metodo nella classe del componente. Usiamo questi nel modello HTML.

Apri greet.component.html, rimuovi il codice esistente e aggiungi il codice.

greet.component.ts

```
<div>
  Enter Your Name: <input type="text" value="{{name}} /> <br />
  <button (click)="greet()">Greet Me!</button>
</div>
```

Nel modello HTML sopra, abbiamo utilizzato la proprietà name nell'interpolazione {{ }} per visualizzare il suo valore e greet() funzionare come evento clic.

Angular e i suoi Componenti

Ora è il momento di caricare il nostro componente, ma prima dobbiamo ospitare la nostra applicazione e caricare il componente root. Questo processo è chiamato bootstrap.

Quindi, dobbiamo ospitare la nostra applicazione in index.html, quindi dobbiamo definire un modulo root per eseguire il bootstrap del nostro componente root. Index.html sarà l'unica pagina web in un'applicazione Angular, ed è per questo che si chiama SPA.

Quando generi un'applicazione Angular utilizzando la CLI Angular, crea automaticamente index.html, root component app.component.ts, root module app.module.ts e template HTML app.component.html per te. La AppComponent classe in app.component.ts è un componente root e la AppModule classe in app.module.ts è un modulo root.

Angular e i suoi Componenti

Qui, caricheremo il nostro componente greet nel componente root in due passaggi. **1. Dichiare un componente nel modulo root.**

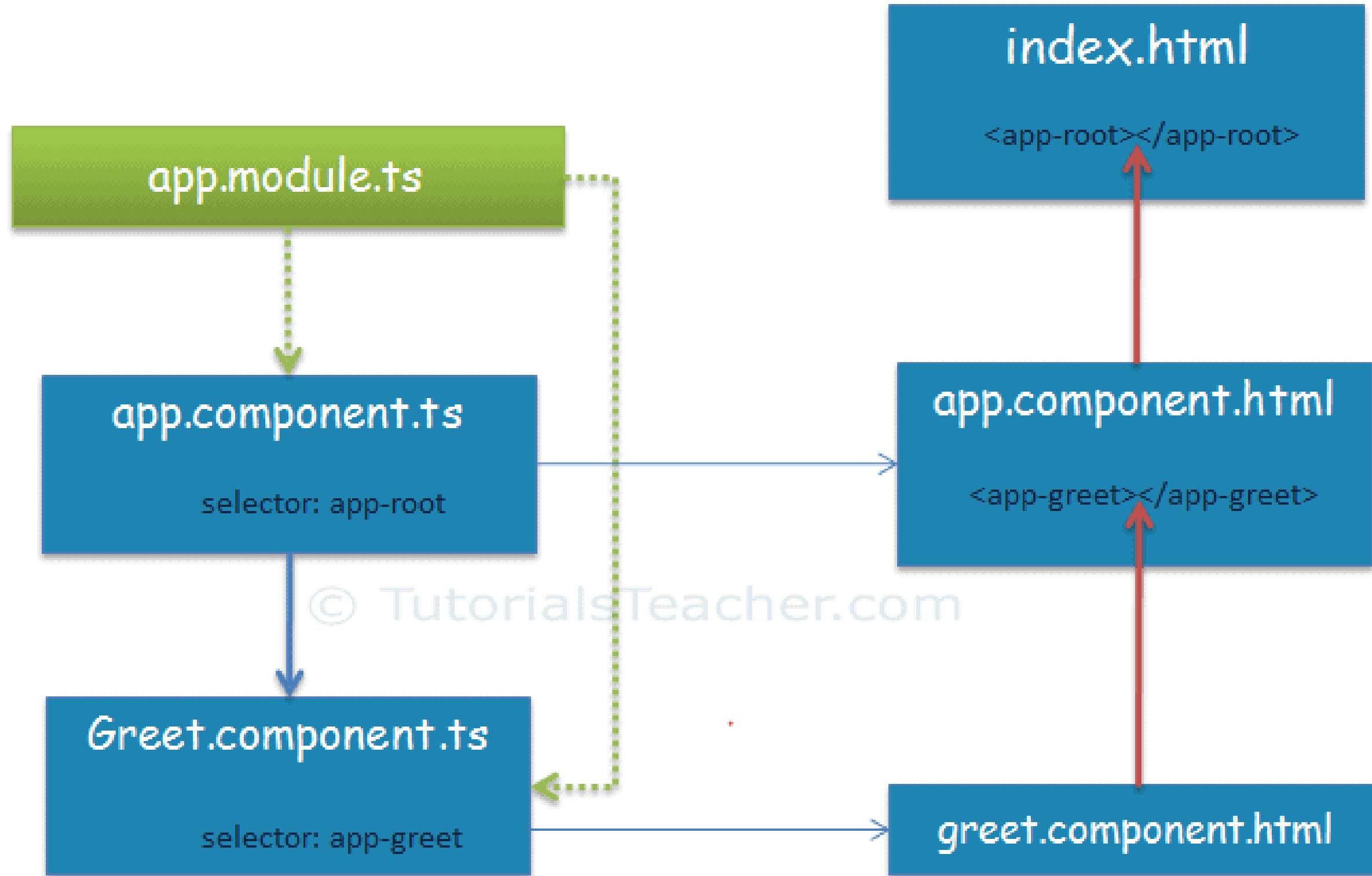
```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { AppRoutingModule } from './app-routing.module';
4. import { AppComponent } from './app.component';
5. import { GreetComponent } from './greet/greet.component'; //import GreetComponent
6. @NgModule({
7.   declarations: [
8.     AppComponent,
9.     GreetComponent // <- include GreetComponent in declarations],
10.   imports: [
11.     BrowserModule,
12.     AppRoutingModule
13.   ],
14.   providers: [],
15.   bootstrap: [AppComponent]
16. })
17. export class AppModule { }
```

