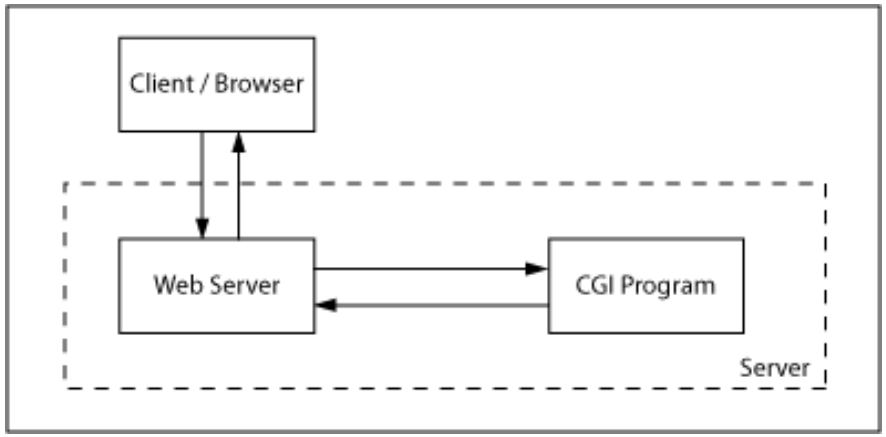
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

13. Web Backend NodeJS

Dr. Bilicki Vilmos
Szegedi Tudományegyetem
Informatikai Tanszékcsoport
Szoftverfejlesztés Tanszék

A web háttérrendszer története

- Első generáció: CGI - Common Gateway Interface
  - CGI szkript nyelvek: CGI C, CGI Perl, CGI TCL, CGI Basic
  - Egy processz minden felhasználói kérésre
  - A CGI script interpretálása (fordítás+ futtatás) a kérés feldolgozási időben történik
  - Egy felhazsnálói kérés egy processszhez kötődik, az erőforrások ide vannak allolkálva.
  - A CGI által létrehozott processz a web szerver hatókörén kívül van.

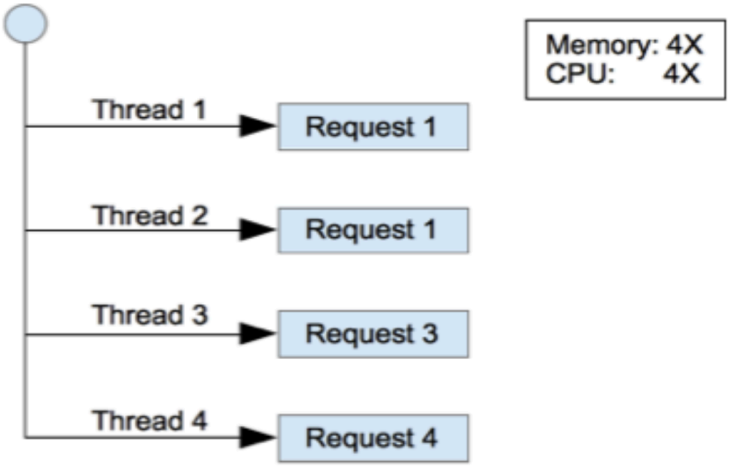


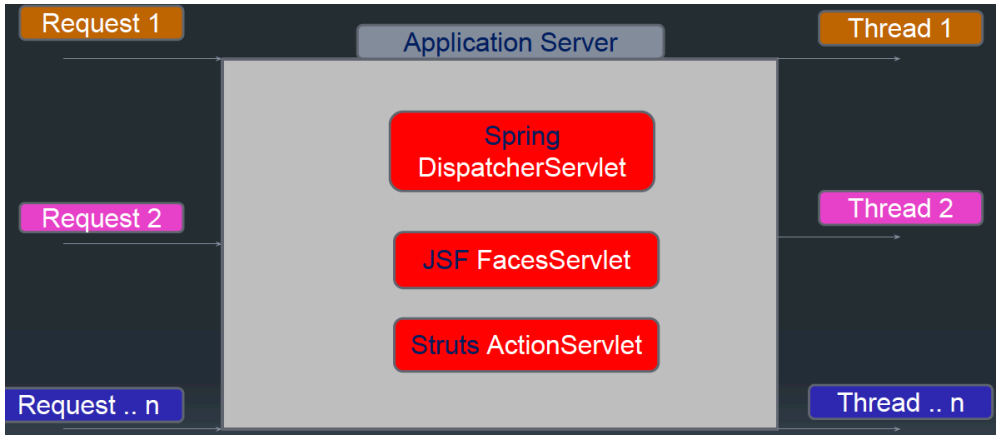
Példa

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
first_name = form.getvalue('first_name')
last_name  = form.getvalue('last_name')

