

Javascript

JavaScript is always synchronous and single-threaded. If you're executing a JavaScript block of code on a page then no other JavaScript on that page will currently be executed.



synchronous, single thread of control



synchronous, two threads of control



asynchronous



Javascript – Callback and Promise

One approach to asynchronous programming is to make functions that perform a slow action take an extra argument, a *callback function*. The action is started, and when it finishes, the callback function is called with the result.

```
setTimeout(() => console.log("Tick"), 500);
```

A *promise* is an asynchronous action that may complete at some point and produce a value. It is able to notify anyone who is interested when its value is available.

```
let fifteen = Promise.resolve(15);  
fifteen.then(value => console.log(`Got ${value}`));
```

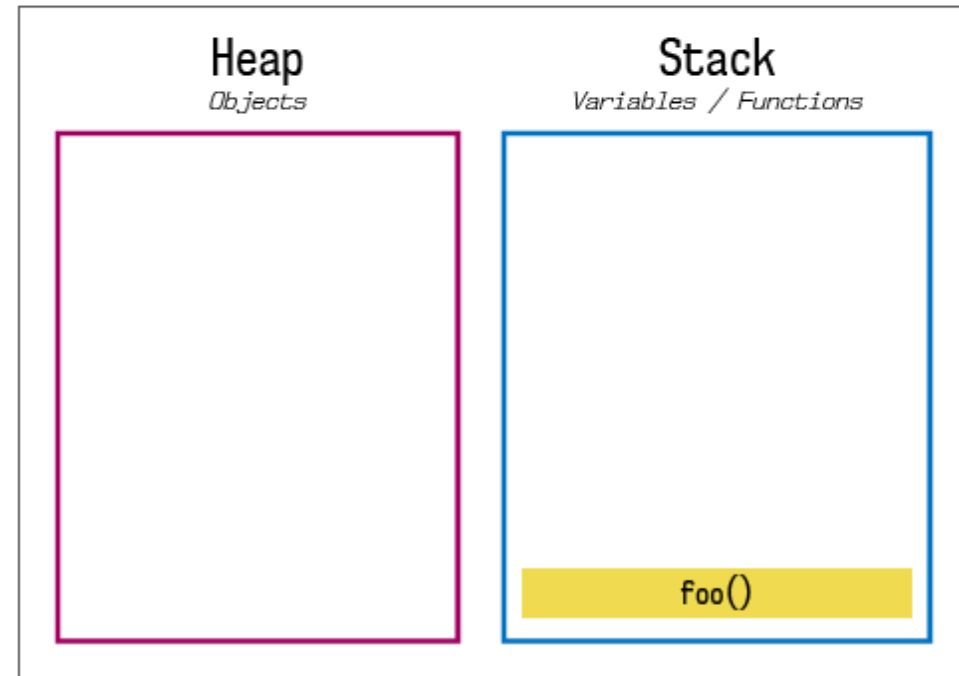


Javascript – Callback and Promise

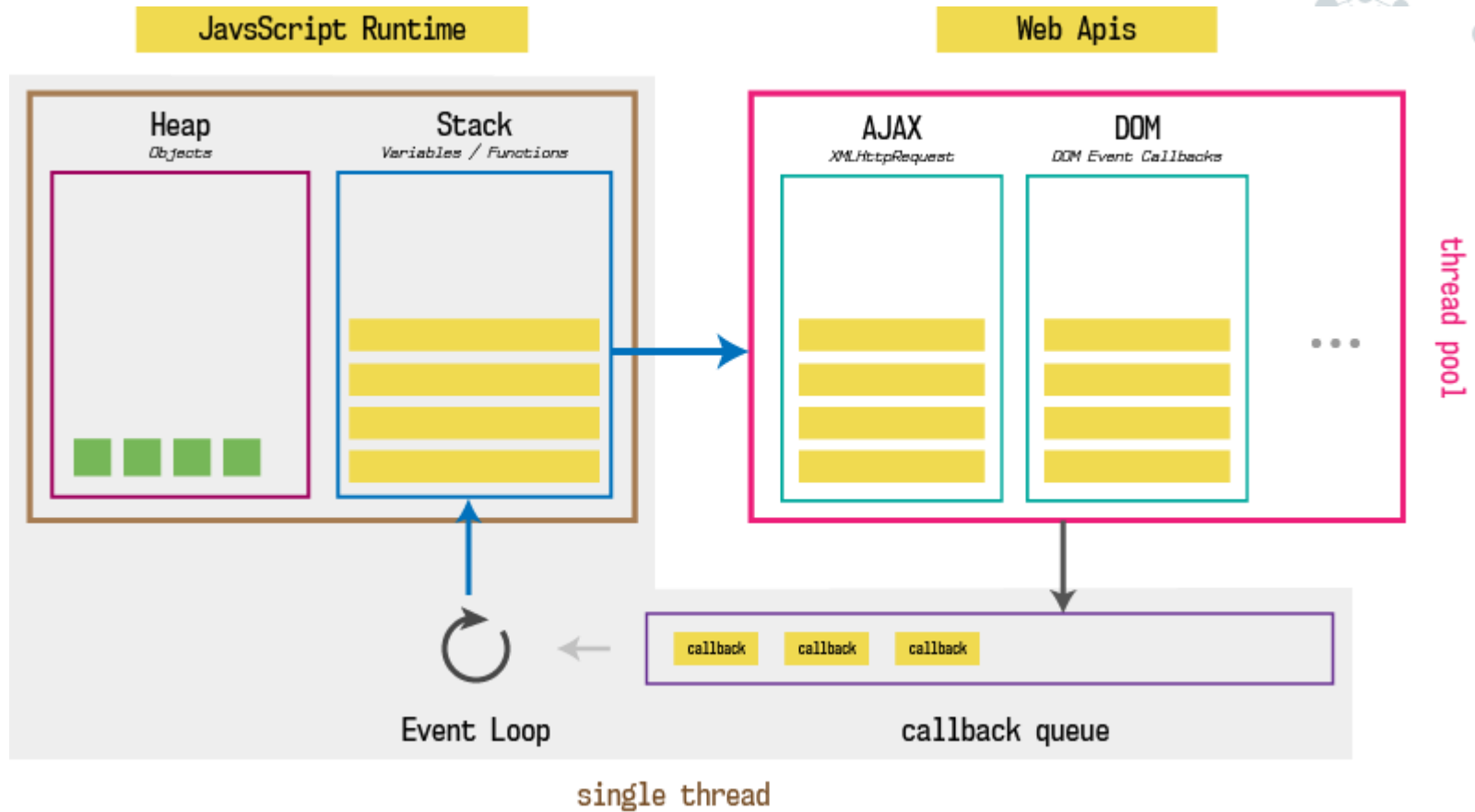
Javascript Program

```
function baz(){  
  console.log('Hello from baz');  
}  
  
function bar() {  
  baz();  
}  
  
function foo() {  
  bar();  
}  
  
foo();
```

Javascript Runtime



Javascript – Callback and Promise



Javascript – Callback and Promise

The screenshot shows the Loupe web browser interface. On the left is a code editor with the following JavaScript code:

```
1  
2  
3 function printHello() {  
4   console.log('Hello from baz');  
5 }  
6  
7 function baz() {  
8   setTimeout(printHello, 3000);  
9 }  
10  
11 function bar() {  
12   baz();  
13 }  
14  
15 function foo() {  
16   bar();  
17 }  
18  
19 foo();
```

Below the code editor is a button labeled "Click me!" and an "Edit" button. To the right of the code editor are three panels: "Call Stack", "Web Apis", and "Callback Queue". The "Call Stack" and "Web Apis" panels are empty. The "Callback Queue" panel contains a single blue circle representing a callback. An orange circular arrow icon is positioned between the "Web Apis" and "Callback Queue" panels, indicating the flow of execution from the Web Apis to the Callback Queue.

