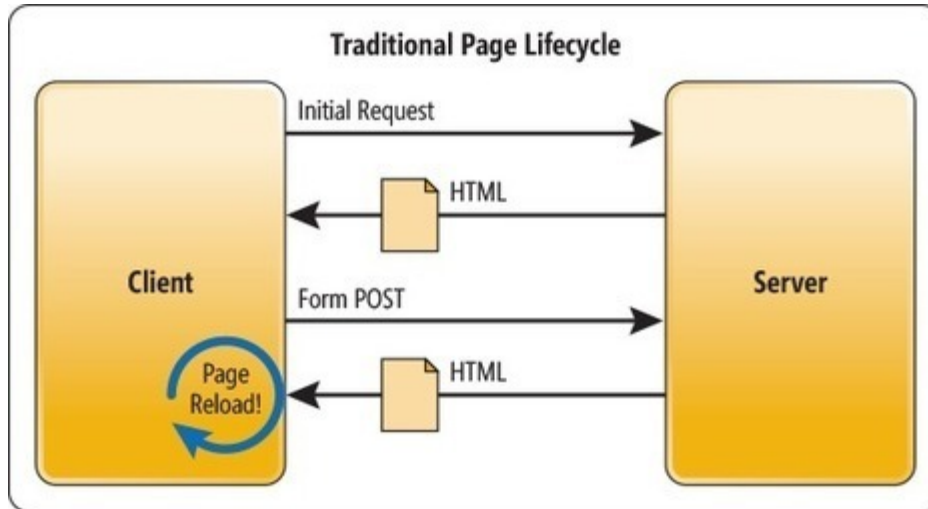


Web e pattern architettonici

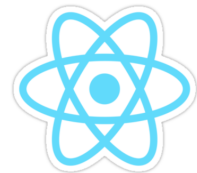
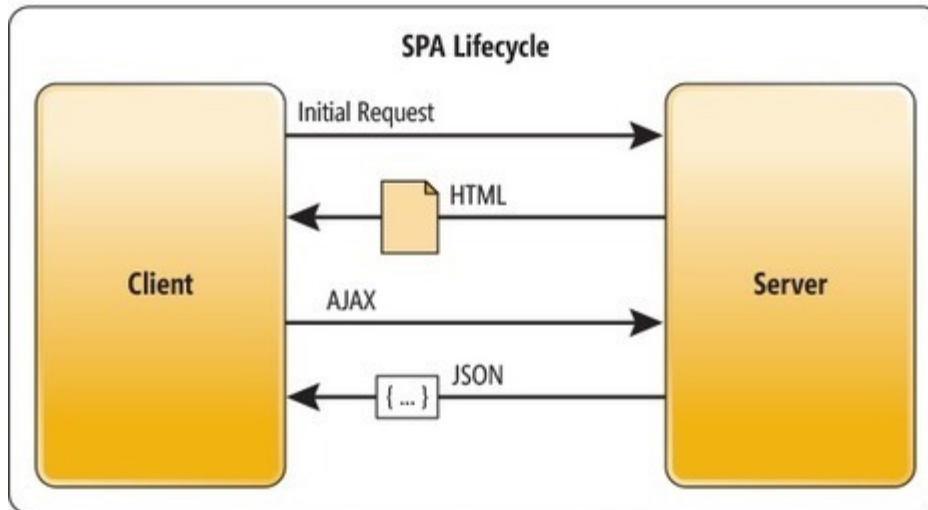


Pattern architetturali

Multi-Page Application



Single-Page Application



© 2019 Red Hat, Inc.

