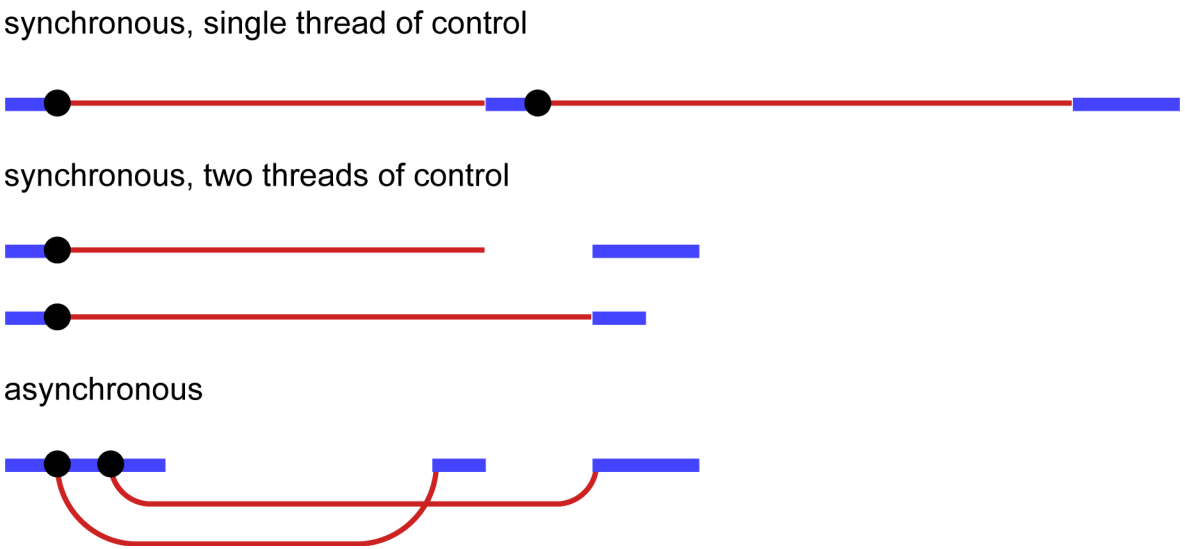
Javascript

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.



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.

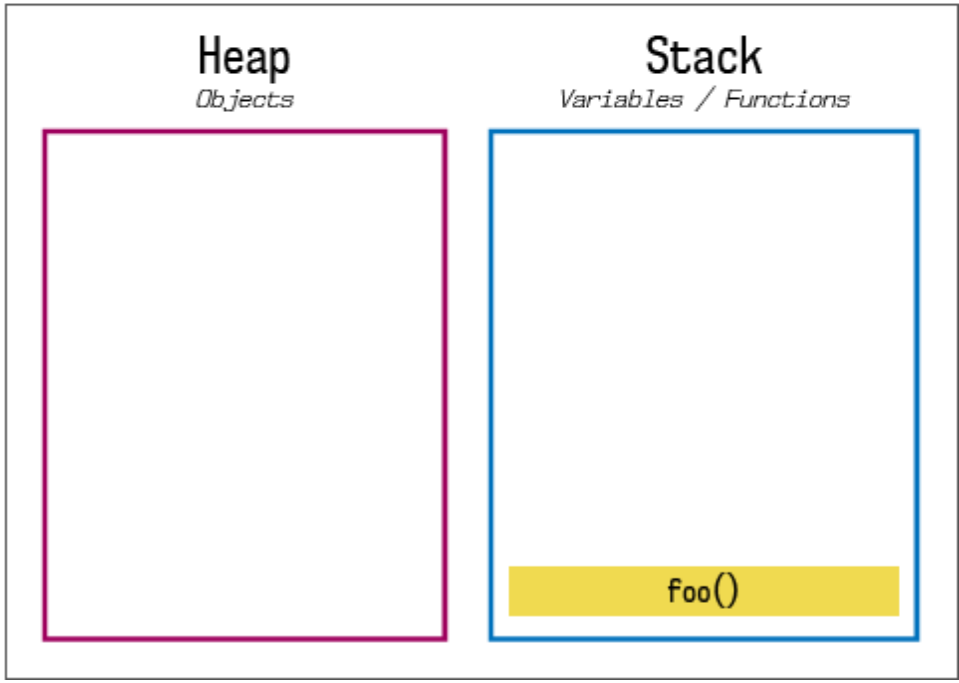
```
let fifteen = Promise.resolve(15);
fifteen.then(value => console.log(`Got ${value}`));
```

