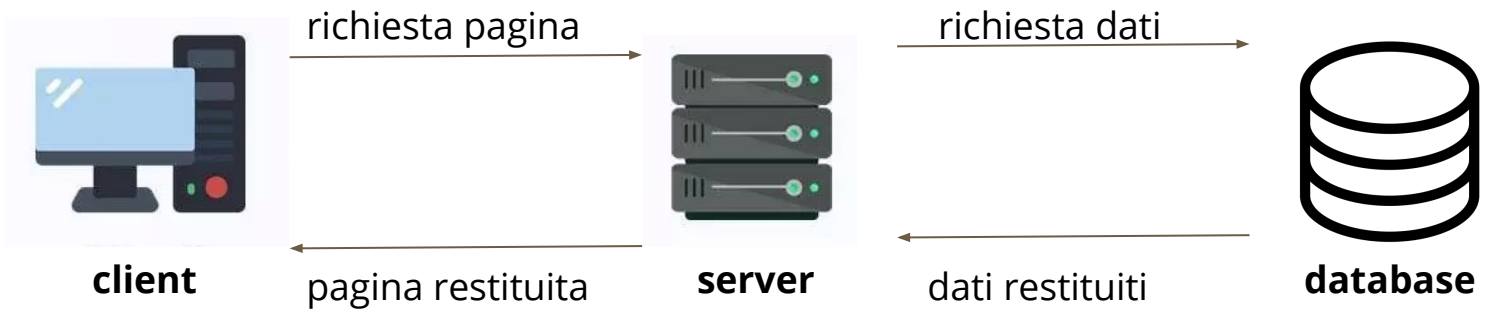
Come creare un web server con NodeJS?