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Hello - Second CGI Program</title>"
print "</head>"
print "<body>"
print "<h2>Hello %s %s</h2>" % (first_name, last_name)
print "</body>"
print "</html>"
```

A web háttérrendszer története

- Mádodik generációs: ISAPI Engines (Internet Server Application Programming Interface)
  - J2EE, .NET, LAMP, Ruby on Rails, Groovy on Grails
  - Többszálú futtatási környezet
  - Egy szál – egy felhasználói kérés
  - Ez a szál a hozzá kapcsolódó erőforrásokat zárolja az I6O várakozs idejére
  - A szálak közötti kontextus váltás költséges





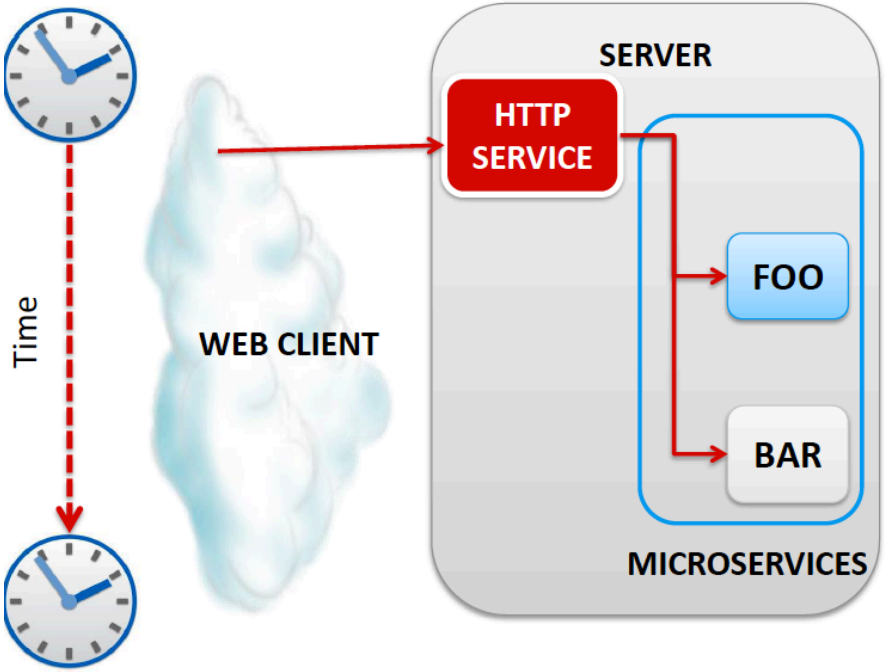
Little törvénye

$$L = \lambda W$$

- **Little törvénye a skálázhatóságra és hibatűrésre**
  - **λ** – átlagos kérés érkezési ütem
  - **W** –átlagos kiszolgálási idő
  - **L** – párhuzamosan feldolgozható kérések száma
- Pl: A mi rendszerünkben, 1000 kérés, átlagosan egy másodpercenként
  - Egynek a feldolgozása 0.5 másodperc
  - Mennyit kell lekezelni?
  - 1000*0.5 = 500
- 1. Ha tudjuk **L**-t és **W-t** ki tudjuk számítani a kiszolgálható beérkezési ütemet
- **L** megnöveléséhez, növelnünk kell a kapacitást és/vagy, le kell csökkenteni a **W**-t afeldolgozási időt/késleltetést

Skálázhatóság

- *FOO* és *BAR*, 500ms átlagos válaszidővel bír,
- A feldolgozási késleltetés 1 s, ez a mi **W** értékünk
- A webszervernek 2000 szál indítását engedélyeztük (ez a mi **L** értékünk)



Mennyi kérést tudunk másodpercenként kiszolgállni?

$\lambda = L/W = 2000/1 = 2000$

Skálázhatóság

- *L a futtató környezettől és egyébb korlátozó tényezőktől függ* (OS, HW, …)
- Ez határozza meg a kapacitást.
- **Mi határozza meg L-t?**
  - TCP: Egy szerver több tízezer kapcsolatot is kezelhet
  - BW: Több 100k vagy millió kapcsolatot kezelhet
  - RAM: 1 Mbyte-os kérés mérettel több millió kapcsolat
  - CPU: Több százezer kapcsolat (GHz-es processzorok…) Ez alapján L néhány 100k vagy millió nagyságrendő??
  - Szálak: Ez valahol 2k és 20k közötti érték. Az ***egy szál kapcsolatonkénti modellel*** a szerver < 20k kérést fog kiszolgállni
- *L = MIN(1,2,3,4,5) L teljes mértékben a szálkezeléssel korlátozott*

Skálázhatóság

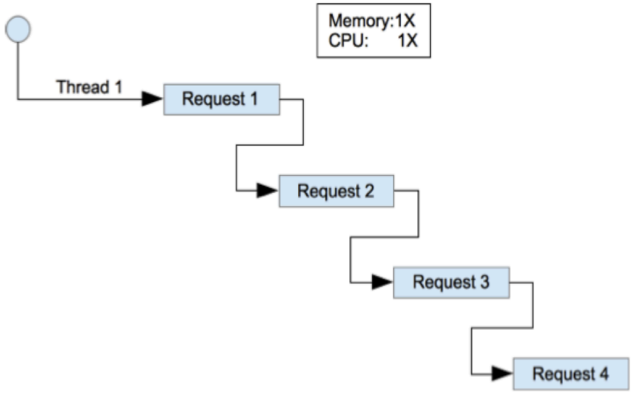
- 1. Probléma
  - A szálak közötti váltás költséges
  - A szálakhoz tartozó vermek memóriát foglalnak
  - Nem használhatunk egy-egy OS szálat kapcsolatonként
- 2. Probléma
  - Az I/O blokkolja a további futást:
    - Egy web szolgáltatás meghívása
    - Adatbázis írás/olvasás

The cost of I/O

L1-cache	3 cycles
L2-cache	14 cycles
RAM	250 cycles
Disk	41 000 000 cycles
Network	240 000 000 cycles

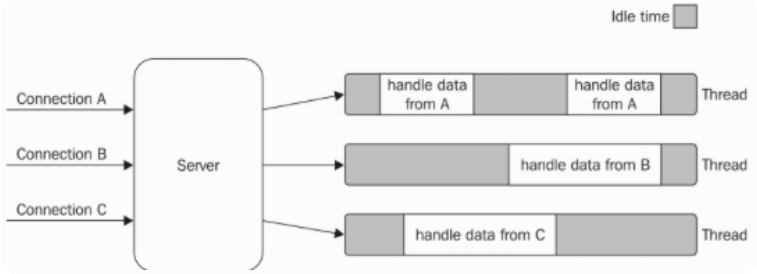