Angular e i suoi Componenti

2. Aggiungere il tag del componente nel modello HTML radice.

Dopo aver aggiunto una dichiarazione di componente, utilizzare il tag del componente `<app-greet></app-greet>` nel file HTML del componente principale, che è `app.component.html`, come mostrato di seguito.

```
<div>
  <app-greet></app-greet>
</div>
```



Angular e i suoi Componenti

Possiamo anche creare un singolo file componente greet.component.ts se il codice HTML di un componente è inferiore. Utilizzare il parametro template nel @Component decoratore per includere l'HTML del componente. Il seguente greetcomponent fornisce lo stesso risultato di cui sopra.

```
1. import { Component } from '@angular/core';
2. @Component({
3.   selector: "app-greet",
4.   template: `<div>
5.     Enter Your Name: <input type="text" value="{{name}} /> <br/>
6.     <button (click)="greet()">Greet Me!</button>
7.   </div>` })
8.
9. export class GreetComponent {
10.   name: string = "Steve";
11.   greet(): void {
12.     alert("Hello " + this.name); };}
```

Angular e i suoi Componenti

Il modello HTML non è altro che un normale codice HTML con una sintassi aggiuntiva specifica per Angular per comunicare con la classe del componente.

Modello HTML = HTML + Collegamenti e direttive angular.

L'API angular interpreta un modello HTML di un componente, genera l'HTML e lo rende.

Puoi creare un modello HTML in un componente in due modi:

1. Modello in linea

2. Modello collegato

Modello Lineare

Un modello HTML in linea per un componente viene definito utilizzando la configurazione del modello nel @Component decoratore.

Può essere un modello a riga singola racchiuso tra virgolette doppie o virgolette singole.

```
@Component({  
  selector: "app-greet",  
  template: "Enter Your Name: <input value={{name}} />"  
})
```

Può anche essere un modello multilinea racchiuso all'interno di apici inversi char `.

```
@Component({  
  selector: "app-greet",  
  template: `<div>  
    Enter Your Name: <input type="text" value={{name}} /> <br/>  
    <button (click)="greet()">Greet Me!</button>  
  </div>`  
})
```

Modello Per collegamento

Un componente può avere un file HTML separato per includere un modello HTML di un componente.