Javascript – Callback and Promise

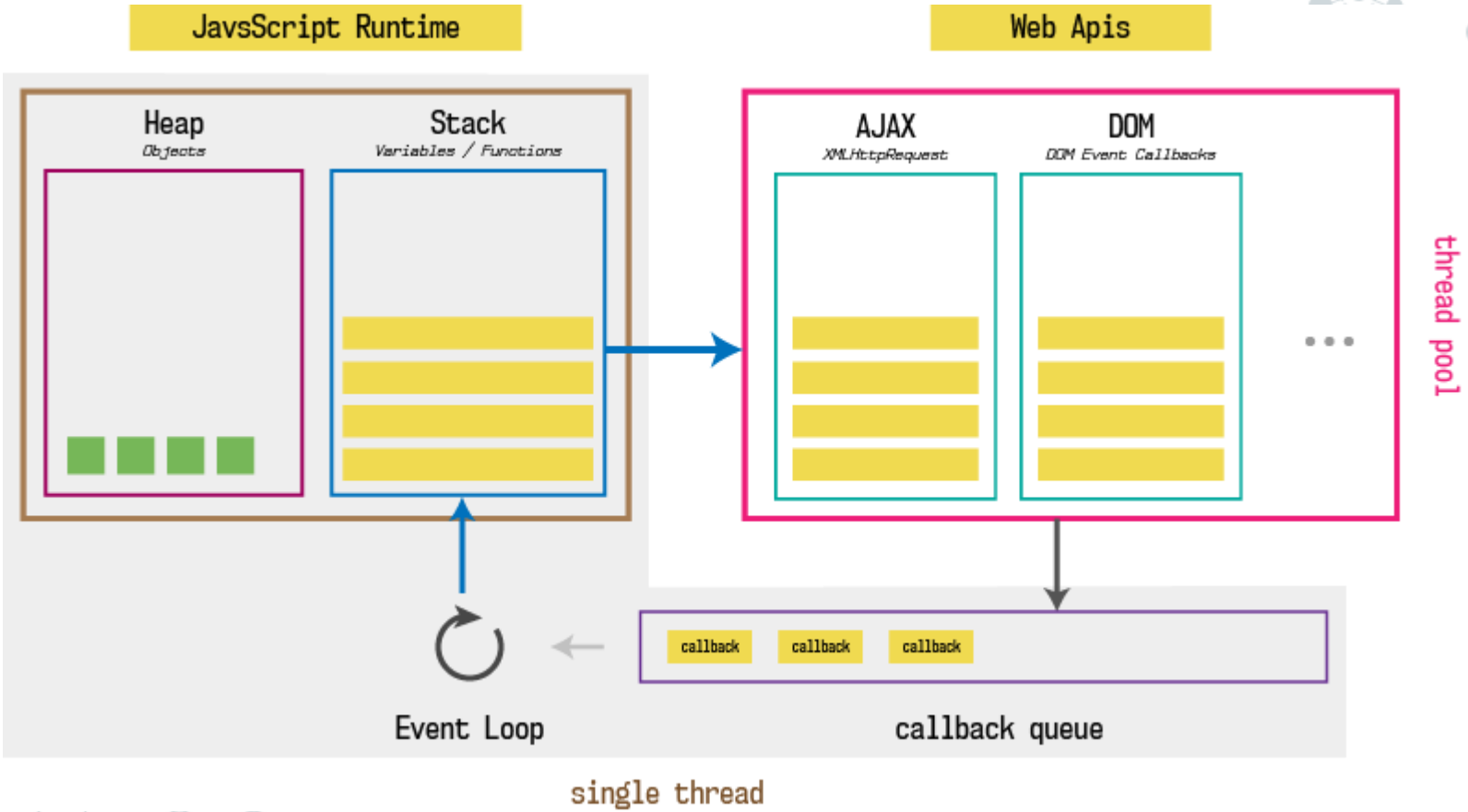
JavsScript Program

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

function bar() {
   baz();
}

function foo() {
   bar();
}

foo();
```

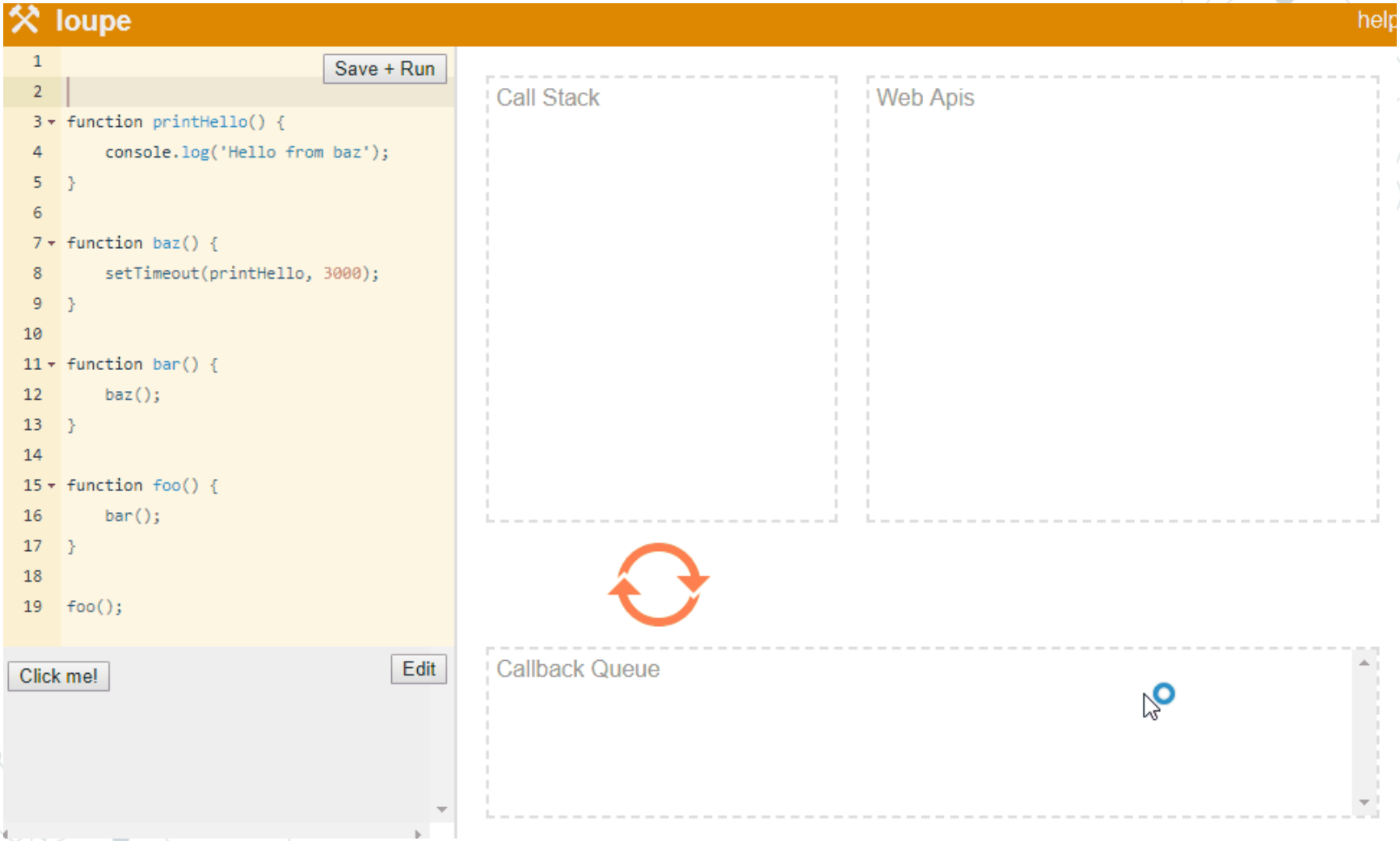
JavsScript Runtime



Javascript – Callback and Promise



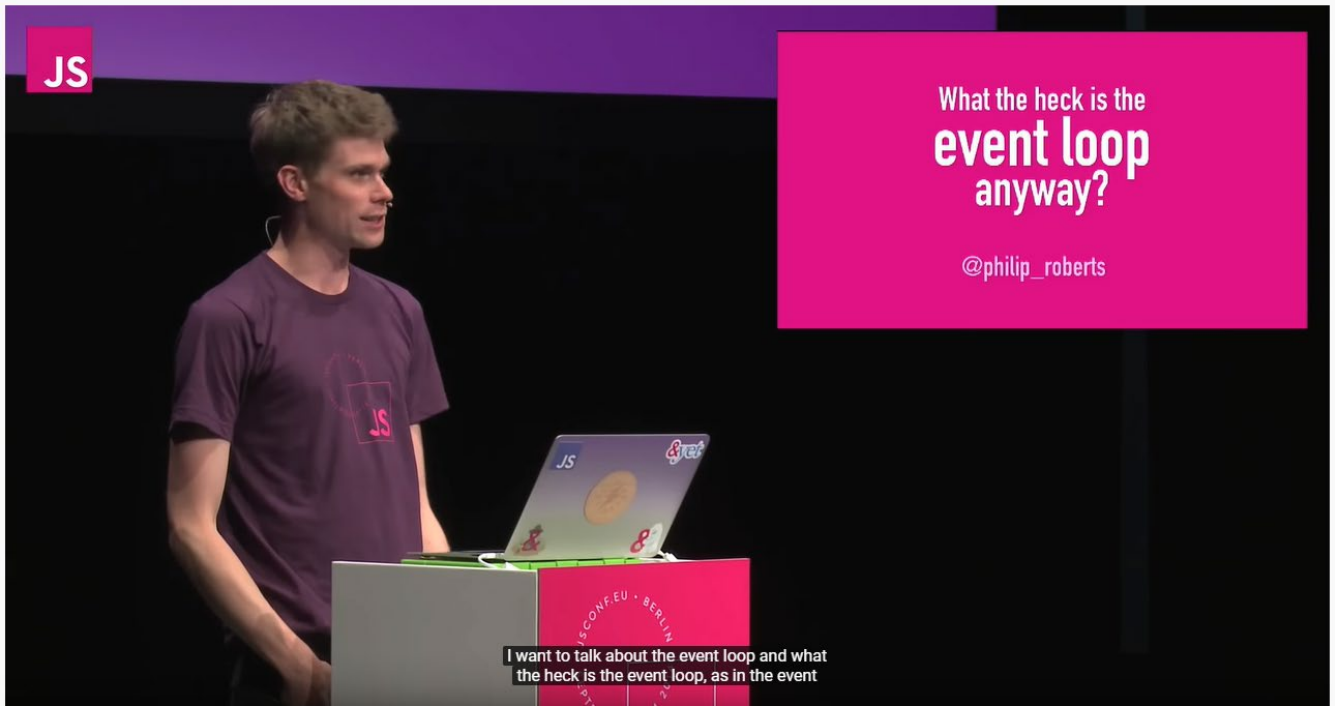