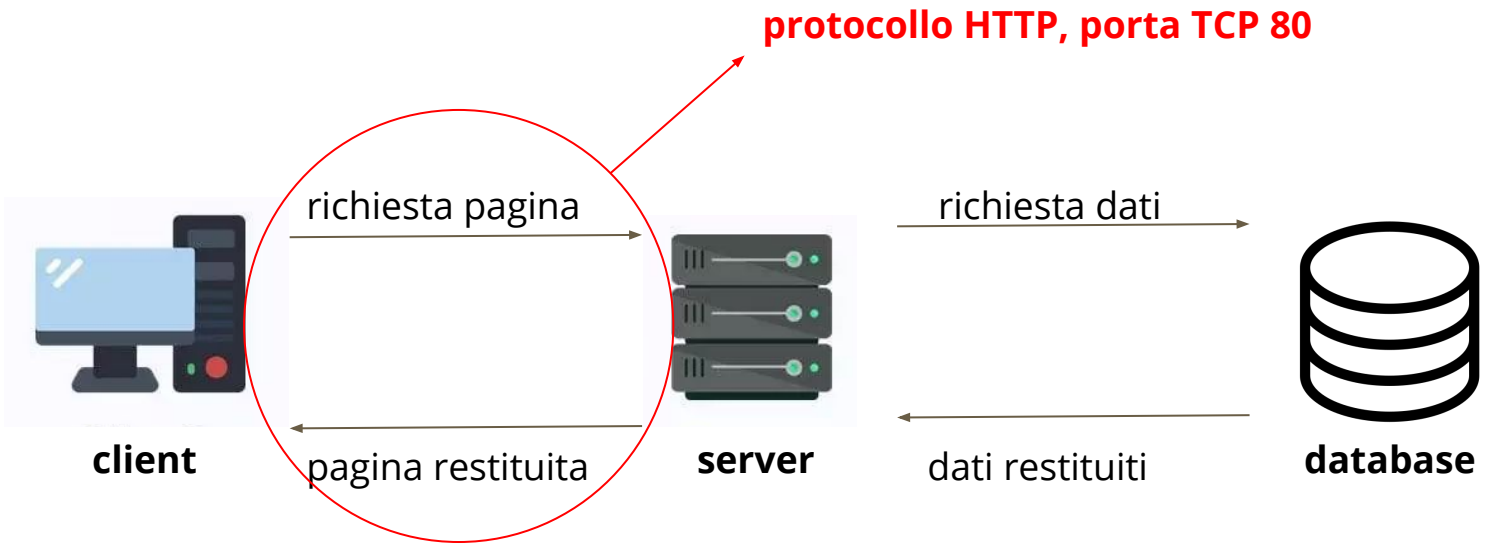
Web Server

- **Archivia i file di un sito web** e lo mette a disposizione dei client che visitano il sito.
- La comunicazione tra server e client avviene tramite il **protocollo HTTP**, che utilizza la **porta TCP 80 (o 8080),** o eventualmente la versione sicura HTTPS, che utilizza invece la 443.
- L'insieme di tutti i web server interconnessi a livello mondiale dà vita al **World Wide Web**.

Server vs Database



Server vs Database



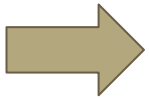
Creazione di un server con NodeJS

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

var server = http.createServer(function(req, res) {
  res.writeHead(200);
  res.end('Ciao a tutti, sono un web server!');
});
server.listen(8080);
```

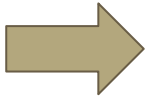
Creazione di un server con NodeJS

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



La **funzione require** consente di richiamare la **libreria http** che ci restituirà i metodi necessari per creare il nostro web server

```
var server = http.createServer();
```



La **variabile http** rappresenta l'oggetto JavaScript che ci permetterà di lanciare il server web

Creazione di un server con NodeJS

Il **metodo createServer** prende in ingresso dei parametri, e guarda caso il parametro che gli abbiamo passato è proprio una funzione!

```
var server = http.createServer(function(req, res) {
  res.writeHead(200);
  res.end('Ciao a tutti, sono un web server!');
});
```

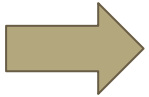
Tutte queste 4 righe corrispondono ad una singola chiamata del **metodo createServer()**!

Funzione di callback

- La **funzione di callback** che gli stiamo passando verrà richiamata nel momento in cui un visitatore si connetterà al sito.
- La funzione callback riceve due parametri in ingresso:
  - Richiesta del visitatore (req nel codice): questo oggetto contiene tutte le informazioni della richiesta del visitatore, ad esempio il suo IP, l'url che vuole visitare, se ha passato GET o POST e cosi via.
  - La risposta (res nel codice): questo è l'oggetto che conterrà la risposta restituita dal server al browser del visitatore. In questo caso abbiamo inserito un testo, solitamente contiene un codice HTML o, in caso di API dei dati JSON.

Funzione di callback

```
res.writeHead(200);
```

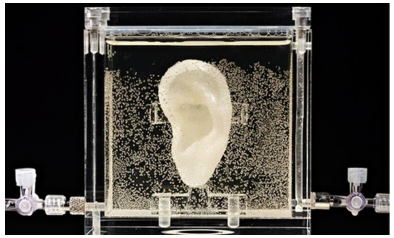


Quel **codice 200** si riferisce proprio all'intestazione, e serve per dire al browser "*Tutto ok amico, la pagina è questa!*"

```
res.end('Ciao a tutti,
sono un web server!');
```

Terminiamo la risposta con **end()**, che invierà il messaggio e l'intestazione al browser.

Server in ascolto



Per concludere, dobbiamo mettere in ascolto il nostro server verso una porta, la **porta 8080**, con l' istruzione:

```
server.listen(8080);
```

Piccola nota: *la porta standard quando si vuole contattare un web server è la 80. Quando vai su google.it in realtà stai contattando il server di google alla porta 80. Per il momento sei in un ambiente di sviluppo (il tuo computer) quindi puoi usare la porta 8080 oppure la 8081. Nel momento in cui vorrai pubblicare la tua web app Node.js dovrai modificare la porta su 80, sarà a questa porta che i visitatori contatteranno il tuo server.*

Come lanciare e verificare lo stato del server?

1. Aprire il terminale e dirigersi nella cartella in cui avete salvato il file
2. Lanciare il file con Node => `node myserver.js`
3. Connettersi al server scrivendo sulla barra degli indirizzi del browser:

http://localhost:8080/

Come lanciare e verificare lo stato del server?

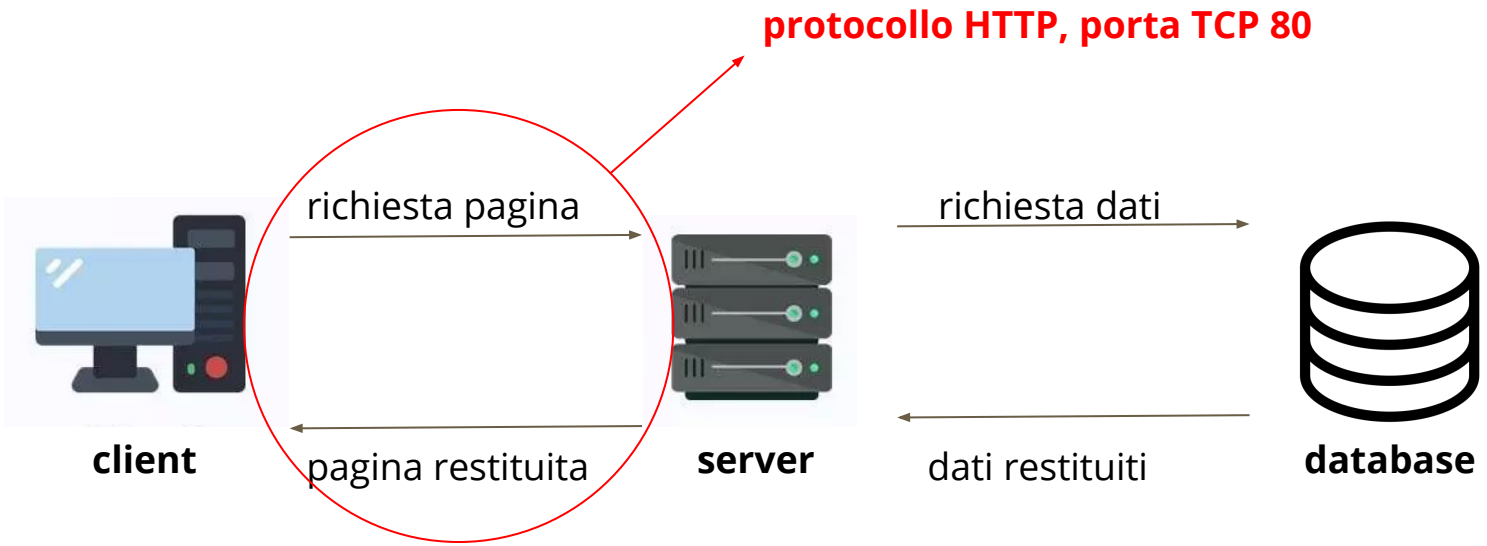
1. Aprire il terminale e dirigersi nella cartella in cui avete salvato il file
2. Lanciare il file con Node => `node myserver.js`
3. Connettersi al server scrivendo sulla barra degli indirizzi del browser:

http://localhost:8080/

ATTENZIONE!!! Valido solo quando il server gira sulla porta 8080!

Come lanciare e verificare lo stato del server?

Server vs Database



Tipi di richieste

- Il client (normalmente un browser) effettua una richiesta HTTP al server, il server elabora la richiesta e restituisce una risposta. La risposta può essere una pagina HTML, un file JSON, un'immagine o qualsiasi altro formato.
- La specifica HTTP definisce 9 tipi di metodi alcuni dei quali non sono però usati o supportati da PHP; i più diffusi restano sicuramente **GET** e **POST**.
- GET e POST spiegazione discorsiva => https://www.html.it/pag/62463/le-richieste-http-get-e-post/

GET

- Il metodo con cui vengono richieste la maggior parte delle informazioni ad un Web server.
- Tali richieste vengono veicolate tramite **query string**, cioè la parte di un URL che contiene dei parametri da passare in input ad un'applicazione.
- È consigliato soprattutto in quelle richieste in cui è utile salvare nell'URL i parametri richiesti, ad esempio per poter essere cachati. Un classico caso d'uso è una **query di ricerca**.

http://www.miosito.com/pagina-richiesta?id=123&page=3

http://localhost/search.php?author=Pippo

POST

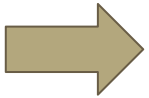
- Il metodo POST si differenza da GET in quanto i parametri della richiesta non vengono passati in query string e quindi non possono essere tracciati nemmeno negli access log dei web server.
- Caso d'uso comune di una richiesta in POST è un form che invia dati personali, come in una registrazione.

Effettuare una GET con NodeJS (pagina richiesta)