Utilizziamo il template Url come parametro per dichiarare il percorso del file modello HTML, come mostrato di seguito.

```
@Component({  
  selector: "app-greet",  
  templateUrl: "./mycomponent.component.html"})
```

È consigliabile disporre di un file .html separato per un modello. Sarà facile lavorare con i tag HTML e mantenerli.

Un componente può anche utilizzare il file SVG come file modello.

```
@Component({  
  selector: 'app-svg',  
  templateUrl: './draw.component.svg',  
  styleUrls: ['./draw.component.css']})
```

Event Binding in Angular

Gli eventi vengono gestiti in Angular utilizzando la seguente sintassi speciale.

(target event name) = "template statement"

Associa il nome dell'evento di destinazione tra parentesi a sinistra di un segno di uguale e il metodo o l'istruzione del gestore eventi a destra.

<button (click)="onShow()">Show</button>

Sopra, (click) associa l'evento clic del pulsante e onShow() l'istruzione chiama il onShow() cioè un metodo di un componente.

```
@Component({  
  selector: 'event-demo',  
  template: '<button (click)="onShow()">Show</button>'})
```

```
export class EventBindingDemoComponent implements OnInit {  
  constructor() {}  
  ngOnInit(): void {}  
  onShow() {  
    alert('Show button clicked!');}  
}
```

Data binding bidirezionale in Angular

Il data binding bidirezionale si riferisce alla condivisione dei dati tra una classe di componenti e il relativo modello.

**Se modifichi i dati in un punto, verranno riformulati automaticamente all'altra estremità.
Ad esempio, se modifichi il valore della casella di input, aggiornerà anche il valore della proprietà associata in una classe di componenti.**

L'associazione dati bidirezionale esegue le seguenti azioni:

1. Imposta una proprietà di una classe di componenti

2. Ascolta un evento di modifica dell'elemento DOM

**Angular dalla v2+ supporta il data binding bidirezionale utilizzando la direttiva ngModel
oltreche essere gestibile anche con metodi getter e setter.**

Data binding bidirezionale in Angular

La ngModel è una direttiva con [()] come sintassi (nota anche come sintassi banana box) sincronizza i valori dall'interfaccia utente a una proprietà e viceversa. Pertanto, ogni volta che l'utente modifica il valore sull'interfaccia utente, il valore della proprietà corrispondente verrà aggiornato automaticamente.

[()] = [] + () = dove [] lega l'attributo e () lega un evento.

