https://www.tutorialspoint.com/nodejs/index.htm

Node.js

A Node.js egy JavaScript alapú platform, amelyet a Google Chrome JavaScript V8 motorjára építettek. I / O intenzív webes alkalmazások, például video streaming webhelyek, egyoldalas alkalmazások és egyéb webes alkalmazások fejlesztésére használják. A Node.js nyílt forráskódú, teljesen ingyenes, és a fejlesztők ezrei használják szerte a világon.

Mi a Node.js?

A Node.js egy szerveroldali platform, amelyet a Google Chrome JavaScript motorjára (V8 Engine) építettek. A Node.js-t Ryan Dahl fejlesztette ki 2009-ben. A Node.js hivatalos dokumentációjában szereplő meghatározása a következő:

A Node.js egy olyan platform, amely a JavaScript futtatójára épül, a gyors és skálázható hálózati alkalmazások egyszerű létrehozását teszi lehetővé. A Node.js egy eseményvezérelt, nem blokkoló I / O modellt (aszinkron) használ, amely könnyű és hatékony, tökéletesen alkalmazható adatintenzív valósidejű alkalmazásokhoz, amelyek elosztott eszközökön futnak.

A Node.js alkalmazások JavaScript nyelven íródnak. A Node.js különféle JavaScript modulokból gazdag könyvtárat is kínál, amely nagymértékben leegyszerűsíti a webes alkalmazások fejlesztését a Node.js használatával.

Node.js = Runtime Environment + JavaScript library

A Node.js tulajdonságai

Az alábbiakban bemutatjuk azokat a fontos funkciókat, amelyek miatt a Node.js a szoftver-achitektek első választása.

Aszinkron és eseményvezérelt - A Node.js könyvtárban az összes API aszinkron, vagyis nem blokkolódik. Ez lényegében azt jelenti, hogy a Node.js alapú szerver soha nem várja meg az API-t az adatok visszaadására. A szerver egy API meghívása után a következő API-ra lép, és a Node.js események értesítési mechanizmusa segít a szervernek az előző API-hívás válaszának megszerzésében.

Nagyon gyors - A Google Chrome V8 JavaScript motorjára épülve a Node.js könyvtár nagyon gyors a kód végrehajtásában.

Egyszálú, de erősen skálázható - a Node.js egyszálú modellt használ. Az eseménymechanizmus segíti a szervert a blokkolás nélküli reagálásban, és a kiszolgálót erősen skálázhatóvá teszi, szemben a hagyományos szerverekkel, amelyek bizonyos számú szálakat hoznak létre a kérelmek kezelésére. A Node.js egyetlen szálú programot használ, és ugyanaz a program sokkal nagyobb számú kérést képes kiszolgálni, mint a hagyományos szerverek, például az Apache HTTP Server.

Node.js – Első alkalmazás

Mielőtt létrehoznánk egy valódi "Hello, world!" alkalmazást, nézzük meg egy Node.js alkalmazás összetevőit. A Node.js alkalmazás a következő három fontos összetevőből áll:

- Szükséges modulok importálása
- Szerver létrehozása
- Kérés olvasása és válasz

A Node.js alkalmazás létrehozása

1. lépés - A szükséges modul importálása

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

2. lépés - Szerver létrehozása

A létrehozott http-példányt használjuk, és a http.createServer () metódust hívjuk meg egy szerver létrehozására, majd a 8081-es porthoz kötjük. Egy kérés és válasz paraméterű függvénynek adjuk át.

```
http.createServer(function (request, response) {
   // Send the HTTP header
   // HTTP Status: 200 : OK
   // Content Type: text/plain
   response.writeHead(200, {'Content-Type': 'text/plain'});

   // Send the response body as "Hello World"
   response.end('Hello World\n');
}).listen(8081);

// Console will print the message
console.log('Server running at http://127.0.0.1:8081/');
```

3. lépés - Kérés és válasz tesztelése

Tegyük az 1. és a 2. lépést egy main.js nevű fájlba, és indítsuk el a HTTP szervert az alább látható módon –

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

http.createServer(function (request, response) {
   // Send the HTTP header
   // HTTP Status: 200 : OK
   // Content Type: text/plain
   response.writeHead(200, {'Content-Type': 'text/plain'});
```

