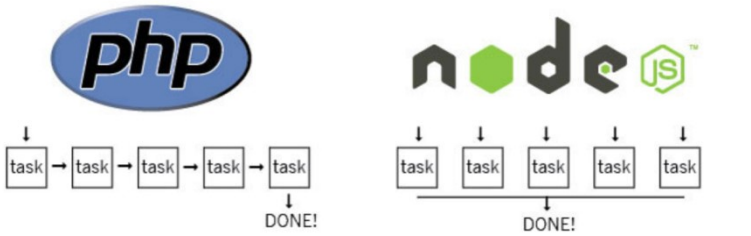
Scambio bilaterale sincronizzato tra client e server

Socket.io si basa su una serie di tecniche diverse che permettono la comunicazione in tempo reale.
La più conosciuta, e più recente, è WebSocket.
Si tratta di un recente sviluppo che è apparso più o meno nello stesso momento di HTML5, ma non è HTML, è una JavaScript API.
WebSocket è una funzione supportata da tutti i browser recenti. Consente lo scambio bilaterale sincronizzato tra client e server.
La comunicazione sul web è di solito non sincronizzata. Internet è sempre stato strutturato così:
***il client chiede di visualizzare una pagina e il server risponde*.**
Il server non ha nessun mezzo per inviare comunicazioni al client, a meno che non sia quest'ultimo a richiederlo.
WebSocket è il nome della tecnologia che consente di creare una sorta di "canale" di comunicazione sempre aperto tra client e server. Il browser e il server rimangono perennemente collegati e possono scambiarsi messaggi, sia in una direzione che nell'altra, proprio come se fosse un canale diretto.
Con questa tecnologia il server può decidere, di propria volontà, di inviare un messaggio al client.
AJAX consente al client e al server di scambiare informazioni senza ricaricare la pagina.
Tuttavia, in AJAX, è sempre il client che chiede e il server che risponde. Il server non può decidere da solo di inviare informazioni al client.
Socket.IO è una libreria javascript per applicazioni web in realtime . Consente la comunicazione in tempo reale e bidirezionale tra client Web e server. Ha due parti: una libreria lato client che viene eseguita nel browser e una libreria lato server per node.js Entrambi i componenti hanno un'API quasi identica. Come node.js, è basato sugli eventi.
Socket.IO utilizza principalmente il protocollo websocket con il polling come opzione di fallback, pur fornendo la stessa interfaccia. Sebbene possa essere utilizzato semplicemente come wrapper per webSocket , offre molte più funzionalità, tra cui la trasmissione a più socket, la memorizzazione dei dati associati a ciascun client e l'I / O asincrono.
Node.js ti permette di eseguire codice JavaScript su Desktop, questo consente al Web Developer di avere un Sistema di sviluppo web basato completeamente su JavaScript.
Node.js è un framework per realizzare applicazioni Web in JavaScript, questo linguaggio, tipicamente utilizzato nella "client-side", è utile per la scrittura di applicazioni "server-side".
Il modello event-driven, o "programmazione ad eventi", si basa su un concetto piuttosto semplice: si lancia una azione quando accade qualcosa. Ogni azione quindi risulta asincrona a differenza dei pattern di programmazione più comune in cui una azione succede ad un'altra solo dopo che essa è stata completata.

Di seguito una semplice grafica che mostra la differenza tra una semplice applicazione PHP ed una sviluppata con Node.js:



Al contrario di quanto accade con PHP, la tecnica già ampiamente sfruttata nella programmazione JavaScript della asincronicità permette una ottimizzazione dei tempi e delle risorse molto più soddisfacente in quanto il percorso che porta al risultato non incontra blocchi e tempi morti.

scaricare node.js: https://nodejs.org/en/
Possiamo verificare il buon esito dell'installazione digitando i comandi nella finestra cmd:
node - version (per uscire ctrl c) (ricordarsi dello spazio prima e dopo il trattino)
npm version – version

Primo programma

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

var server = http.createServer(function (req, res) {

  res.writeHead(200, {'Content-Type': 'text/plain'});

  res.end('Ciao Mondo');

})

server.listen(1337, '127.0.0.1');

console.log('Server running at http://127.0.0.1:1337/');
```

salvare il codice in ciaoMondo.js
copiarlo nella directory c:\
nella finestra cmd: c:\>node ciaoMondo.js
nel browser: http://localhost:1337

NPM è un insieme di librerie scritte apposta per poter essere utilizzate con Node.js.

NPM presenta anche un comando per scaricare, ricercare ed aggiornare i pacchetti software da includere nelle nostre applicazioni.
npm install socket.io
Quando usiamo socket.io, dobbiamo sempre lavorare su due file contemporaneamente:
Il file server (es. app.js): è questo che centralizza e gestisce le connessioni dei diversi client collegati al sito.
Il file client (ad es. index.html): è questo che si connette al server e visualizza i risultati nel browser.

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

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

codice app.js

```
// Carichiamo il file index.html e mostriamo la pagina al visitatore
var server = http.createServer(function(req, res) {
  fs.readFile('./index.html', 'utf-8', function(error, content)
    {
      res.writeHead(200, {"Content-Type": "text/html"});
      res.end(content);
    });
});
// Carichiamo Socket.io
var io = require('socket.io').listen(server);
// Quando i client si connettono, lo scriviamo nella console
io.sockets.on('connection', function (socket)
  {
    console.log('Nuovo visitatore connesso!');
  });
server.listen(8080);
// il server invia un messaggio al client quando si connette
io.sockets.on('connection', function (socket)
  {
    socket.emit('message', 'Sei connesso amico!');
  });.
```

Codice index.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Socket.io</title>
</head>
<body>
  <h1>Aperta una comunicazione tramite Socket.io!</h1>
  <script src="/socket.io/socket.io.js"></script>
  <script>
  var socket = io.connect('http://localhost:8080');
  </script>
</body>
</html>
```

Aggiungere al file index.html per ascoltare i messaggi del server in arrivo

```
<script>
  var socket = io.connect('http://localhost:8080');
  socket.on('message', function(message) {
    alert('Il server dice: ' + message);
  });
```

salvare il codice in app.js
copiarlo nella directory c:\
nella finestra cmd: c:\>node app.js
nel browser: http://localhost:8080