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>
```

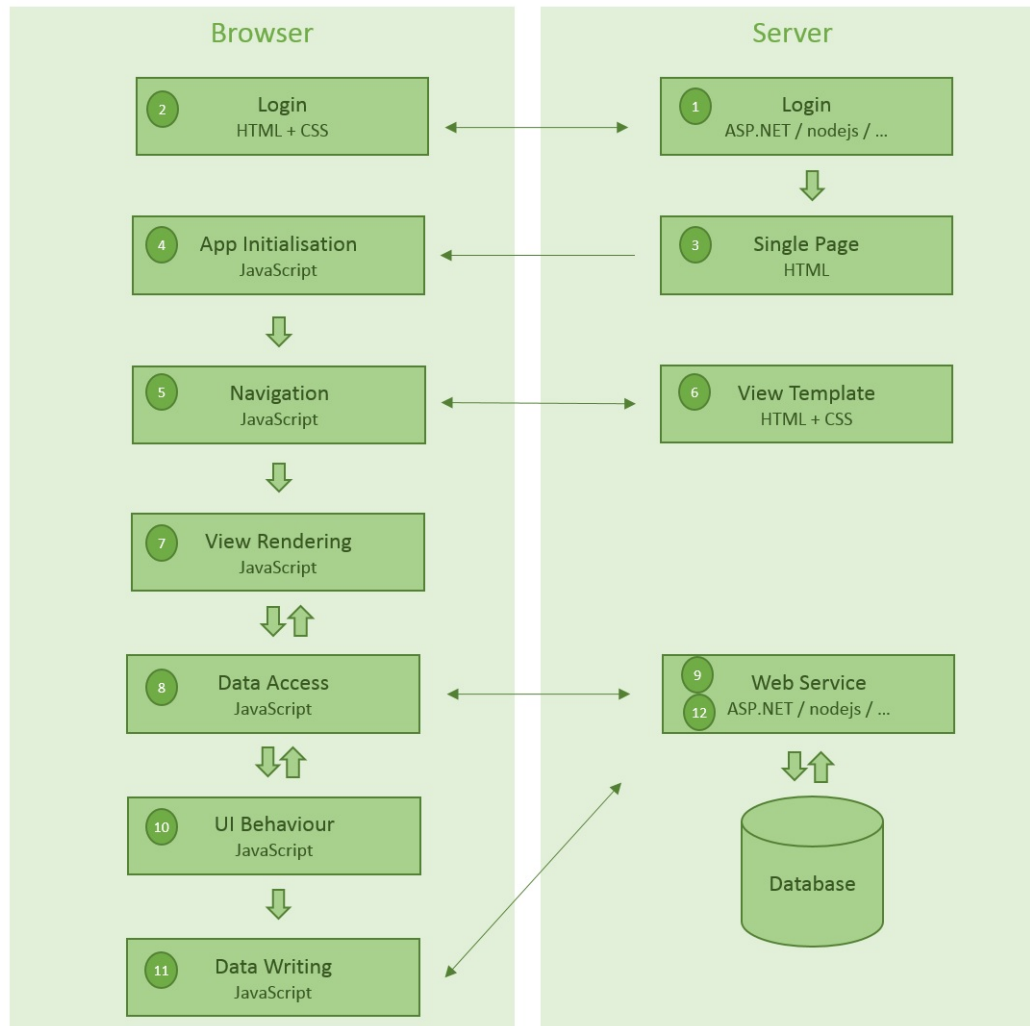
```
$(document).ready(function() {
  reload_user();
});

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



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/tedmiston/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);
```

Security

Malicious code on library



Un buon inizio da leggere

If an attacker successfully injects any code at all, it's pretty much game over

XSS is too small scale, and really well protected against.

Chrome Extensions are too locked down.

Lucky for me, we live in an age where people install npm packages like they're popping pain killers.

I was excited at this point — I had a compelling package — but I didn't want to wait around while people slowly discovered it and spread the word. So I set about making PRs to existing packages that added my colourful package to their dependencies.

I've now made several hundred PRs (various user accounts, no, none of them as "David Gilbertson") to various frontend packages and their dependencies. "Hey, I've fixed issue x and also added some logging."

Look ma, I'm contributing to open source!

There are a *lot* of sensible people out there that tell me they don't want a new dependency, but that was to be expected, it's a numbers game.

<https://medium.com/hackernoon/im-harvesting-credit-card-numbers-and-passwords-from-your-site-here-s-how-9a8cb347c5b5>

Un caso famoso

Malicious code found in npm package event-stream downloaded 8 million times in the past 2.5 months



NOVEMBER 26, 2018 | IN [VULNERABILITIES](#) | BY DANNY GRANDER

A widely used npm package, `event-stream`, has been found to contain a malicious package named `flatmap-stream`. This was disclosed via a [GitHub issue](#) raised against the source repo.

<https://snyk.io/blog/malicious-code-found-in-npm-package-event-stream/>

`flatmap-stream` is a malicious package which was used in order to steal bitcoins from wallets. The malicious code was able to check if the `copydash` package was installed, and then attempt to steal the bitcoins stored in it. It was distributed by hijacking the popular `event-stream` package and adding `flatmap-stream` as a dependency.

<https://snyk.io/vuln/SNYK-JS-FLATMAPSTREAM-72637>