Javascript – Callback and Promise

<https://www.youtube.com/watch?v=8aGhZQkoFbQ>

<http://latentflip.com/loupe/>



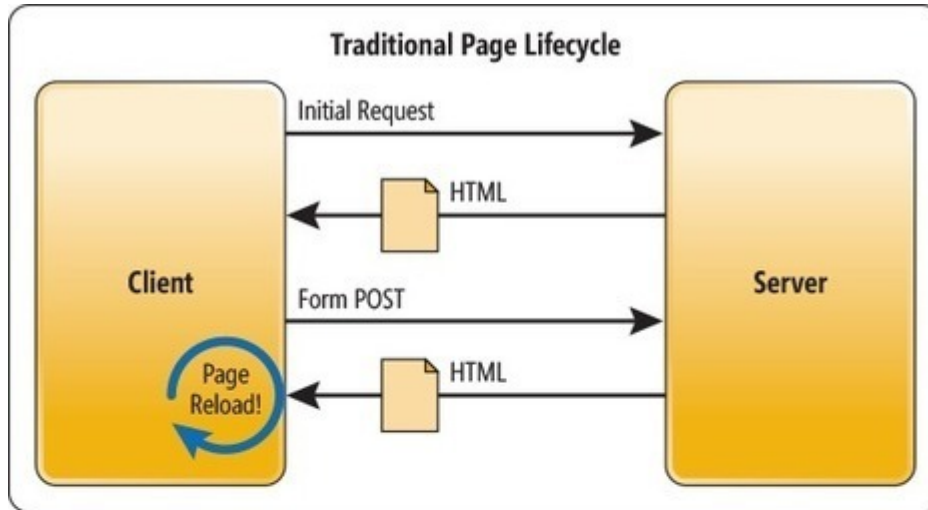
A video frame showing a man in a purple t-shirt with a 'JS' logo on it, standing at a podium. The podium has a laptop with 'JS' and '&reat' logos. A large pink slide is displayed behind him with the text 'What the heck is the event loop anyway?' and '@philip_roberts'. A small 'JS' logo is in the top left corner of the video frame. At the bottom of the video frame, there is a subtitle: 'I want to talk about the event loop and what the heck is the event loop, as in the event'. The podium also has a sign that says 'YCONFEU · BERLIN'.

Web e pattern architettonici

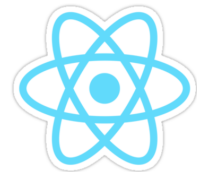
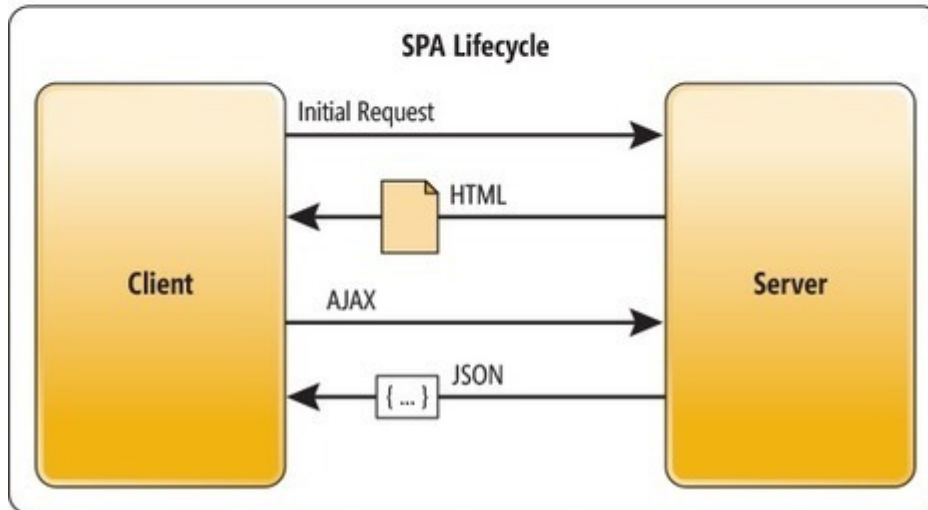