Harmadik generációs web háttér

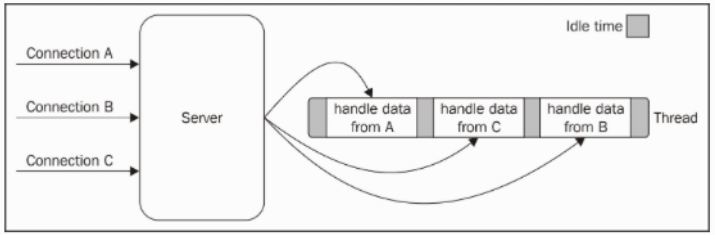
- Egy szálú futtató környezet
- Egy szál minden beérkező hívásnak
- Amíg a felhasználói hívás I/O műveletre várakozik addig a következő kérést szolgálhatja ki (aszinkron futtatás – esemény hurok).
- Az egész olyan gyors, hogy egy szál minden hívást lekezelhet
- Node.JS, Vert.x, Libevent, Twisted, EventMachine, Ngix

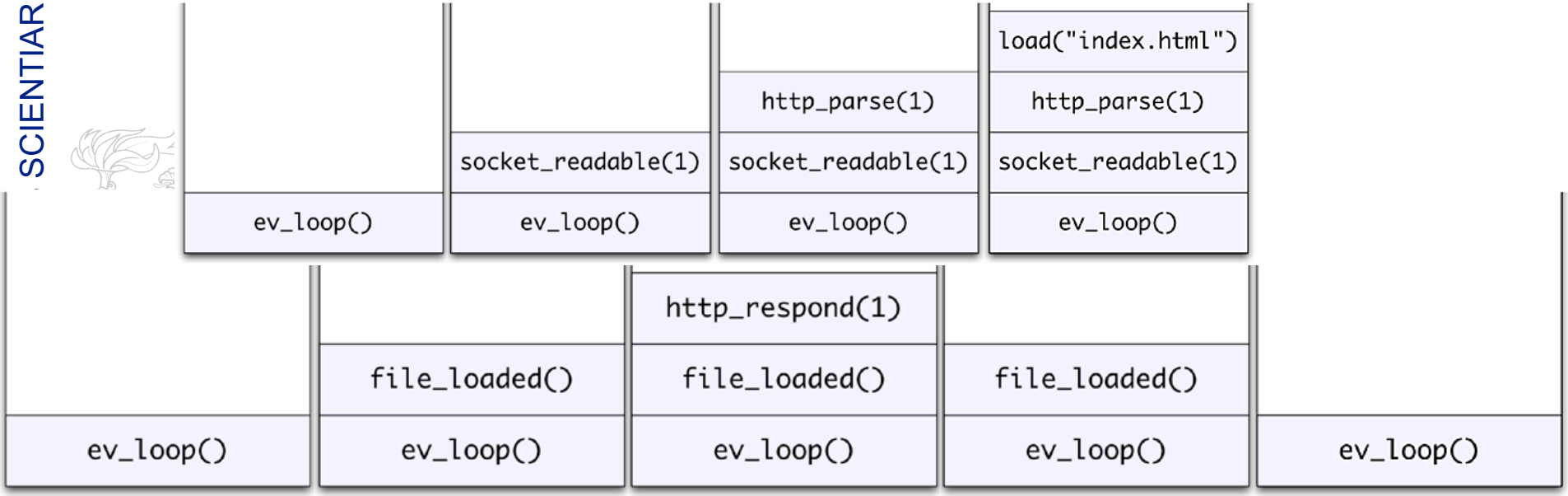


Esemény hurok példa

- “index.html” kérés érkezik be
- Elindul a betöltés és az ev_loop aludni megy
- A fájl betöltődik és elküldik a kliensnek





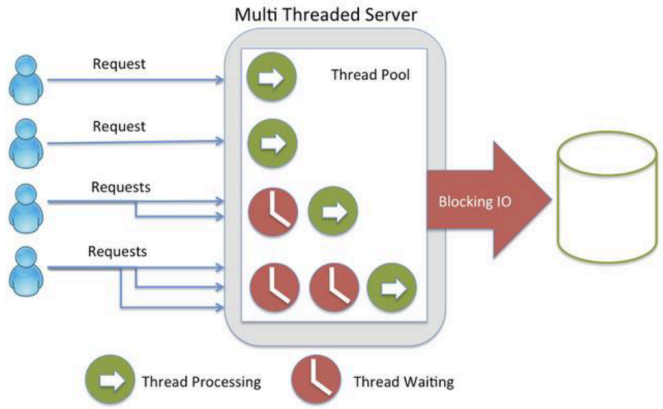


Szálak vs. Esemény alapú

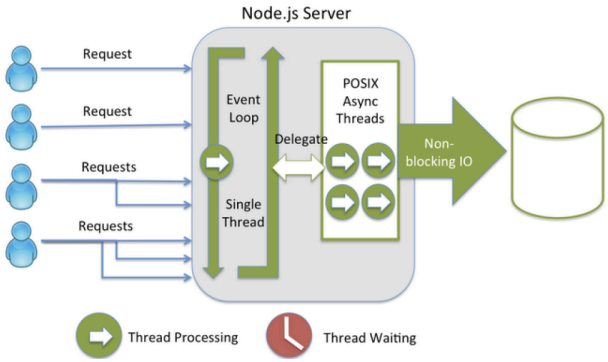
Szálak	Aszinkron esemény alapú
Zárolja az alkalmazást/ egy kérés egy hallgató-dolgozó szál	Egy szál amely folyamatosan dolgozza fel az eseményeket
Beérkező kérés modell	Várakozási sor és feldolgozás
A többszálú szerver blokkolja a kérést amely több eseményt is érinthet	Lementi az állapotot és elkezdi a következő eseményt
Kontextus váltás	Nincs kontextus váltás

2. Gen vs 3. Gen

- **Többszálú szerver architektúra** Szinkron, blokoló I/O = az I/O megvalósítás legegyszerűbb módja.
- Több szál, mivel közvetlen kapcsolat van a kapcsolatok számával.
- Nagyobb memória és CPU felhasználás a kontextus váltás miatt



- **Egyszálú szerver acrhitektúra**
- Esemény hurok (fő szál) az első lépésben és aszinkron I/O kernel szinten.
- Mivel nincs közvetlen kapcsolat a kapcsolatk száma éa s szálak száma között ezért csak a fő szál és a kernel szálak vannak az IO kiszolgálására.
- Kevesebb szál, kevesebb kontextus váltás kevesebb memória, CPU.



2nd Gen vs 3rd Gen

- *Apache Prefork processzek:*

1. *PHP kérés megérkezik, elküldi egy processznek.*
2. *A processz megkapja és elküldi a PHP-nak.*
3. *Megkap egy kép lekérés kérést és látja, hogy a processz foglalt.*
4. *A processz befejezi a PHP kérést és visszadja a választ.*
5. *A processz megkapja a képet és visszadja azt a hívónak.*

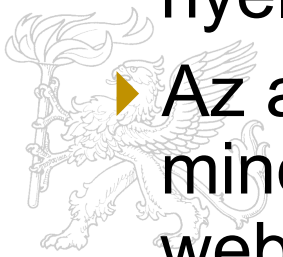
- *Nginx esemény alapú feldolgozás:*

1. *Kap egy kérést és elindít egy eseményt a processzben.*
2. *A procesz lekezeli az össz eseményt és visszatér a válasszal*

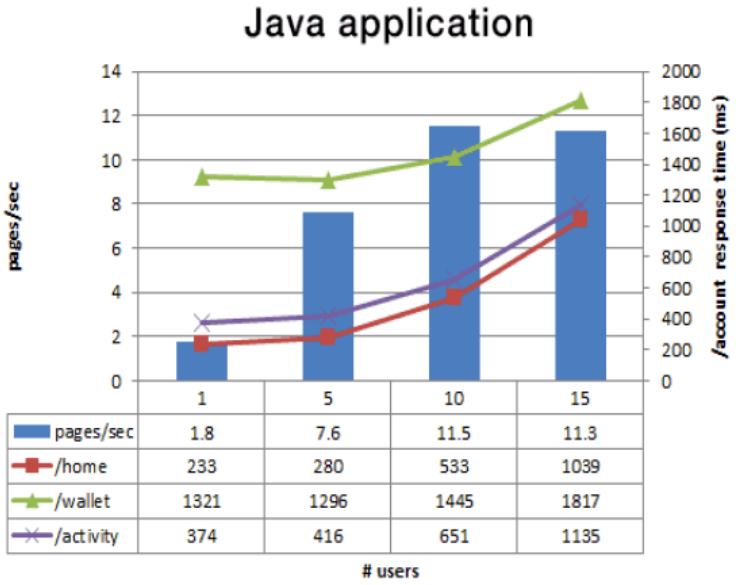
```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 1581 root      20   0  211m 178m   868 S 59.5   0.3   6:16.07 redis-server
12321 apache    20   0  239m  18m 7364 S 28.6   0.0   0:59.29 httpd
 1565 apache    20   0  239m  19m 7512 R 22.3   0.0   2:05.18 httpd
 2029 apache    20   0  239m  18m 7396 R 21.9   0.0   1:59.11 httpd
 7496 apache    20   0  239m  20m 9112 R 21.3   0.0   1:25.25 httpd