```
   // Send the response body as "Hello World"
   response.end('Hello World\n');
}).listen(8081);

// Console will print the message
console.log('Server running at http://127.0.0.1:8081/');
```

A main.js szerver elindítása:

```
$ node main.js
```

Node.js - REPL Terminal

A REPL a Read Eval Print Loop kifejezést jelenti, és egy számítógépes környezetet reprezentál, mint például a Windows konzol vagy az Unix / Linux shell, ahol parancsot ad, és a rendszer interaktív módban válaszol a kimenettel. A Node.js vagy a Node csomag REPL környezettel kerül forgalomba. A következő feladatokat hajtja végre –

Read – Beolvassa a felhasználó bemeneteit, elemzi a bemenetet a JavaScript adatszerkezetében, és tárolja a memóriában.

Eval - elvégzi és kiértékeli az a parancsot.

Print - Az eredmény kiírása.

Loop - addig ismétli a fenti parancsot, amíg a felhasználó kétszer megnyomja a ctrl-c billentyűt.

A Node REPL szolgáltatása nagyon hasznos a Node.js kódokkal való kísérletezés és a JavaScript kódok hibakeresése során

REPL indítása

```
$ node

Egyszerű kifejezések:

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

Változók használata:

```
$ node
> x = 10
```

```
10
> var y = 10
undefined
> x + y
20
> console.log("Hello World")
Hello World
undefined
```

Többsoros kifejezés:

```
$ node
> var x = 0
undefined
> do {
   ... x++;
   ... console.log("x: " + x);
   ... }
while ( x < 5 );
x: 1
x: 2
x: 3
x: 4
x: 5
undefined
>
```

Underscore variable: Az utolsó eredményt kapjuk meg vele:

```
$ node
> var x = 10
undefined
> var y = 20
undefined
> x + y
30
> var sum = _
undefined
> console.log(sum)
30
undefined
>
```

REPL parancsok

ctrl + c - az aktuális parancs lezárása.

ctrl + c kétszer -bezárás.

ctrl + d -bezárás.

Fel / le gombok - a parancselőzmények.

.help - az összes parancs listája.

.break - kilépés a többsoros kifejezésből.

.clear - kilépés a többsoros kifejezésből.

.save fájlnév - az aktuális Node REPL munkamenet mentése fájlba.

.load fájlnév - fájltartalom betöltése az aktuális Node REPL munkamenetben.

Node.js - NPM

A Node Package Manager (NPM) két fő funkciót nyújt -

A node.js csomagok / modulok online tárolói, amelyek a search.nodejs.org oldalon találhatóak meg.

Parancssori segédprogram a Node.js csomagok telepítéséhez, a verziókezeléshez és a Node.js csomagok függőségi kezeléséhez.

Az NPM a v0.6.3 verzió után a Node.js telepíthető csomagokkal kerül szállításra. Ennek ellenőrzéséhez nyissa meg a konzolt, írja be a következő parancsot, és

```
$ npm - version
```

2.7.1

Könnyű frissíteni a legújabb verzióra:

```
$ sudo npm install npm –g
```

Modulok telepítése az NPM használatával

```
$ npm install  <Modul neve>
```

Például express telepítése,

```
$ npm install express
```

Ezt a modult használhatjuk a js fájljában:

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

Globális vs. helyi telepítés

Alapértelmezés szerint az NPM lokálisan telepíti a függőségeket. Ekkor a függőségek abban a mappában találhatóak, ahol a Node alkalmazás jelen van. A lokálisan telepített csomagok a require() függvény segítségével érhetők el. Lokálisan telepített dependency esetén a node_modules mappába kerülnek a dependency-k.

Az npm ls parancs felsorolja az összes lokálisan telepített modult.