Pattern architetturali

Multi-Page Application



Single-Page Application



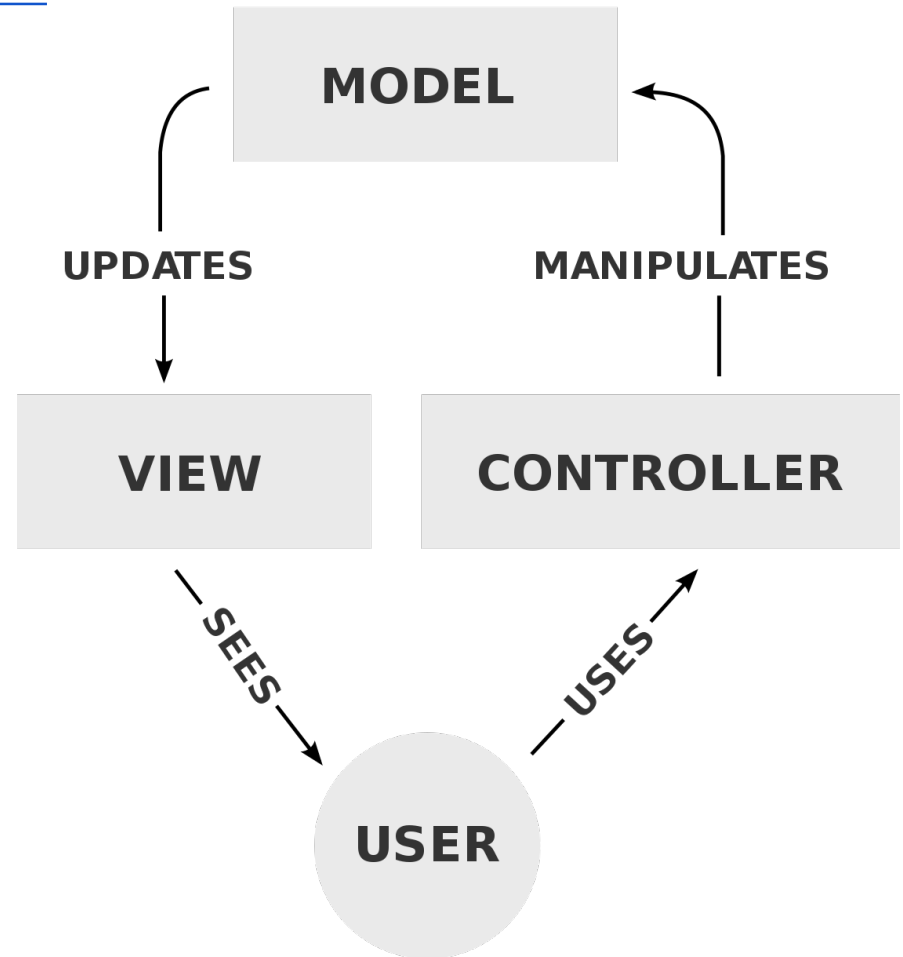
and rest of world

Pattern MVC

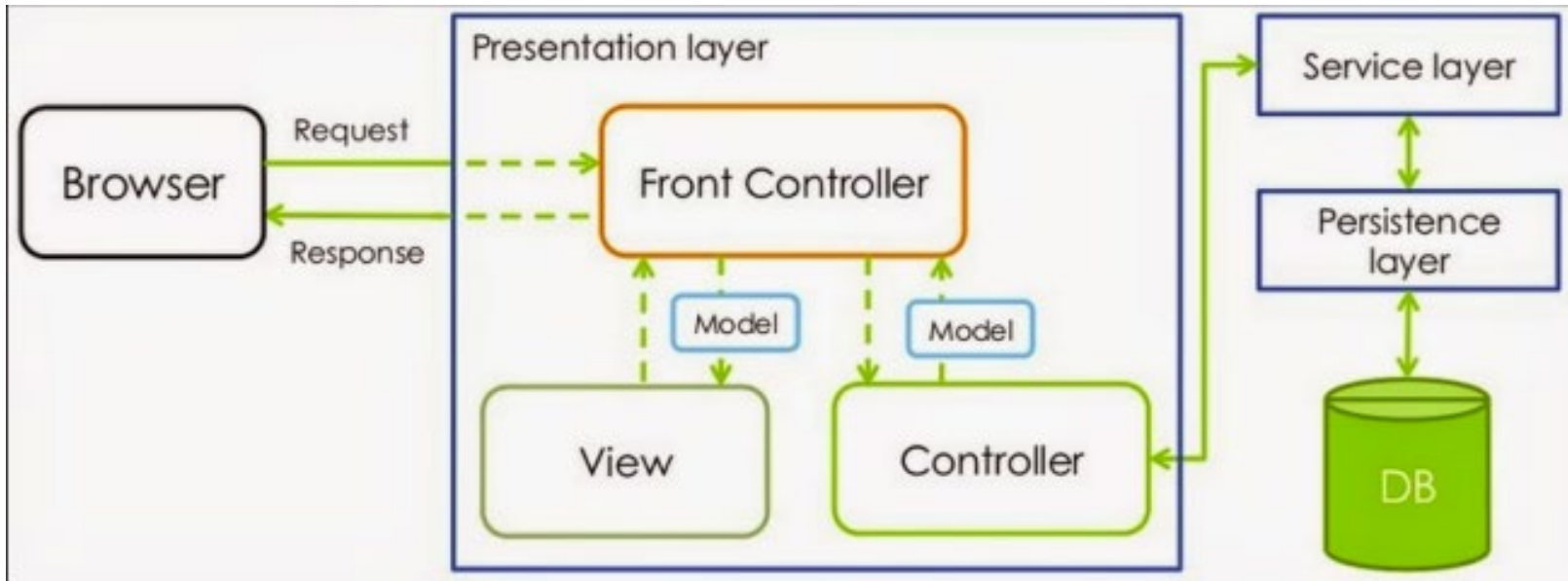
<https://it.wikipedia.org/wiki/Model-view-controller>

Vantaggi:

- 1) Disaccoppiare
- 2) Responsabilità certe
- 3) View multiple



Architettura generica



The presentation layer is where the data is formatted and presented to the user.

The service layer is where the business logic of the application is implemented.

The persistence layer is where the data is simply saved or retrieved.

AJAX – L'inizio delle SPA

<https://embed.plnkr.co/rgh75JGDGuyB4UhBvTYN/>

```
<body>
  <h1>Hello <span id="firstname"></span> <span id="lastname"></span>!</h1>
  <button onclick="reload_user();">Reload!</button>
  <p id="loadingtext">LOADING....</p>
</body>

</html>
```