 1560 apache    20   0  240m  19m 7412 S 19.0   0.0   2:04.45 httpd
 1561 apache    20   0  240m  21m 9076 S 19.0   0.0   2:02.07 httpd
 1564 apache    20   0  242m  22m 9120 S 19.0   0.0   1:54.75 httpd
 1566 apache    20   0  240m  21m 9136 S 19.0   0.0   2:04.25 httpd
 1562 apache    20   0  239m  20m 9148 S 18.6   0.0   2:02.02 httpd
 1567 apache    20   0  240m  19m 7400 S 18.6   0.0   2:00.39 httpd
 2758 apache    20   0  240m  19m 7396 S 18.6   0.0   1:55.23 httpd
 1563 apache    20   0  242m  22m 9080 S 10.6   0.0   2:00.34 httpd
14092 nginx     20   0  106m 5964 1908 S  5.7   0.0   0:16.39 nginx
```

2nd Gen vs 3rd Gen

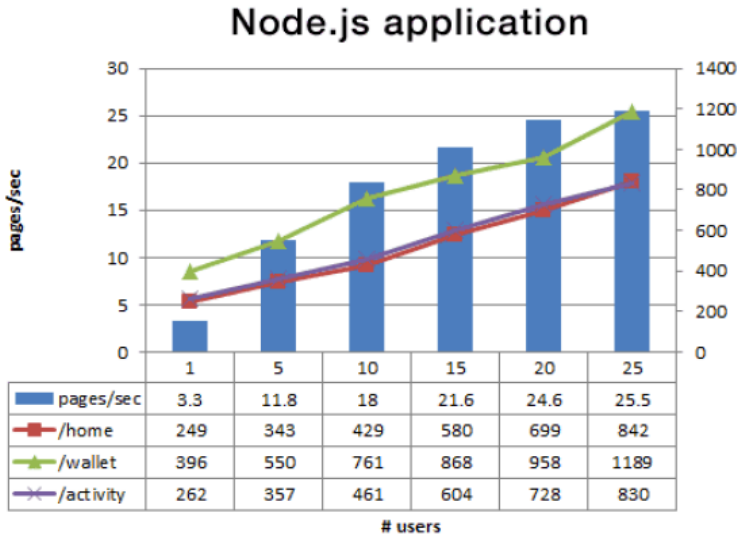
- 2013 November 22-én a PayPal mérnök csapat egy blogbejegyzést közölt arról, hogy átáll Java-ról Node.JS-re a web alkalmazások esetén
- Az első ilyen alkalmazás a témaszám áttekintő oldal volt
- Két csapatot indítottak azonos alkalmazás elkészítésére. Az egyik Java a másik JavaScript nyelvet és technológia cermet használt
- Az alkalmazás három útvonalat tartalmazott mindegyik 2-5 API hívással és az ezekre épülő weboldallal.



2nd Gen vs 3rd Gen



	1	5	10	15
pages/sec	1.8	7.6	11.5	11.3
/home	233	280	533	1039
/wallet	1321	1296	1445	1817
/activity	374	416	651	1135

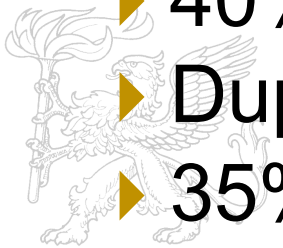


	1	5	10	15	20	25
pages/sec	3.3	11.8	18	21.6	24.6	25.5
/home	249	343	429	580	699	842
/wallet	396	550	761	868	958	1189
/activity	262	357	461	604	728	830

- **Dupla rérés kiszolgállás mint Java esetén**.
- **35% -kal** csökkent válaszidő ugyanarra az oldalra. Az oldalakat **200ms –al gyorsabban** szolgálták ki

2nd Gen vs 3rd Gen

- Teljes verem (Full-stack) mérnökök: A JavaScript használatával mindkét oldalon (frontend - backend) egy mesterséges fal lett lebontva.
- Kétszer gyorsabban fejlesztették, kevesebb emberrel
- 33%-al kevesebb kódsor
- 40%-al keveebb fájl
- Dupla kérés kiszolgálás.
- 35% -al csökkent válaszidő



NodeJS

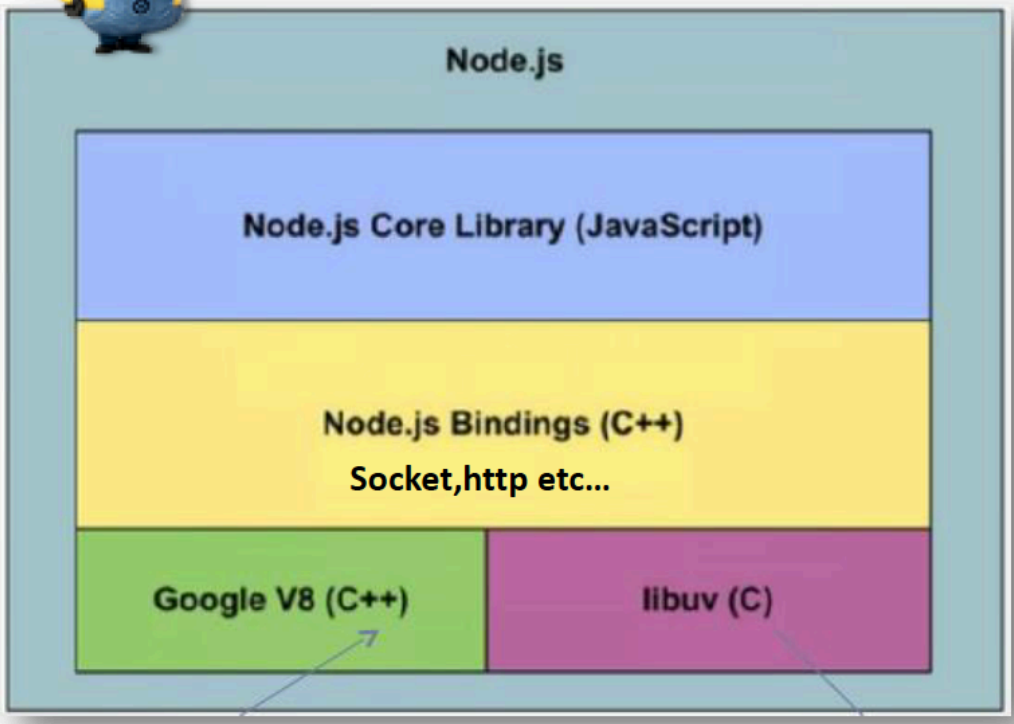
- Szerver oldali JavaScript alapú platform (Google v8 JavaScript motor)
- Esemény alapú párhuzamosság kezelés
- Egy szál és nem blokkoló I/O
- A JSON és a JavaScript természetesen kezelhetőek
- Valós idejű, szerver oldali push és kétirányú kommunikációt megvalósító alkalmazások kezelésére
- I/O műveletek párhuzamosan hajtódnak végre (a kódunk nem)

NodeJS filozófia

- Nem blokkoló I/O - minden I/O hívás egy callback-et használ, (fájl hozzáférés, hálózat kezelés, …).
- Beépített támogatás a legfontosabb protokollokra (HTTP, DNS, TLS)
- Alacsony szintű. Ne vegyünk el funkcinalitást a POSIX rétegtől.
- Minden folyam; soha sem kényszeríti az adatok tárazását (buffering)

NodeJS