A globálisan telepített csomagokat / függőségeket a rendszerkönyvtárban tárolják. Az ilyen függőségek felhasználhatók bármely Node.js CLI (Command Line Interface) függvényében, de közvetlenül nem importálhatók a request () használatával a Node alkalmazásban. Most próbáljuk telepíteni az expressz modult a globális telepítéssel.

```
$ npm install express –g
```

A következő parancs segítségével ellenőrizhetjük az összes globálisan telepített modult -

```
$ npm ls –g
```

A package.json használata

A package.json megtalálható minden node alkalmazás / modul gyökérkönyvtárában, és egy csomag tulajdonságainak meghatározására szolgál. Nyissuk meg a node_modules / express /

```
{
   "name": "express",
      "description": "Fast, unopinionated, minimalist web
framework",
      "version": "4.11.2",
      "author": {

         "name": "TJ Holowaychuk",
         "email": "tj@vision-media.ca"
      },

   "contributors": [{
      "name": "Aaron Heckmann",
      "email": "aaron.heckmann+github@gmail.com"
   },

   {
      "name": "Ciaran Jessup",
      "email": "ciaranj@gmail.com"
   },

   {
      "name": "Douglas Christopher Wilson",
      "email": "doug@somethingdoug.com"
   },

   {
      "name": "Guillermo Rauch",
      "email": "rauchg@gmail.com"
   },

   {
      "name": "Jonathan Ong",
      "email": "me@jongleberry.com"
```

```
  },

  {
     "name": "Roman Shtylman",
     "email": "shtylman+expressjs@gmail.com"
  },

  {
     "name": "Young Jae Sim",
     "email": "hanul@hanul.me"
  } ],

  "license": "MIT", "repository": {
     "type": "git",
     "url": "https://github.com/strongloop/express"
  },

  "homepage": "https://expressjs.com/", "keywords": [
     "express",
     "framework",
     "sinatra",
     "web",
     "rest",
     "restful",
     "router",
     "app",
     "api"
  ],

  "dependencies": {
     "accepts": "~1.2.3",
     "content-disposition": "0.5.0",
     "cookie-signature": "1.0.5",
     "debug": "~2.1.1",
     "depd": "~1.0.0",
     "escape-html": "1.0.1",
     "etag": "~1.5.1",
     "finalhandler": "0.3.3",
     "fresh": "0.2.4",
     "media-typer": "0.3.0",
     "methods": "~1.1.1",
     "on-finished": "~2.2.0",
     "parseurl": "~1.3.0",
     "path-to-regexp": "0.1.3",
     "proxy-addr": "~1.0.6",
     "qs": "2.3.3",
     "range-parser": "~1.0.2",
     "send": "0.11.1",
     "serve-static": "~1.8.1",
     "type-is": "~1.5.6",
     "vary": "~1.0.0",
     "cookie": "0.1.2",
     "merge-descriptors": "0.0.2",
```

```
      "utils-merge": "1.0.0"
   },

   "devDependencies": {
      "after": "0.8.1",
      "ejs": "2.1.4",
      "istanbul": "0.3.5",
      "marked": "0.3.3",
      "mocha": "~2.1.0",
      "should": "~4.6.2",
      "supertest": "~0.15.0",
      "hjs": "~0.0.6",
      "body-parser": "~1.11.0",
      "connect-redis": "~2.2.0",
      "cookie-parser": "~1.3.3",
      "express-session": "~1.10.2",
      "jade": "~1.9.1",
      "method-override": "~2.3.1",
      "morgan": "~1.5.1",
      "multiparty": "~4.1.1",
      "vhost": "~3.0.0"
   },

   "engines": {
      "node": ">= 0.10.0"
   },

   "files": [
      "LICENSE",
      "History.md",
      "Readme.md",
      "index.js",
      "lib/"
   ],

   "scripts": {
      "test": "mocha --require test/support/env
         --reporter spec --bail --check-leaks test/
test/acceptance/",
      "test-cov": "istanbul cover node_modules/mocha/bin/_mocha
         -- --require test/support/env --reporter dot --check-
leaks test/ test/acceptance/",
      "test-tap": "mocha --require test/support/env
         --reporter tap --check-leaks test/ test/acceptance/",
      "test-travis": "istanbul cover
node_modules/mocha/bin/_mocha
         --report lcovonly -- --require test/support/env
         --reporter spec --check-leaks test/ test/acceptance/"
   },

   "gitHead": "63ab25579bda70b4927a179b580a9c580b6c7ada",
   "bugs": {
      "url": "https://github.com/strongloop/express/issues"
```

```
  },

  "_id": "express@4.11.2",
  "_shasum": "8df3d5a9ac848585f00a0777601823faecd3b148",
  "_from": "express@*",
  "_npmVersion": "1.4.28",
  "_npmUser": {
    "name": "dougwilson",
    "email": "doug@somethingdoug.com"
  },

  "maintainers": [{
    "name": "tjholowaychuk",
    "email": "tj@vision-media.ca"
  },

  {
    "name": "jongleberry",
    "email": "jonathanrichardong@gmail.com"
  },

  {
    "name": "shtylman",
    "email": "shtylman@gmail.com"
  },

  {
    "name": "dougwilson",
    "email": "doug@somethingdoug.com"
  },

  {
    "name": "aredridel",
    "email": "aredridel@nbtsc.org"
  },

  {
    "name": "strongloop",
    "email": "callback@strongloop.com"
  },

  {
    "name": "rfeng",
    "email": "enjoyjava@gmail.com"
  }],

  "dist": {
    "shasum": "8df3d5a9ac848585f00a0777601823faecd3b148",
    "tarball": "https://registry.npmjs.org/express/-/express-
4.11.2.tgz"
  },

  "directories": {},
```

```
      "_resolved": "https://registry.npmjs.org/express/-/express-
4.11.2.tgz",
      "readme": "ERROR: No README data found!"
}
```