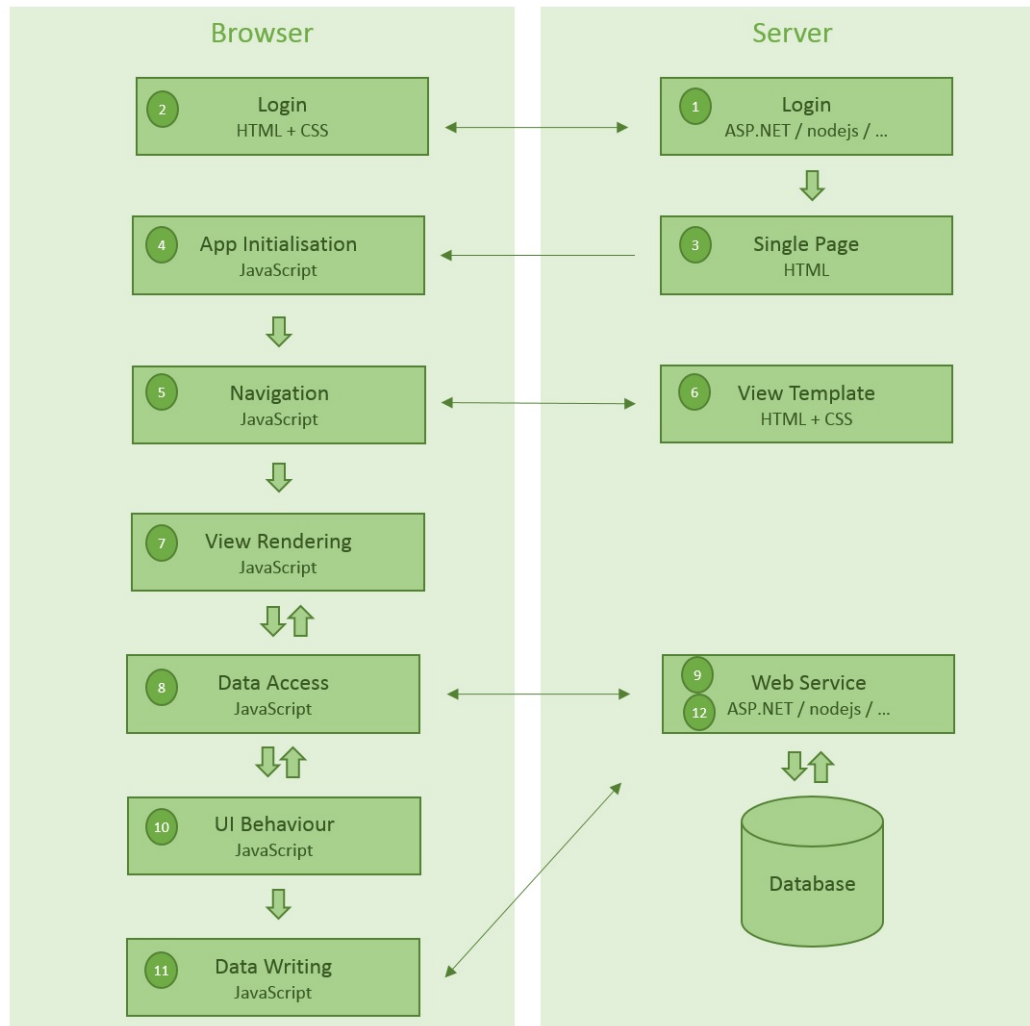
```
$(document).ready(function() {
  reload_user();
});

function reload_user(){
  $("#loadingtext").show();
  $.ajax({
    method: "get",
    url: "https://randomuser.me/api",
    datatype: "json",
    data: { results: 1 },
    success: function(r) {
      $("#firstname").text(r.results[0].name.first);
      $("#lastname").text(r.results[0].name.last);
    },
    error: function() {
      alert("error");
    },
    complete: function(){
      $("#loadingtext").fadeOut();
    }
  });
}
```

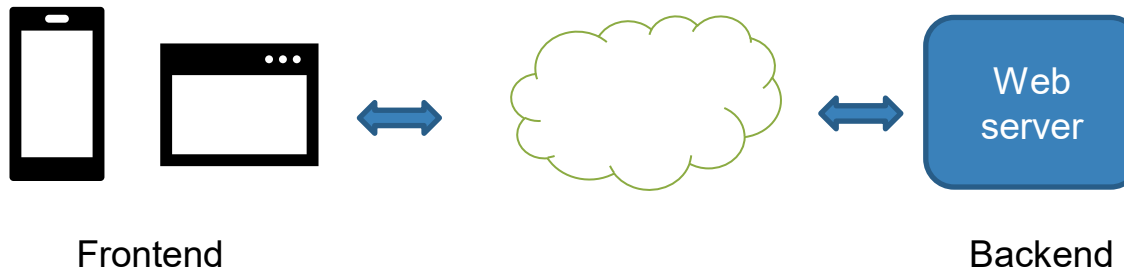


Asynchronous Javascript And XML

Esempio di SPA



Il mondo reale è composto da soluzioni ibride



Approcci ibridi
Mix di soluzioni
Evoluzione continua

Trend:

- PWA vs Mobile
- Low Code
- Serverless
- Static site generators
- MicroService

UI Bakery

<https://www.youtube.com/watch?v=xbB3MrEi5bo>

Less servers for your Angular app

<https://www.youtube.com/watch?v=WEYtDYBkall>

Mastering Chaos - A Netflix Guide to Microservices

<https://www.youtube.com/watch?v=CZ3wluvmHeM>

Backend



Di cosa si occupa il backend (o i backend)

- Rispondere a richieste da parte dei client su protocollo http/https/http2
- Interpretare le URL richieste/header/cookie
- Autenticare un utente
- Autorizzare un utente dopo la sua autenticazione
- Servire contenuti statici
- Generare pagine dinamiche
- Rispondere a chiamate REST da una SPA
- Gestire cache
- Servire contenuti in streaming
-

Come fa il backend a rispondere alle richieste?

Semplicemente utilizzando i socket ed i metodi di listen

<https://docs.microsoft.com/it-it/dotnet/framework/network-programming/synchronous-server-socket-example>

```
// Create a TCP/IP socket.
Socket listener = new Socket(ipAddress.AddressFamily,
    SocketType.Stream, ProtocolType.Tcp );

// Bind the socket to the local endpoint and
// listen for incoming connections.
try {
    listener.Bind(localEndPoint);
    listener.Listen(10);

    // Start listening for connections.
    while (true) {
        Console.WriteLine("Waiting for a connection...");
        // Program is suspended while waiting for an incoming connection.
        Socket handler = listener.Accept();
        data = null;

        // An incoming connection needs to be processed.
        while (true) {
            int bytesRec = handler.Receive(bytes);
            data += Encoding.ASCII.GetString(bytes,0,bytesRec);
            if (data.IndexOf("<EOF>") > -1) {
                break;
            }
        }

        // Show the data on the console.
        Console.WriteLine( "Text received : {0}", data);

        // Echo the data back to the client.
        byte[] msg = Encoding.ASCII.GetBytes(data);

        handler.Send(msg);
        handler.Shutdown(SocketShutdown.Both);
        handler.Close();
    }
} catch (Exception e) {
    Console.WriteLine(e.ToString());
}
```

<https://gist.github.com/teadmiston/5935757>

```
9  var net = require('net');
10
11  var server = net.createServer(function(socket) {
12      socket.write('Echo server\r\n\r\n');
13      socket.pipe(socket);
14  });
15
16  server.listen(1337, '127.0.0.1');
17
```

Ma devo implementarmi il protocollo HTTP?

node
express

NEXT.js

spring
boot

ASP.NET Core

nest

Come fa il backend a rispondere alle richieste con express?

<https://expressjs.com/en/starter/hello-world.html>



```
1 const express = require('express' 4.17.1 )
2 const app = express()
3 const port = 3000
4
5 app.get('/', (req, res) => res.send('Hello World!'))
6
7 app.listen(port, () => console.log(`Example app listening on port ${port}!`))
```

Save on RunKit

Node 10 ↕

help

URL: <https://jt9ee7g2hkau.runkit.sh>

Come restituire un file html

```
//assuming app is express Object.
app.get('/', function(req, res) {
  res.sendFile('index.html');
});
```

Routing: Interpretare le URL richieste

Il routing è responsabile del mapping degli URI di richiesta agli endpoint e dell'invio di richieste in ingresso a tali endpoint. Le route sono definite e configurate all'avvio.

Metodi di route

Un metodo di route deriva da uno dei metodi HTTP ed è collegato ad un'istanza delle classe express.

Il codice seguente è un esempio di route definite per i metodi GET e POST nella root dell'app.

```
// GET method route
app.get('/', function (req, res) {
  res.send('GET request to the homepage');
});

// POST method route
app.post('/', function (req, res) {
  res.send('POST request to the homepage');
});
```

Routing con parametri:

```
8
9  app.get('/contact', function(req, res){
10   res.send('this is the contact page');
11  });
12
13  app.get('/profile/:id', function(req, res){
14   res.send('You request get(key: ?) profile with the id of ' + req.params.id);
15  });
16
17  app.listen(3000);
```