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-greet',
  template: `
    User Name: <input type="text" [(ngModel)]="userName" ><br/>
    {{userName}}
  `})
export class GreetComponent implements OnInit {
  constructor() { }
  userName: string = "jbond";
  ngOnInit(): void { }
```

La [(ngModel)] come sintassi è il metodo consigliato per l'associazione dati bidirezionale.

La ngModel con solo [] è la sintassi che viene utilizzata per l'associazione dati unidirezionale.

[ngModel] associa un valore a una proprietà e al controllo dell'interfaccia utente.

Architettura monolitica vs. architettura di microservizi

Nelle architetture monolitiche tutti i processi sono strettamente collegati tra loro e vengono eseguiti come un singolo servizio.

Ciò significa che se un processo dell'applicazione sperimenta un picco nella richiesta, è necessario ridimensionare l'intera architettura. Aggiungere o migliorare una funzionalità dell'applicazione monolitica diventa più complesso, in quanto sarà necessario aumentare la base di codice. Tale complessità limita la sperimentazione e rende più difficile implementare nuove idee.

Le architetture monolitiche rappresentano un ulteriore rischio per la disponibilità dell'applicazione, poiché la presenza di numerosi processi dipendenti e strettamente collegati aumenta l'impatto di un errore in un singolo processo.

Architettura monolitica vs. architettura di microservizi

- **Architettura monolitica:** è la diretta conseguenza dell'era dei mainframe IBM e del periodo di monopolio del sistema operativo Microsoft Windows, tradizionalmente adottata negli ambienti IT aziendali.
- **Microservizi:** inizialmente originati da community Open Source, sviluppatori di terze parti e start-up, dove programmati indipendenti hanno messo a disposizione codici destinati ad estendere la funzionalità di base delle piattaforme di server web più diffuse. Ora, la maggior parte delle aziende IT più importanti rilascia i propri microservizi e i propri contributi ai progetti Open Source dove l'adozione degli standard è estesa a una varietà di mercati verticali e team di diversi settori sulla base di elementi fondamentali univoci. I microservizi si basano sullo stesso principio di innovazione degli sviluppatori tramite soluzioni di codice Open Source per applicazioni cloud, sebbene attualmente siano comuni anche i microservizi con licenza proprietaria.

Architettura monolitica vs. architettura di microservizi

- **Innovazione rapida:** aziende e start-up possono immettere sul mercato soluzioni innovative più rapidamente rispetto a quanto sia possibile fare con un'architettura monolitica quando occorre creare nuove funzionalità per le applicazioni software. I clienti che utilizzano applicazioni web e mobili richiedono nuove caratteristiche. Le tecnologie innovative ricevono fondi attraverso l'utilizzo diffuso da parte degli utenti e l'adozione da parte delle aziende. Rimanere all'avanguardia sul fronte programmazione e sviluppo mediante l'integrazione di microservizi comporta vantaggi sia per le aziende IT che per le start-up.
- **Livelli superiori di automazione del data center:** gli sviluppatori preferiscono usare determinate piattaforme o determinati standard per il loro lavoro, incluso il supporto dei linguaggi di programmazione e dei database nelle applicazioni web/mobili con i microservizi. La connessione dei microservizi avviene mediante processi di scripting, ad esempio le API, che possono incrementare i livelli di automazione del data center.

Architettura monolitica vs. architettura di microservizi

- **Autonomi:** Ciascun servizio nell'architettura basata su microservizi può essere sviluppato, distribuito, eseguito e ridimensionato senza influenzare il funzionamento degli altri componenti. I servizi non devono condividere alcun codice o implementazione con gli altri. Qualsiasi comunicazione tra i componenti individuali avviene attraverso API ben definite.
- **Specializzati:** Ciascun servizio è progettato per una serie di capacità e si concentra sulla risoluzione di un problema specifico. Se, nel tempo, gli sviluppatori aggiungono del codice aggiuntivo a un servizio rendendolo più complesso, il servizio può essere scomposto in servizi più piccoli.

Architettura monolitica vs. architettura di microservizi

- Possiamo declinare i microservizi anche a partire da una predefinita metodologia di strutturazione a servizi logici detta distribuita ma lo vedremo nelle prossime lezioni per ora andiamo ad approfondire l'argomento guardando del codice pratico ed esercitandoci sull'isolamento delle funzioni e nella distinzioni delle condizioni di Isolabilità DI UN BLOCCO DI CODICE
- <https://github.com/MaSTERmIKK/microserviceFORU>
- <https://github.com/MaSTERmIKK/microservices-demoFORU>
- <https://github.com/MaSTERmIKK/microserviceFORU2>