A Package.json attribútumai

name - a csomag neve

version - a csomag verziója

description - a csomag leírása

homepage - a csomag honlapja

author - a csomag szerzője

contributors - a csomag közreműködői neve

dependencies - a függőségek listája. Az NPM automatikusan telepíti az itt említett összes függőséget a csomag node_module mappájába.

repository - a repository típusa és a csomag URL-je

main- a csomag fő belépési pontja

keywords – kulcsszavak

Modul eltávolítása:

```
$ npm uninstall express
```

Modul frissítése:

```
$ npm update express
```

Modul keresése:

```
$ npm search express
```

Modul létrehozása:

A modul létrehozásához a package.json létrehozása szükséges. Generáljuk a package.json-t az NPM használatával, amely előállítja a package.json alapvázát.

```
$ npm init
```

Modul közzététele:

```
$ npm publish
```

Ha minden rendben van a modullal, akkor azt közzétehetjük a repository-ban, és elérhető lesz az NPM használatával, mint bármely más Node.js modul.

Node.js - Callbacks Concept

Mi az a callback?

A callback egy aszinkron függvény. Egy adott feladat befejezésekor egy callback függvényt hívunk meg. A NodeJS nagymértékben használja a callback mechanizmust. A NodeJS összes API-ja úgy van megírva, hogy támogatja a callback-et.

Például elkezdhetjük a fájl olvasását, és azonnal visszatérhetünk a következő utasítás végrehajtásához. Amint a fájl I / O befejeződött, meghívja a callback függvényt, miközben átadja a fájl tartalmát paraméterként neki. Tehát nincs blokkolás. Ez a Node.js-t nagyon skálázhatóvá teszi, mivel nagyszámú kérelmet képes feldolgozni anélkül, hogy bármilyen függvény várná az eredmények visszatérését

Példa (szinkron)

Hozzon létre egy input.txt nevű szöveges fájlt a következő tartalommal –

```
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
```

main.js:

```
var fs = require("fs");
var data = fs.readFileSync('input.txt');

console.log(data.toString());
console.log("Program Ended");
```

Futtatás:

```
$ node main.js
```

Példa (aszinkron):

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

fs.readFile('input.txt', function (err, data) {
```

```
   if (err) return console.error(err);
   console.log(data.toString());
});

console.log("Program Ended");
```

Node.js - File System

A Node.js-ben a szokásos fájlműveleteket is el lehet végezni:

```
var fs = require("fs")
```

Szinkron vs aszinkron

Az fs modulban minden metódus rendelkezik szinkron és aszinkron vezióval. Az aszinkron metódusok az utolsó paramétere olyan callback függvény ami ha sikeres volt az aszinron metódus akkor hajtódik végre, az első paraméter pedig hiba esetén. Sokkal jobb aszinkron metódust használni szinkron metódus helyett, mivel az előbbi soha nem blokkolja a programot annak végrehajtása közben, míg a második igen.

Példa:

Készítsünk el egy input.txt :

```
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
```

main.js

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

// Asynchronous read
fs.readFile('input.txt', function (err, data) {
   if (err) {
      return console.error(err);
   }
   console.log("Asynchronous read: " + data.toString());
});

// Synchronous read
var data = fs.readFileSync('input.txt');
console.log("Synchronous read: " + data.toString());

console.log("Program Ended");
```

Fájl megnyitása:

```
fs.open(path, flags[, mode], callback)
```

Paraméterek:

path - elérési út

flag - A flag-ek jelzik a megnyitandó fájl viselkedését.

mode - Beállítja a fájlmódot de csak akkor, ha a fájlt létrehozta.

callback- Ez a callback függvény két argumentumot kap (err, fd).