Javascript – Callback and Promise



Javascript – Callback and Promise

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

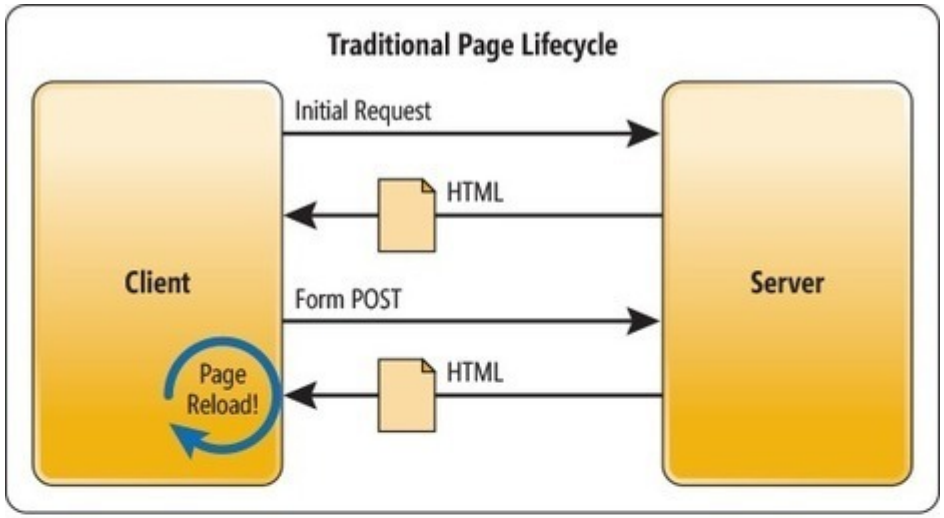
http://latentflip.com/loupe/



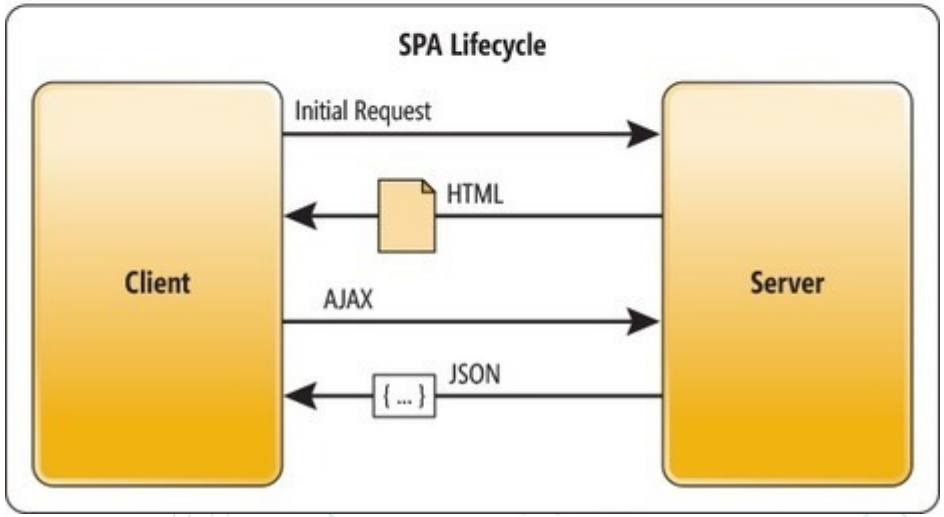
Web e pattern architetturali

Pattern architetturali

Multi-Page Application



Single-Page Application