Architetture REST e realizzazione di RESTFUL API

Prima di parlare delle architetture REST dobbiamo partire da alcuni basi comuni.

API è l'acronimo di *Application Programming Interface*.

È il modo con cui componenti *software* stabiliscono regole di comunicazione, un'interfaccia che consente il dialogo tra parti diverse di uno stesso *software* o tra parti di programmi diversi, nascono con lo scopo di permettere allo sviluppatore di usare lo stesso codice in diversi contesti e aiutano l'interoperabilità semantica.

- **Ci sono API che permettono al PC di eseguire le attività richieste chiamando le varie librerie di sistema**
 - **Ci sono le API di Facebook, Twitter, Instagram, Google dette API Oauth e utilizzate per l'autenticazione**
 - **Ci sono le API di Ebay, Amazon e dei vari e-commerce e altre ancora.**
-

Architetture REST e realizzazione di RESTFUL API

- **Representational State Transfer (REST) è un'architettura software che impone condizioni sul funzionamento di un'API. REST è stata inizialmente creata come linea guida per la gestione delle comunicazioni in una rete complessa come internet. Si può utilizzare l'architettura REST per supportare una comunicazione su vasta scala ad elevate prestazioni e affidabile. Si può facilmente implementare e modificare, apportando visibilità e portabilità multipiattaforma a qualsiasi sistema API.**
- **Gli sviluppatori API possono progettare API utilizzando diverse architetture. Le API che seguono lo stile architettonico REST sono chiamate API REST. I servizi Web che implementano le architetture REST sono chiamati servizi Web RESTful. Il termine RESTful API si riferisce generalmente alle API web RESTful. Comunque, è possibile utilizzare i termini REST API e RESTful API in maniera interscambiabile.**

Architetture REST e realizzazione di RESTFUL API

La funzione di base di una API RESTful è la stessa della navigazione su internet. Il client contatta il server utilizzando l'API quando richiede una risorsa. Gli sviluppatori API spiegano in che modo il client dovrebbe utilizzare l'API REST nella documentazione API dell'applicazione del server. Di seguito i passaggi generali per qualsiasi chiamata REST API:

- Il client invia una richiesta al server. Il client segue la documentazione API per formattare la richiesta in modo che il server comprenda.**
- Il server autentica il client e conferma che ha il diritto di effettuare la richiesta.**
- Il server riceve la richiesta e la elabora internamente.**
- Il server risponde al client. La risposta contiene informazioni che rivelano al client se la richiesta è avvenuta correttamente, la risposta include inoltre qualsiasi informazione richiesta dal client la dove le condizioni di accesso siano verificate e verificabili.**

La richiesta API REST e i dettagli della risposta variano leggermente in base a come gli sviluppatori progettano l'API.

Architetture REST e realizzazione di RESTFUL API

cos'è un sistema distribuito?

La locuzione sistema distribuito, in informatica, indica genericamente una tipologia di sistema informatico costituito da un insieme di processi interconnessi tra loro in cui le comunicazioni avvengono solo esclusivamente tramite lo scambio di opportuni messaggi. Ogni nodo del sistema esegue un insieme di componenti che comunicano tra di loro utilizzando uno strato software detto middleware che permette all'utente di percepire il sistema come un'unica entità. Con il termine processo si indica, in genere, una qualsiasi entità capace di comunicare con un qualsiasi altro processo e di eseguire un algoritmo distribuito. A differenza di un algoritmo tradizionale è necessario includere nella definizione di algoritmo distribuito anche i messaggi che vengono scambiati tra i vari processi, poiché anch'essi sono essenziali nell'esecuzione e nella terminazione dell'algoritmo. I sistemi distribuiti nascono da esigenze sia di tipo economico che tecnologico.

Architetture REST e realizzazione di RESTFUL API

cos'è un sistema centralizzato?

Si parla di sistema informatico centralizzato quando i dati e le applicazioni risiedono in un unico nodo elaborativo.

Sistema informatico centralizzato

Viceversa, si parla di sistema informatico distribuito quando almeno una delle seguenti due condizioni è verificata:

- **le applicazioni, fra loro cooperanti, risiedono su più nodi elaborativi (elaborazione distribuita);**
- **il patrimonio informativo, unitario, è ospitato su più nodi elaborativi (base di dati distribuita).**

In termini generali, quindi, un sistema distribuito è costituito da un insieme di applicazioni logicamente indipendenti che collaborano per il perseguimento di obiettivi comuni attraverso una infrastruttura di comunicazione hardware e software

Architetture REST e realizzazione di RESTFUL API