Sr.No.	Flag & Description
1	**r**: Olvasás. (Hiba ha a fájl nem létezik)
2	**r+: Olvasás és írás (Hiba ha a fájl nem létezik)**
3	**rs: Fájl megnyitása szinkron módban**
4	**rs+: Fájl megnyitása olvasásra, írásra szinkron módban**
5	**w: Írás. Ha nem létezik a fájl akkor létrehozza, ha létezik, akkor a meglévőhöz hozzáfűzi a tartalmat.**
6	**wx: Mint w, csak hibát dob, ha létezik a fájl**
7	**w+: Írás és olvasás. Ha nem létezik a fájl, akkor létrehozza, ha létezik akkor hozzáfűzi a tartalmat.**
8	**wx+: Olyan mint a w+, csak hiba ha a fájl létezik.**
9	**a: Hozzáfűzés. Ha nem létezik a fájl, akkor létrehozza.**
10	**ax: mint az 'a', csak hiba, ha a fájl létezik.**
11	**a+: olvasás és hozzáfűzés. Ha nem létezik a fájl, akkor létrehozza.**
12	**ax+: mint az 'a+', csak hiba ha létezik a fájl.**

Példa:

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

// Asynchronous - Opening File
console.log("Going to open file!");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File opened successfully!");
});
```

Fájl információk lekérdezése:

```
fs.stat(path, callback)
```

paraméterek:

path-elérési út

callback- callback függvény. Két argumentuma van, (err, stats).

Ezenkívül az fs.Stats számos lehetőséget nyújt:

1	**stats.isFile(): fájl-e**
2	**stats.isDirectory(): directory-e**

Példa:

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

console.log("Going to get file info!");
fs.stat('input.txt', function (err, stats) {
   if (err) {
      return console.error(err);
   }
   console.log(stats);
   console.log("Got file info successfully!");

   // Check file type
   console.log("isFile ? " + stats.isFile());
   console.log("isDirectory ? " + stats.isDirectory());
```

```
});
```

Fájl írása:

```
fs.writeFile(filename, data[, options], callback)
```

Ez a metódus felülírja a fájlt, ha a fájl már létezik.

paraméterek

path-elérési út

data-adatok .

option-opciók - A harmadik paraméter egy objektum, amely {encoding, mode, flag}. Alapértelmezés szerint. a kódolás utf8, a mód értéke 0666. és a flag „w”

callback-

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

console.log("Going to write into existing file");
fs.writeFile('input.txt', 'Simply Easy Learning!', function(err)
{
   if (err) {
      return console.error(err);
   }

   console.log("Data written successfully!");
   console.log("Let's read newly written data");

   fs.readFile('input.txt', function (err, data) {
      if (err) {
         return console.error(err);
      }
      console.log("Asynchronous read: " + data.toString());
   });
});
```

Fájl olvasása

```
fs.read(fd, buffer, offset, length, position, callback)
```

fd - Ez az fájl leíró, amelyet az fs.open () ad vissza.

buffer - ez a buffer, amelyben az adatokat vannak.

offset - Ez az eltolás a bufferben, amikor az írás megkezdésekor.

length - Ez egy egész szám, amely meghatározza az olvasandó bájtok számát.

position - Ez egy egész szám, amely meghatározza, hogy honnan kezdje el az olvasást a fájlban. Ha a pozíció nulla, akkor az adatokat az aktuális fájlhelyről fogja kiolvasni.

callback - Ez a függvény három argumentumot kap (err, bytesRead, puffer).