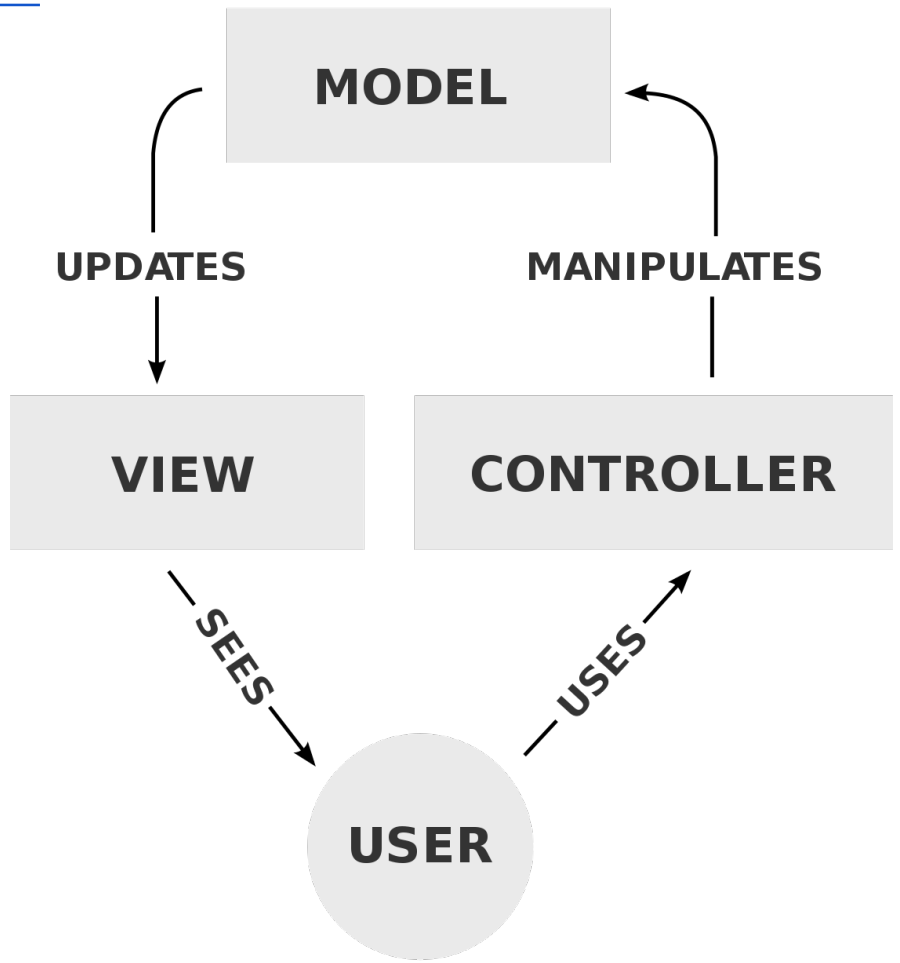


Pattern MVC

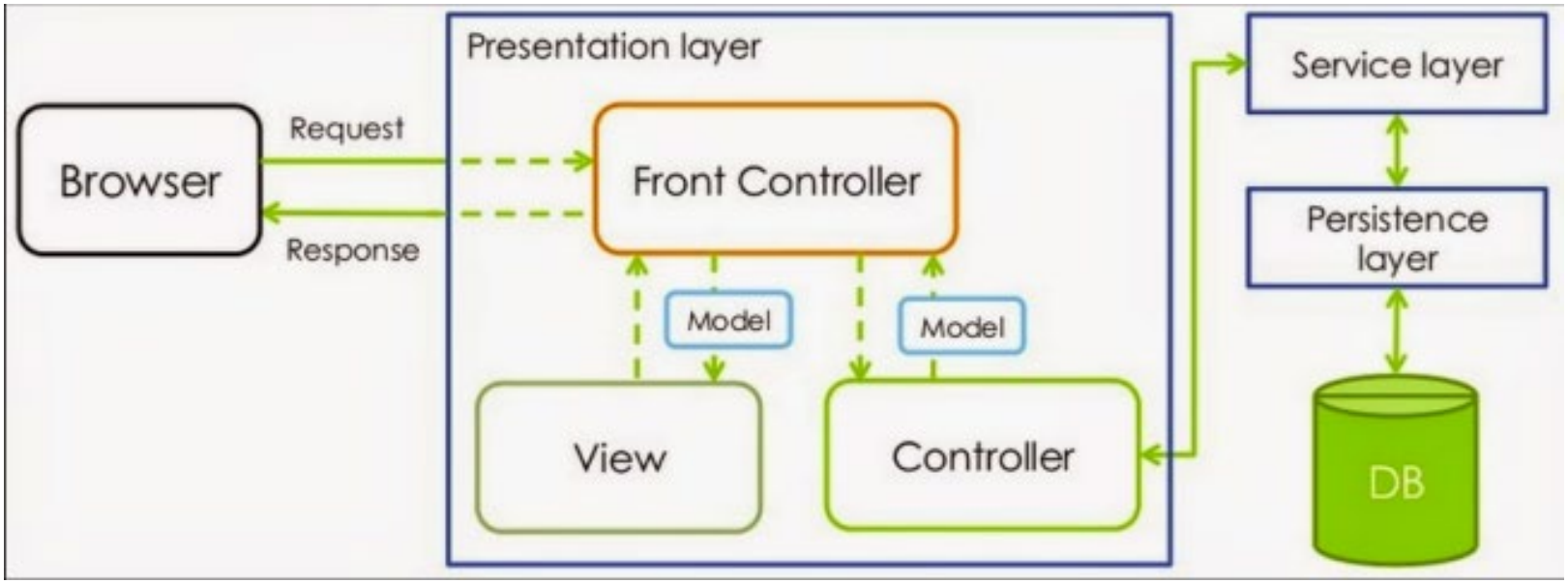
https://it.wikipedia.org/wiki/Model-view-controller

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 wherethe business logic of the application is implemented.

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

AJAX – L'inizio delle SPA