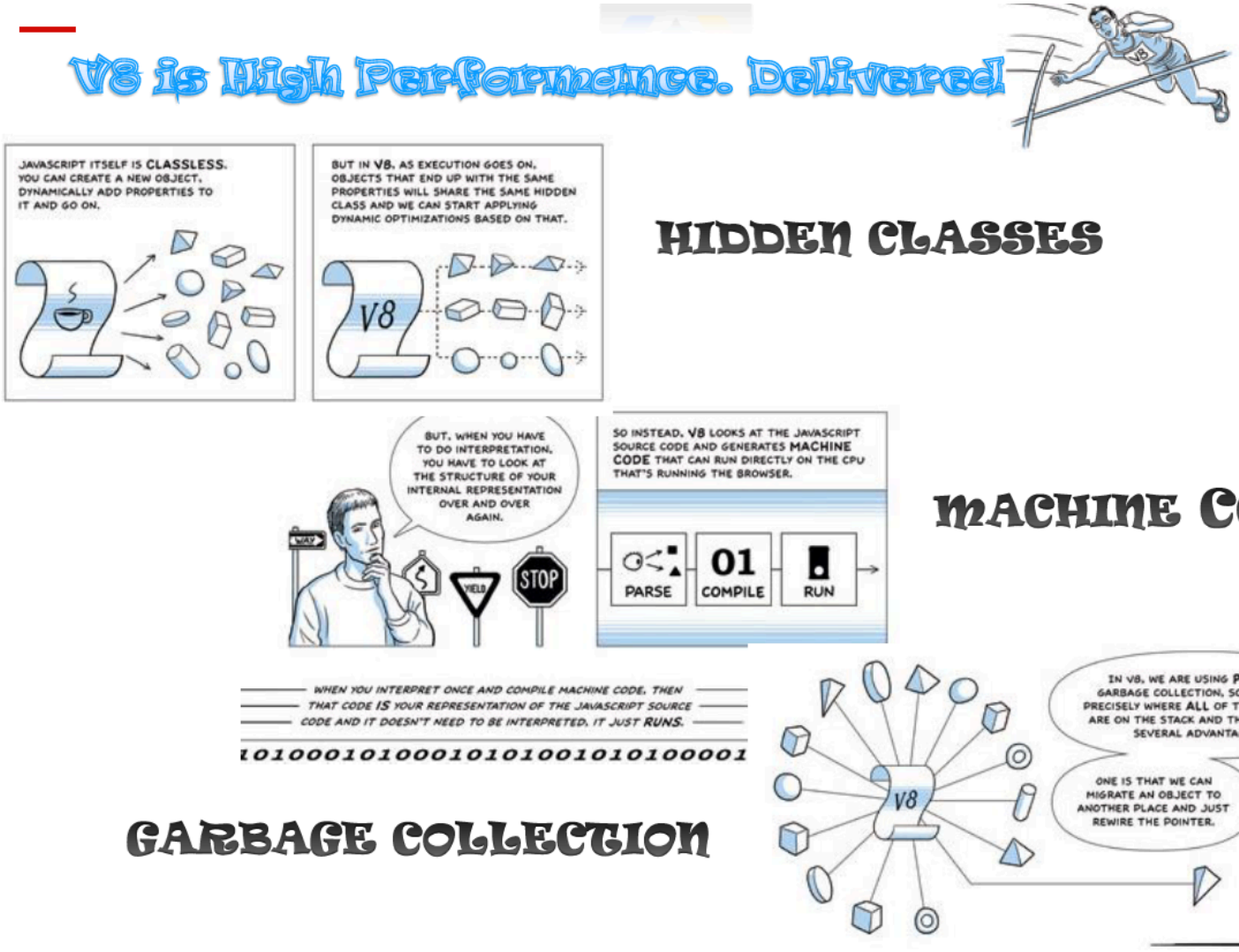
Callbacks + Polling
Epoll, POSIX AIO, Kqueue ...



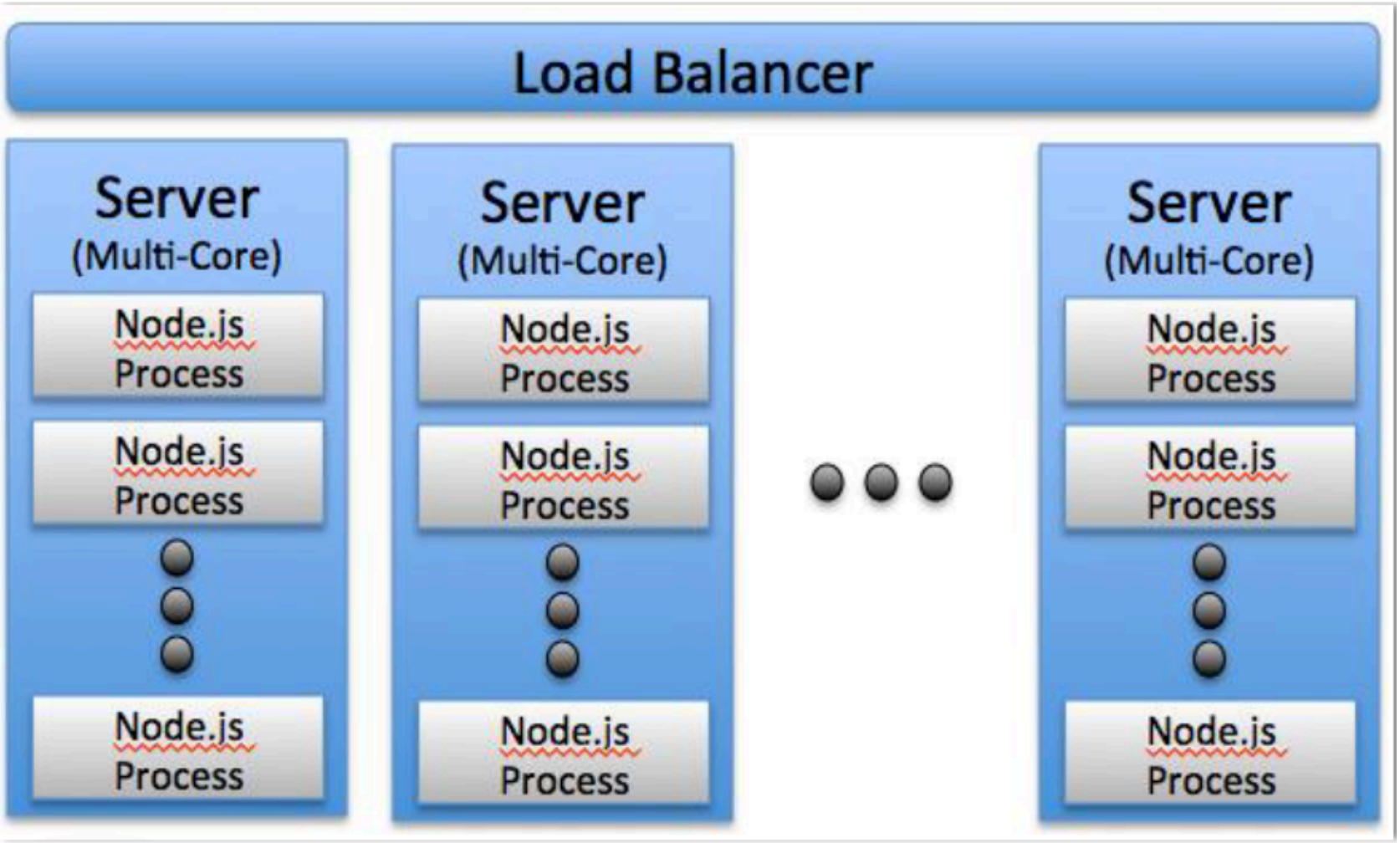
Open source JavaScript engine, platform
Optimizer, JIT, inline caching, GC

Cross-platform asychronous I/O.
Event Loop, Worker Thread Pool

Google Chrome V8



NodeJS skálázása



NodeJS területei

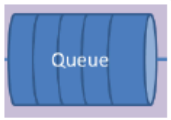
APIs
The "glue" that connects devices and browsers to data and services

Mobile
backends and fullstack JavaScript hybrid apps

Web
servers and single page apps

IoT
(Internet of Things) network connected embedded devices and sensors

CHAT 

QUEUED INPUTS 

DATA STREAMING 

PROXY 

JSON API on top of an object based DB (such as MongoDB). 

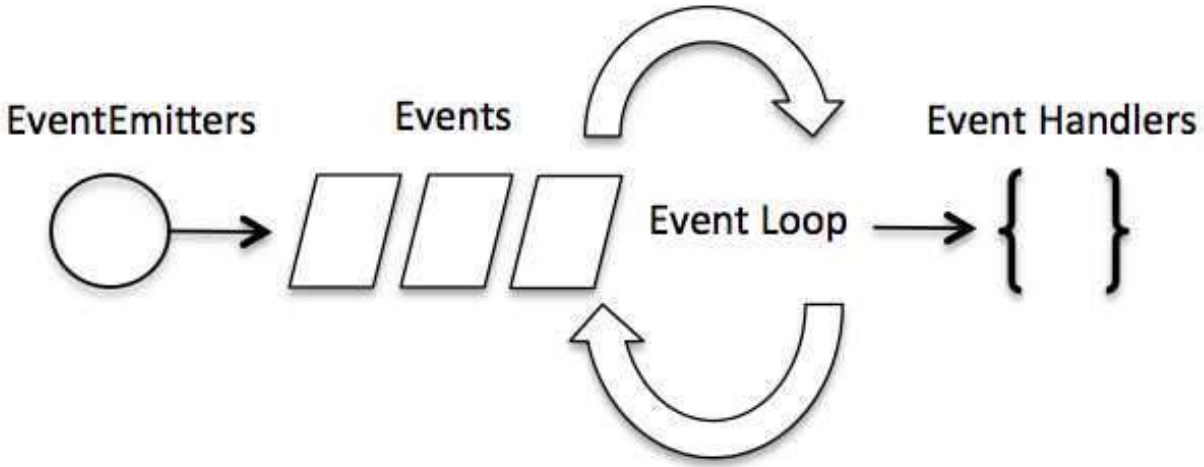
BROKERAGE - STOCK TRADE DASHBOARD 

APPLICATION MONITORING DASHBOARD 

SYSTEM MONITORING DASHBOARD 

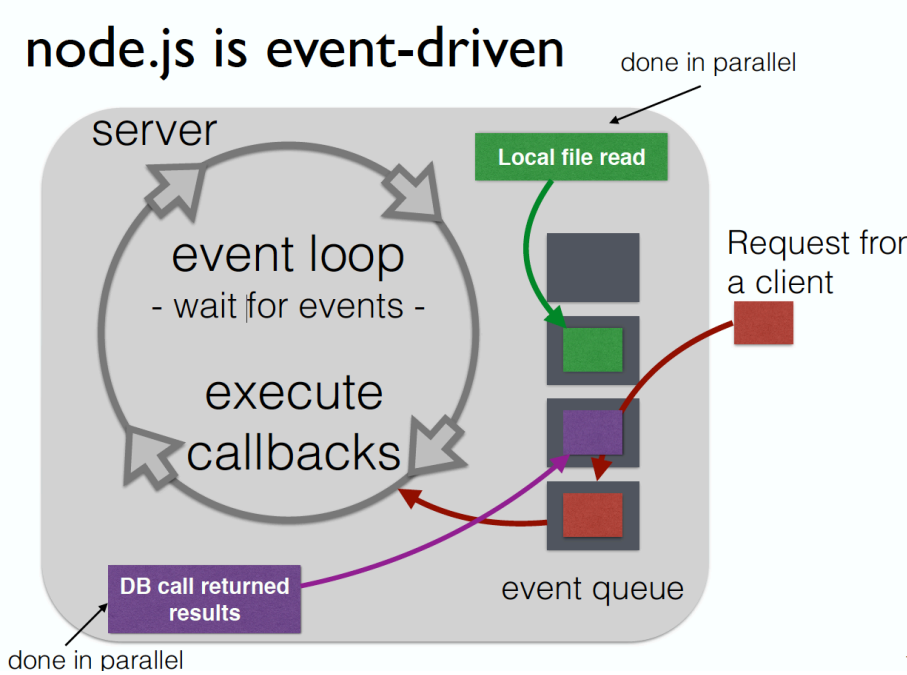
Esemény alapú programozás

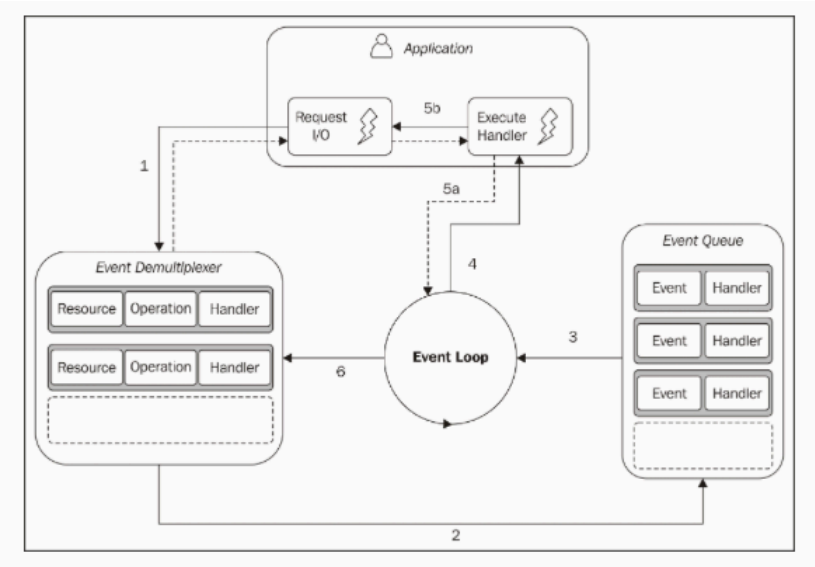
- Általában esemény hurokkal van megoldva, ahol az eseményekre callback-ek vannak regisztrálva



NodeJS

- Node.js az események hatására callback-eket hajt végre
- A fejelsztő callbackeket ír



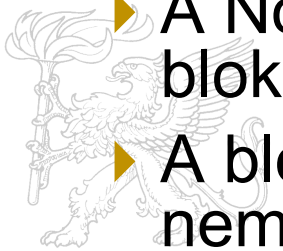


NodeJS

- Blokkoló I/O
  - Olvasás kérés
  - Kérés feldolgozása & adatbázis hozzáférés
  - Várakozás az adatbázisra és amikor megérkezett feldolgozás
  - A következő kérés feldolgozása
- Nem blokkoló I/O
  - Olvasás kérés
  - Feldolgozzuk a kérést és létrehozunk egy **callback-et** az adatbázis hozzáférésre
  - Más tevékenységek
  - Amikor a **callback** visszatér feldolgozzuk

Blokkoló vs Nem blokkoló

- **Blokkoló** amikor a JavaScript futásnak meg kell várnia a nem JavaScript művelet eredményét. Ekkor az esemény hurok nem tud továbbfutni mivel nem adja vissza a vezérlést amíg a **blokkoló** művelet fut.
- Azokat a JavaScript eljárásokat amelyek CPU intenzívek nem nevezzük blokkolóknak. A Node.JS szabvány könyvtár szinkron metódusait (libuv-t használó) nevezzük blokkolónak. A natív metóduso is lehetnek blokkolóak.
- A Node.JS az I/O könyvtárak bolokkoló és nem blokkoló változatát is megvalósítja. (Synch előtag)
- A blokkoló metódusok szinkron módon futnak míg a nem blokkolóak aszinkron módon.



NodeJS teljesítménye

- A JavaScript futtatása a Node.js-ben egy szálú. A párhuzamosság azt jelenti hogy ezen a szálon callbakc függvényeket futtatnuk..
- Például: A web szervernek 50ms a válaszideje ebből 45 ms az adatbázis amelyet aszinkron módon kezel. Amennnyiben aszinkron végrehajtást választunk akkor kérésenként 45ms időt nyerünk.
- Az esemény hurok nem azonos a szálak indításával, szálkezléssel

NodeJS nem blokkoló