```
var fs = require("fs");
var buf = new Buffer(1024);

console.log("Going to open an existing file");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File opened successfully!");
   console.log("Going to read the file");

   fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){
      if (err){
         console.log(err);
      }
      console.log(bytes + " bytes read");

      // Print only read bytes to avoid junk.
      if(bytes > 0){
         console.log(buf.slice(0, bytes).toString());
      }
   });
});
```

Fájl lezárása:

```
fs.close(fd, callback)
```

fd - Ez az fájl leíró, amelyet az fs.open () metódus ad vissza.

Callback függvény

```
var fs = require("fs");
var buf = new Buffer(1024);

console.log("Going to open an existing file");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File opened successfully!");
   console.log("Going to read the file");
```

```
   fs.read(fd, buf, 0, buf.length, 0, function(err, bytes) {
      if (err) {
         console.log(err);
      }

      // Print only read bytes to avoid junk.
      if(bytes > 0) {
         console.log(buf.slice(0, bytes).toString());
      }

      // Close the opened file.
      fs.close(fd, function(err) {
         if (err) {
            console.log(err);
         }
         console.log("File closed successfully.");
      });
   });
});
```

Fájl törlése:

```
fs.unlink(path, callback)
```

path-elérési út

callback

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

console.log("Going to delete an existing file");
fs.unlink('input.txt', function(err) {
   if (err) {
      return console.error(err);
   }
   console.log("File deleted successfully!");
});
```

Directory készítése:

```
fs.mkdir(path[, mode], callback)
```

path- elérési út

mód - engedély. Alapértelmezés: 0777.

callback

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

console.log("Going to create directory /tmp/test");
fs.mkdir('/tmp/test',function(err) {
   if (err) {
      return console.error(err);
   }
   console.log("Directory created successfully!");
});
```

Directory olvasása:

```
fs.readdir(path, callback)
```

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

console.log("Going to read directory /tmp");
fs.readdir("/tmp/",function(err, files) {
   if (err) {
      return console.error(err);
   }
   files.forEach( function (file) {
      console.log( file );
   });
});
```

Directory törlése:

```
fs.rmdir(path, callback)
```

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

console.log("Going to delete directory /test");
fs.rmdir("/test",function(err) {
   if (err) {
      return console.error(err);
   }
   console.log("Going to read directory /tmp");

   fs.readdir("/tmp/",function(err, files) {
      if (err) {
         return console.error(err);
      }
      files.forEach( function (file) {
         console.log( file );
      });
```

```
    });
});
```

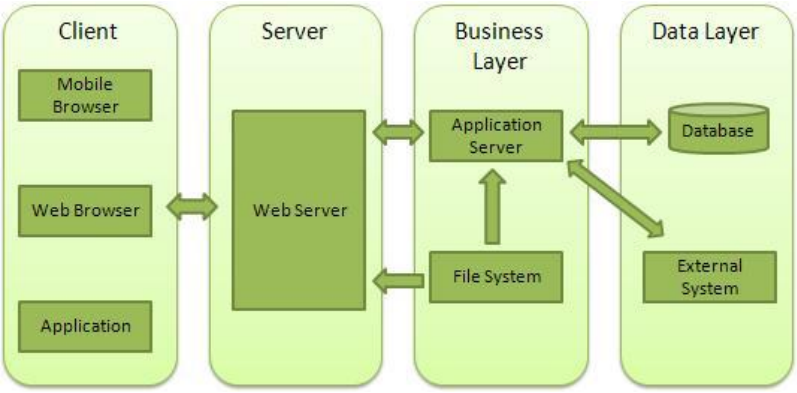
Node.js - Web Module

Mi az a webszerver?

A webszerver olyan szoftver alkalmazás, amely kezeli a HTTP kliens által küldött HTTP kéréseket, például a webböngészők, és a kérésre válaszol.

A legtöbb webszerver támogatja a szerveroldali szkripteket, szkriptnyelvek használatával vagy a feladatot egy alkalmazás szerverhez irányít át, amely adatokat kér az adatbázisból, komplex logikát hajt végre, majd az eredményt a webszerveren keresztül elküldi a HTTP kliensnek.

Web Alkalmazás architektúra



Kliens - Ez a réteg webböngészőből, mobil böngészőből vagy alkalmazásból áll, amelyek HTTP kéréseket tehetnek a webszerverhez.

Szerver – Ez a régteg a webszerver, amely képes lekezelni a kliens kéréseit, és továbbítani a válaszokat.

Üzleti réteg- Ez a réteg az alkalmazásszervert tartalmazza, amelyet a webszerver használ a szükséges feldolgozáshoz. Ez a réteg az adatbázissal vagy néhány külső programmal lép kölcsönhatásba.

Adat réteg - Ez a réteg az adatbázisokat vagy bármilyen más adatforrást tartalmaz.

Webszerver létrehozása a Node segítségével

A Node.js egy http modult biztosít, amely felhasználható egy szerver HTTP kliensének létrehozására. Az alábbiakban bemutatjuk a HTTP szerver minimális felépítését, amely 8081 porton figyel.

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

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

// Create a server
http.createServer( function (request, response) {
   // Parse the request containing file name
   var pathname = url.parse(request.url).pathname;

   // Print the name of the file for which request is made.
   console.log("Request for " + pathname + " received.");

   // Read the requested file content from file system
   fs.readFile(pathname.substr(1), function (err, data) {
      if (err) {
         console.log(err);

         // HTTP Status: 404 : NOT FOUND
         // Content Type: text/plain
         response.writeHead(404, {'Content-Type': 'text/html'});
      } else {
         //Page found
         // HTTP Status: 200 : OK
         // Content Type: text/plain
         response.writeHead(200, {'Content-Type': 'text/html'});

         // Write the content of the file to response body
         response.write(data.toString());
      }

      // Send the response body
      response.end();
   });
}).listen(8081);

// Console will print the message
console.log('Server running at http://127.0.0.1:8081/');
```