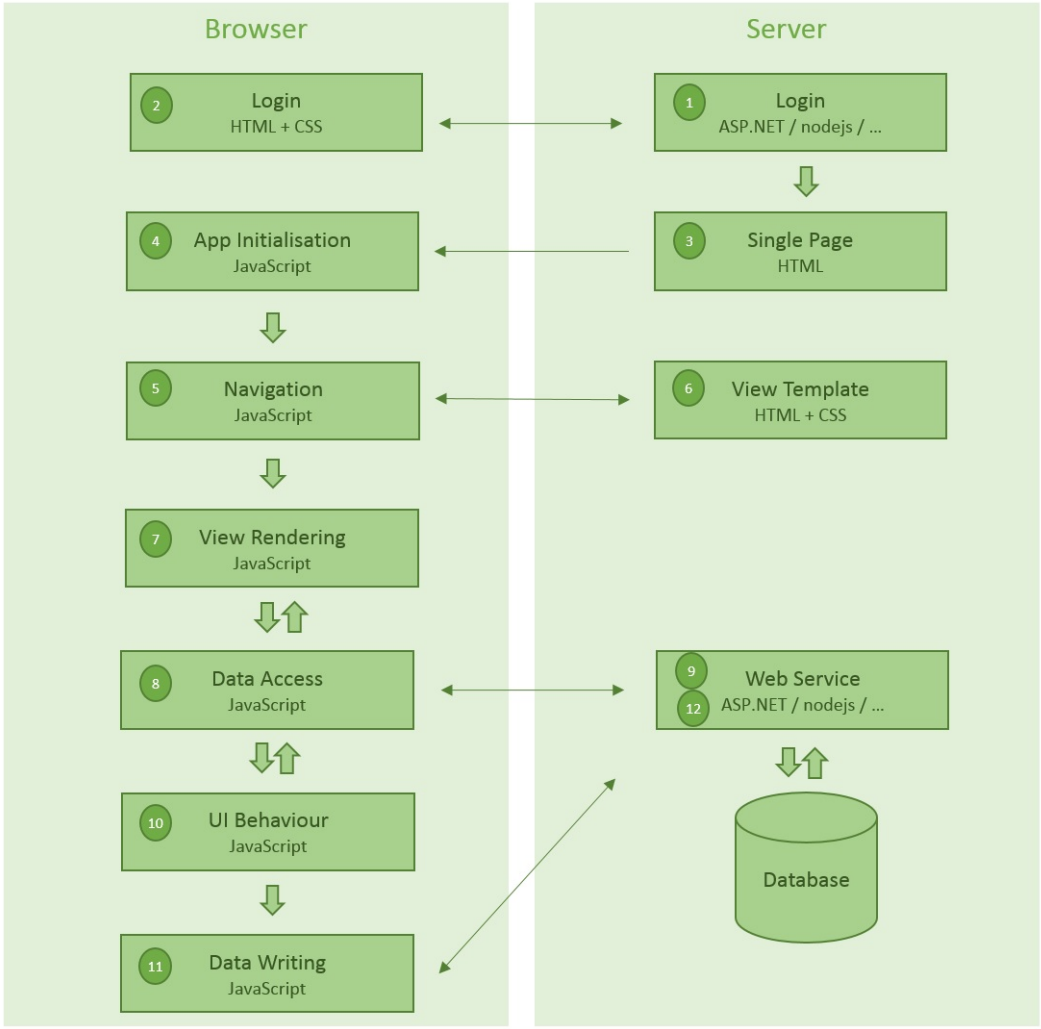
https://embed.plnkr.co/rgh75JGDGuyB4UhBvTYN/

```
<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

Frontend

Backend

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=WEYtDYBkaII

Mastering Chaos - A Netflix Guide to Microservices
https://www.youtube.com/watch?v=CZ3wIuvmHeM

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

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

var server = net.createServer(function(socket) {
        socket.write('Echo server\r\n');
        socket.pipe(socket);
});

server.listen(1337, '127.0.0.1');
```

Ma devo implementarmi il protocollo HTTP?

Come fa il backend a rispondere alle richieste con express?

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


```
const express = require('express' 4.17.1 )
const app = express()
const port = 3000

app.get('/', (req, res) => res.send('Hello World!'))

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: ?) rofile with the id of ' + req.params.id);
15  });
16
17  app.listen(3000);
```