```
function getAge(user_id) {
    sql = 'SELECT age FROM users WHERE id = ' + user_id;
    connection.query(sql, function(err, rows, fields) {
      if (err) throw err;
      return rows[0].age;
    });
return 0;
}
```

```
function getAge(user_id, callback) {
    sql = 'SELECT age FROM users WHERE id = ' + user_id;
    connection.query(sql, function(err, rows, fields) {
      if (err) throw err;
      var age = rows[0].age;
      callback(age);
    });
}
```

Sync vs Async

```
const fs = require('fs');
const data = fs.readFileSync('/file.md'); // blocks here until file is read
console.log(data);
// moreWork(); will run after console.log
```

```
const fs = require('fs');
fs.readFile('/file.md', (err, data) => {
  if (err) throw err;
  console.log(data);
});
// moreWork(); will run before console.log
```

Aszinkron veszélyek

```
const fs = require('fs');
fs.readFile('/file.md', (err, data) => {
  if (err) throw err;
  console.log(data);
});
fs.unlinkSync('/file.md');
```

- A fenti példában az fs.unlinkSync() valószínűleg az fs.readFile() előtt fog lefutni ami törölni fogja a file.md fájlt annk olvasása előtt

```
const fs = require('fs');
fs.readFile('/file.md', (err, data) => {
  if (err) throw err;
  console.log(data);
  fs.unlink('/file.md', (err) => {
    if (err) throw err;
  });
});
```

- A **nem blokkoló** fs.unlink() hívást az fs.readFile() callback-en belül helyzve garantálja a soros végrehajtást

NodeJS példák

```
1 const fs = require('fs');
2 fs.watch('todos.txt', function() {
3     console.log("File 'todos.txt' has just
4                  changed");
5 });
6 console.log("Now watching 'todos.txt'");
```

- Node.js modul: a funkcionalitás egy kompakt alhamaza. Ezt szeretnénk újrahasznosítani
- require() egy JavaScript objektumot ad vissza

NodeJS példa

```
 1 var http = require(“http");
 2 var server;
 3
 4 var sentCounter = 0;
 5
 6 server = http.createServer(function (req, res) {
 7     res.writeHead(200, {"Content-Type":
 8                   "text/plain"});
 9     res.end("Hello World!");
10     sentCounter++;
11     console.log(sentCounter+" HTTP responses sent
12                                in total");
13 });
14
15 var port = 2345;
16 server.listen(port);
17 console.log("Server listening on port “ + port);
```

NodeJS példa

```
1  var http = require("http");
2  var url = require('url');
3  var server;
4
5  var simpleHTTPResponder = function (req, res) {
6    var url_parts = url.parse(req.url, true);
7    if(url_parts.pathname == "/greetme") {
8       res.writeHead(200, {"Content-Type":
9                   "text/plain"});
10      var query = url_parts.query;
11      if( query["name"]!=undefined) {
12         res.end("Greetings "+query["name"]);
13      }
14      else { res.end("Greetings Anonymous"); }
15   }
16   else {
17      res.writeHead(404,{"Content-Type":
18                  "text/plain"});
19      res.end("Only /greetme is implemented.");
20   }
21 }
22
23 server = http.createServer(simpleHTTPResponder);
24 var port = process.argv[2];
25 server.listen(port);
```

if the pathname is /greetme we say ok

we can extract params from the URL

otherwise send back a 404 error

Modulok használata

```
exports.funcname = function() {
  return 'Hello World';
};
```

```
var hello = require('./hello.js');
console.log(hello.funcname()); // Print "Hello World"
```

Modulok szervezése

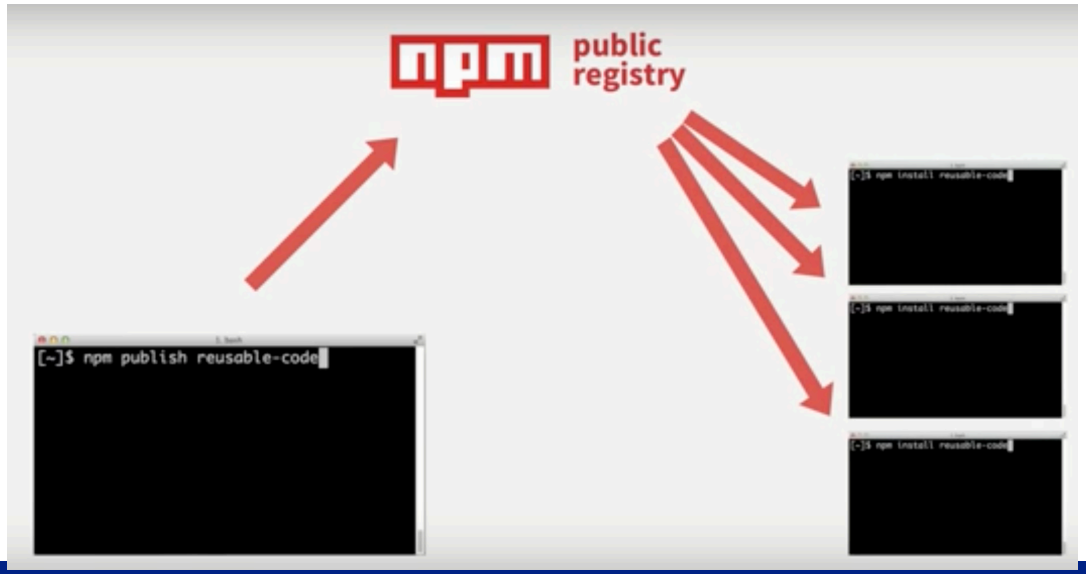
- Három módon kérhetünk modulokat require() files:
  - Relatív útvonallal: foo = require('./lib/bar.js');
  - Abszolút útvonallal: foo = require('/home/foo/lib/bar.js')
  - kereséssel: foo = require('bar')

NodeJS alap modulok

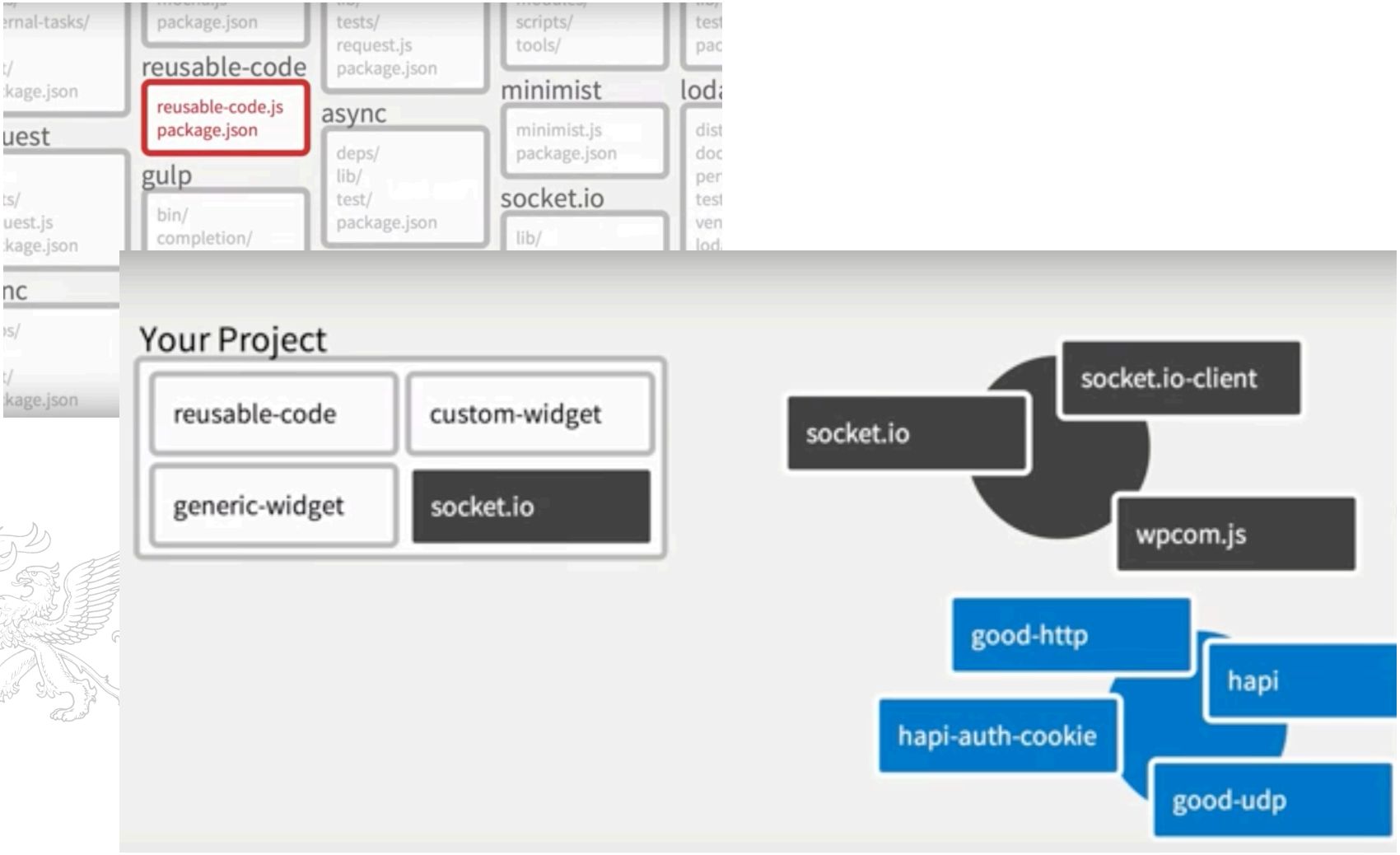
- Alapok
  - Globals STDIO Timers Modules Events Buffers Streams C/C++ Addons
- Hálózat
  - I/O HTTP HTTPS URL Query Strings Net UDP/Datagram DNS
- Fájl rendszer
  - I/O File System Path Process I/O and V8 VM Process VM Child Processes Cluster
- Terminál/konzol
  - REPL Readline TTY
- Tesztelés - hibakeresés
- Egyébb/Kripto
  - TLS/SSL _String Decoder * ZLIB * OS_

NPM

- Node Package Manager (npm) két szolgáltatása van:
  - Kereshető online modul tárak search.nodejs.org
  - Parancssori csomagkezelő verzió és függőség kezeléssel

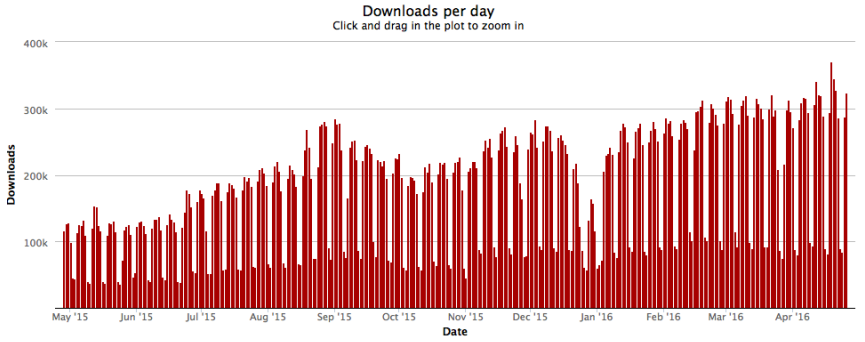


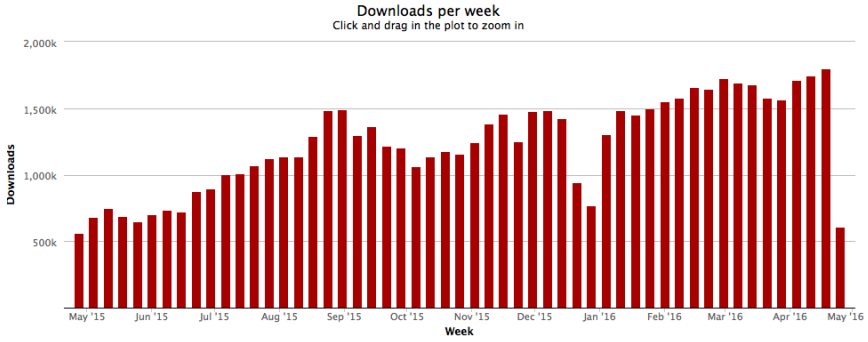
NPM motiváció

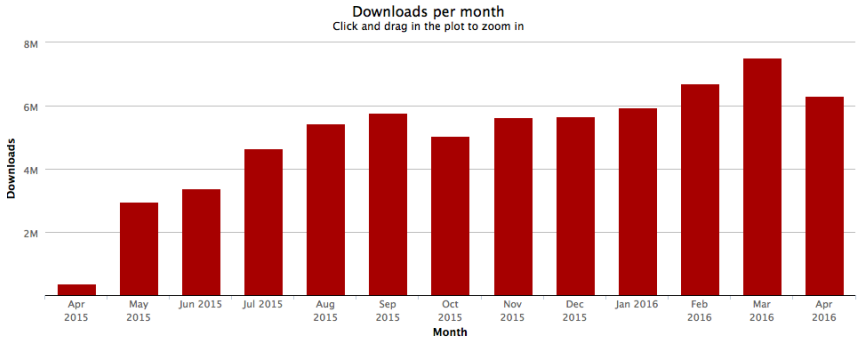


NPM közösség

Több mint 211,000 regsiztrált npm felhasználó, és közülük 73,000 publikált csomagot.







NPM

- Node.js Package Management (NPM)
  - Node.js mouldok számára csomag kezelő
  - $ npm init a CMD (Win) vagy a Terminal (MAC/Linux) interfészen
  - Létrehoz egy üres Node.js projektet a package.json fájlllal