```
var http = require('http');
var url = require('url');
var server = http.createServer(function(req, res) {
  var page = url.parse(req.url).pathname;
  console.log(page);
  res.writeHead(200, {"Content-Type": "text/plain"});
  if (page == '/') {
    res.write('Sei nella Reception. Posso aiutarti?');
  } else if (page == '/basement') {
    res.write('Sei nella cantina dei vini. Le bottiglie sono mie!');
  }  else if (page == '/floor/1/bedroom') {
    res.write('Oh, non dovresti essere qui!');
  }
  res.end();
});
server.listen(8080);
```

Effettuare una GET con NodeJS

```
var url = require("url");
```



Per scoprire quale pagina ha richiesto il visitatore useremo un nuovo modulo Node chiamato “url”.

```
url.parse(req.url).pathname;
```

Utile ad ottenere il nome della pagina richiesta

Effettuare una GET con NodeJS (parametri)

```
var http = require('http');
var url = require('url');
var querystring = require('querystring');
var server = http.createServer(function(req, res) {
  var params = querystring.parse(url.parse(req.url).query);
  res.writeHead(200, {"Content-Type": "text/plain"});
  if ('firstname' in params && 'lastname' in params) {
    res.write('Il tuo nome: ' + params['firstname'] + ' ' + params['lastname']);
  } else {
    res.write('Tu hai un nome giusto?');
  }
  res.end();
});
server.listen(8080);
```

Effettuare una GET con NodeJS

```
url.parse(req.url).query;
```

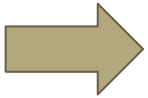
Utile a recuperare i parametri GET che vengono aggiunti **alla fine dell'URL, dopo il percorso del file**.

```
http://localhost:8080/page?firstname=Alberto&
lastname=Olla
```

?firstname=Alberto&lastname=Olla

Effettuare una GET con NodeJS

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



Modulo utile per ottenere dalla stringa contenente i parametri, i parametri stessi

```
var params = querystring.parse(url.parse(req.url).query);
```

A questo punto avremo una lista di parametri “params”. Per recuperare il valore del primo (il nome) basta scrivere: params[‘firstname’]. Il **‘firstname’ di params** permette di verificare se l’array params contiene una voce ‘firstname’. Se manca puoi visualizzare un messaggio di errore, in caso contrario verrebbe mostrato il valore ‘undefined’, ovvero non definito.

Effettuare una POST con NodeJS

1. Il **‘firstname’ di params** permette di verificare se l’array params contiene una voce ‘firstname’. Se manca puoi visualizzare un messaggio di errore, in caso contrario verrebbe mostrato il valore ‘undefined’, ovvero non definito.
2. Inoltre, **non abbiamo controllato la pagina chiamata**. Questo codice funziona anche su http://localhost:8080 come su http://localhost:8080/imaginarypage. Questo e il codice precedente devono essere combinati per gestire sia la pagina che i parametri.

Effettuare una POST con NodeJS

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

var server = http.createServer(function(req, res) {
  var params = querystring.parse(url.parse(req.url).query);
  res.writeHead(200, {"Content-Type": "text/plain"});
  if ('firstname' in params && 'lastname' in params) {
    res.write('Il tuo nome: ' + params['firstname'] + ' ' + params['lastname']);
  } else {
    res.write('Tu hai un nome giusto?');
  }
  res.end();
});
server.listen(8080);
```