```
$ npm init
//enter package details
name: "NPM demos"
version: 0.0.1
description: "Demos for the NPM package management"
entry point: main.js
test command: test
git repository: http://github.com/user/repository-name
keywords: npm, package management
author: doncho.minkov@telerik.com
license: BSD-2-Clause
```

NPM

- Modulok telepítése
  - $ npm install package-name [--save-dev]
    - Telepíti az adott csomagot
    - --save-dev hozzáadja a függőséget a package.json fájlhoz

```
$ npm install express --save-dev
$ npm install jade --save-dev
```

- A projekt futása előtt
  - $ npm install
  - Minden hiányzó csomagot letölt a package.json fájl alapján

```
$ npm install
```

Globális vs Lokális telepítés

- Az npm alapértelmezésben a függségeket lokális módba telepíti
  - A Node alkalmazás könyvtárában lévő node_modules könyvtárba.
  - A lokális modulok a require() függvény segítségével érhetőek el.
- A globálisan telepített csomagok a rendszer könyvtárban tárolódnak.
  - Ezeket a CLI-n keersztül használhatjuk
  - Nem tudjuk közvetlenül betölteni a require() segítségével

Példa

```
$ npm install express -g
```

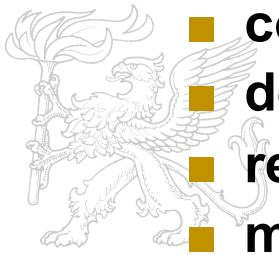
```
$ npm ls -g
```



```
express@4.12.2 /usr/lib/node_modules/express
├── merge-descriptors@1.0.0
├── utils-merge@1.0.0
├── cookie-signature@1.0.6
├── methods@1.1.1
├── fresh@0.2.4
├── cookie@0.1.2
├── escape-html@1.0.1
├── range-parser@1.0.2
├── content-type@1.0.1
├── finalhandler@0.3.3
├── vary@1.0.0
├── parseurl@1.3.0
├── content-disposition@0.5.0
├── path-to-regexp@0.1.3
├── depd@1.0.0
├── qs@2.3.3
├── on-finished@2.2.0 (ee-first@1.1.0)
├── etag@1.5.1 (crc@3.2.1)
├── debug@2.1.3 (ms@0.7.0)
├── proxy-addr@1.0.7 (forwarded@0.1.0, ipaddr.js@0.1.9)
├── send@0.12.1 (destroy@1.0.3, ms@0.7.0, mime@1.3.4)
├── serve-static@1.9.2 (send@0.12.2)
├── accepts@1.2.5 (negotiator@0.5.1, mime-types@2.0.10)
└── type-is@1.6.1 (media-typer@0.3.0, mime-types@2.0.10)
```

Package.json

- Package.json a Node alkalamzás/modul könyvtárában találhatóapplication/module
- A csomag tulajdonságait írja le
- Attribútumok:
  - **name** – a csomag neve
  - **version** – a csomag verziója
  - **description** – a csomag leírása
  - **homepage** – a csomag honlapja
  - **author** - a csomag szerzője
  - **contributors** – a részvevők listája
  - **dependencies** – a függőségek listája
  - **repository** – a csomag tároló típusa és a hivatkozása
  - **main** – a csomag belépési pontja
  - **keywords** - kulcssszavak



Node.js - REPL Terminal

- Ezekt biztosítja.
  - **Read** – Olvassa és értelmezi a felhasználó által begépelt szöveget.
  - **Eval** – Kiértékeli azt
  - **Print** – Kiírja az eredményt
  - **Loop** – A fentieket ciklikusan végrehajtja amíg a felhasználó **ctrl-c** nem nyom kétszer


```
$ node
> 1 + 3
4
> 1 + ( 2 * 3 ) - 4
3
>
```

Node vs. Böngésző

- Modul kezelés, verziózás
- npm install xxx
- Webpack,
- A legtöbb Node könyvtár böngésző sepcifikus verzióját is biztosítja
- Webpack --watch

Express

- Minimális és flexibilis Node.js web alkalmazás keretrendszer.
- Az Express modul a http szolgáltatásra építve nyújt magasabb szintű szolgáltatásokat (statikus weboldal kiszolgállása, …)
- Express alap képességek:
  - Köztesréteg a HTTP kérések kiszolgálására.
  - Forgalimirányítótáblát biztosít az URL alapú tartalom irányításhoz
  - Sablon alapú dinamikus környezeteket tud támogatni

Express

- Telepítése.

```
$ npm install express --save
```

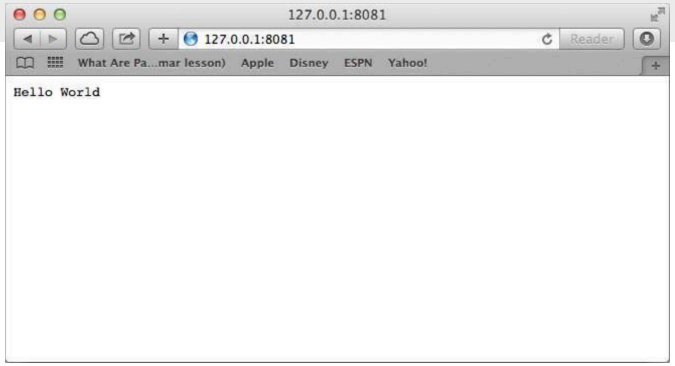
- **body-parser** - A JSON, Raw, Text és URL encoded űrlap adatok kezelésére.
- **cookie-parser** – Sütik kezelése (fejléc írás/olvasás).
- **multer** - multipart/form-data kezelése


```
$ npm install body-parser --save
$ npm install cookie-parser --save
$ npm install multer --save
```

Hello world Example

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
   res.send('Hello World');
})

var server = app.listen(8081, function () {

  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)

})
```

```
$ node server.js
```



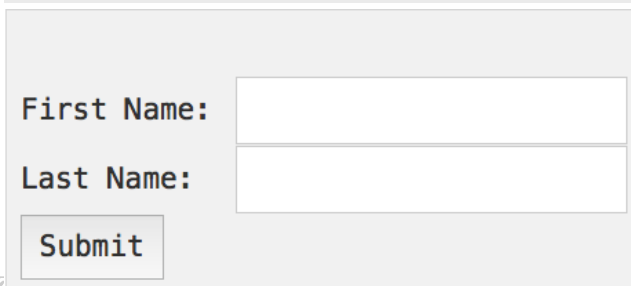
Forgalomirányítás

- URL/HTTP/GET /POST alapú forgalimrányítás


```
var express = require('express');
var app = express();

// This responds with "Hello World" on the homepage
app.get('/', function (req, res) {
   console.log("Got a GET request for the homepage");
   res.send('Hello GET');
})


// This responds a POST request for the homepage
app.post('/', function (req, res) {
   console.log("Got a POST request for the homepage");
   res.send('Hello POST');
})

// This responds a DELETE request for the /del_user page.
app.delete('/del_user', function (req, res) {
   console.log("Got a DELETE request for /del_user");
   res.send('Hello DELETE');
})

// This responds a GET request for the /list_user page.
app.get('/list_user', function (req, res) {
   console.log("Got a GET request for /list_user");
   res.send('Page Listing');
})

// This responds a GET request for abcd, abxcd, ab123cd, and so on
app.get('/ab*cd', function(req, res) {
   console.log("Got a GET request for /ab*cd");
   res.send('Page Pattern Match');
})


var server = app.listen(8081, function () {

  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)

})
```

GET kezelése

```
<html>
<body>
<form action="http://127.0.0.1:8081/process_get" method="GET">
First Name: <input type="text" name="first_name">  <br>

Last Name: <input type="text" name="last_name">
<input type="submit" value="Submit">
</form>
</body>
</html>
```




```
var express = require('express');
var app = express();

app.use(express.static('public'));

app.get('/index.htm', function (req, res) {
   res.sendFile( __dirname + "/" + "index.htm" );
})

app.get('/process_get', function (req, res) {

   // Prepare output in JSON format
   response = {
       first_name:req.query.first_name,
       last_name:req.query.last_name
   };
   console.log(response);
   res.end(JSON.stringify(response));
})

var server = app.listen(8081, function () {

  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)

})
```

POST kezelése

```
<html>
<body>
<form action="http://127.0.0.1:8081/process_post" method="POST">
First Name: <input type="text" name="first_name">  <br>

Last Name: <input type="text" name="last_name">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

First Name:

Last Name:

Submit


```
var express = require('express');
var app = express();
var bodyParser = require('body-parser');

// Create application/x-www-form-urlencoded parser
var urlencodedParser = bodyParser.urlencoded({ extended: false })

app.use(express.static('public'));

app.get('/index.htm', function (req, res) {
   res.sendFile( __dirname + "/" + "index.htm" );
})

app.post('/process_post', urlencodedParser, function (req, res) {

   // Prepare output in JSON format
   response = {
       first_name:req.body.first_name,
       last_name:req.body.last_name
   };
   console.log(response);
   res.end(JSON.stringify(response));
})

var server = app.listen(8081, function () {

  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)

})
```

Fájl feltöltés

```
<html>
<head>
<title>File Uploading Form</title>
</head>
<body>
<h3>File Upload:</h3>
Select a file to upload: <br />
<form action="http://127.0.0.1:8081/file_upload" method="POST"
      enctype="multipart/form-data">
<input type="file" name="file" size="50" />
<br />
<input type="submit" value="Upload File" />
</form>
</body>
</html>
```

File Upload:

Select a file to upload:

Fájl kiválasztása nincs kiválasztva fájl

Upload File

NOTE: This is just dummy form and would not work, but it must work at your server.

```
var express = require('express');
var app = express();
var fs = require("fs");

var bodyParser = require('body-parser');
var multer  = require('multer');

app.use(express.static('public'));
app.use(bodyParser.urlencoded({ extended: false }));
app.use(multer({ dest: '/tmp/'}));

app.get('/index.htm', function (req, res) {
   res.sendFile( __dirname + "/" + "index.htm" );
})

app.post('/file_upload', function (req, res) {

   console.log(req.files.file.name);
   console.log(req.files.file.path);
   console.log(req.files.file.type);

   var file = __dirname + "/" + req.files.file.name;
   fs.readFile( req.files.file.path, function (err, data) {
        fs.writeFile(file, data, function (err) {
         if( err ){
              console.log( err );
         }else{
               response = {
                   message:'File uploaded successfully',
                   filename:req.files.file.name
               };
          }
          console.log( response );
          res.end( JSON.stringify( response ) );
       });
   });
})

var server = app.listen(8081, function () {

  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)

})
```

RESTful API

- Az alábbi 4 HTTP metódus van általános alkalmazva a REST megoldásokban:
- **GET** – Erőforrás olvasása.
- **PUT** – Erőforrások létrehozása
- **DELETE** – Erőforrások törlése
- **POST** – Erőforrások létrehozása és módosítása

REST minta

```
{
   "user1" : {
      "name" : "mahesh",
          "password" : "password1",
          "profession" : "teacher",
          "id": 1
   },
   "user2" : {
      "name" : "suresh",
          "password" : "password2",
          "profession" : "librarian",
          "id": 2
   },
   "user3" : {
      "name" : "ramesh",
          "password" : "password3",
          "profession" : "clerk",
          "id": 3
   }
}
```

S. N.	URI	HTTP Method	POST body	Result
1	listUsers	GET	empty	Show list of all the users.
2	addUser	POST	JSON String	Add details of new user.
3	deleteUser	DELETE	JSON String	Delete an existing user.
4	:id	GET	empty	Show details of a user.

```
var express = require('express');
var app = express();
var fs = require("fs");

app.get('/listUsers', function (req, res) {
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
       console.log( data );
       res.end( data );
   });
})

var server = app.listen(8081, function () {

  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)

})
```

```
var express = require('express');
var app = express();
var fs = require("fs");

app.get('/:id', function (req, res) {
   // First read existing users.
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
       users = JSON.parse( data );
       var user = users["user" + req.params.id]
       console.log( user );
       res.end( JSON.stringify(user));
   });
})

var server = app.listen(8081, function () {

  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)

})
```

Use of template engines

```
$ npm install pug --save
```

```
app.set('view engine', 'pug');
```

```
html
  head
    title= title
  body
    h1= message
```

```
app.get('/', function (req, res) {
  res.render('index', { title: 'Hey', message: 'Hello there!'});
});
```

Jade Template Engine

```
doctype html
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript').
      if (foo) {
         bar(1 + 5)
      }
  body
    h1 Jade - node template engine
    #container.col
      if youAreUsingJade
        p You are amazing
      else
        p Get on it!
      p.
        Jade is a terse and simple
        templating language with a
        strong focus on performance
        and powerful features.
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Jade</title>
    <script type="text/javascript">
      if (foo) {
         bar(1 + 5)
      }
    </script>
  </head>
  <body>
    <h1>Jade - node template engine</h1>
    <div id="container" class="col">
      <p>You are amazing</p>
      <p>
        Jade is a terse and simple
        templating language with a
        strong focus on performance
        and powerful features.
      </p>
    </div>
  </body>
</html>
```