Web kliens létrehozása a Node segítségével

A web kliens létrehozható a http modul használatával. Nézzük meg a következő példát:

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

// Options to be used by request
var options = {
   host: 'localhost',
   port: '8081',
   path: '/index.htm'
};

// Callback function is used to deal with response
var callback = function(response) {
   // Continuously update stream with data
```

```
   var body = '';
   response.on('data', function(data) {
      body += data;
   });

   response.on('end', function() {
      // Data received completely.
      console.log(body);
   });
}
// Make a request to the server
var req = http.request(options, callback);
req.end();
```

Node.js - Express Framework

Az Express egy minimális és rugalmas Node.js webalkalmazási keretrendszer, amely robusztus funkciókkal rendelkezik a webes és mobil alkalmazások fejlesztéséhez. Megkönnyíti a Node alapú webes alkalmazások gyors fejlesztését.

Útvonal választást tesz lehetővé a http metódus és URL alapján.

Lehetővé teszi a HTML oldalak dinamikus megjelenítését az argumentumok átadása alapján.

Telepítések:

```
$ npm install express --save


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

body-parser – This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.

cookie-parser – Parse Cookie header and populate req.cookies with an object keyed by the cookie names.

multer – This is a node.js middleware for handling multipart/form-data

```
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)
})
```

Kérés és válasz

Az Express callback függvényt használ, amelynek paraméterei kérés és válaszobjektumok.

```
app.get('/', function (req, res) {
   // --
})
```

Request object- A kérési objektum képviseli a HTTP kérést, és rendelkezik a kérés sztringgel, paraméterekkel, törzzsel, HTTP fejléccel stb. Tulajdonságaival.

Response object - A válaszobjektum reprezentálja a HTTP-választ, amelyet egy Express alkalmazás küld, amikor megkapja a HTTP kérést.

Alap routing:

```
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)
})
```

Statikus fájlok:

Az Express az express.static-al statikus fájlok, például képek, CSS, JavaScript stb kiszolgálására is alkalmas.

A fájlok közvetlen kiszolgálásának elindításához egyszerűen át kell adnia annak a könyvtárnak a nevét, ahol a statikus eszközöket tárolja. Például a képeket, CSS- és JavaScript- fájlokat.

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

A mappa tartalma:

```
node_modules
server.js
public/
public/images
public/images/logo.png
```

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

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

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)
})
```

GET metódus:

Form:

```
<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>
```

server.js:

```
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 metódus:

```
<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>
```

```
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)
})
```

Cookies:

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

```
var cookieParser = require('cookie-parser')

var app = express()
app.use(cookieParser())

app.get('/', function(req, res) {
   console.log("Cookies: ", req.cookies)
})
app.listen(8081)
```

Node.js - RESTful API

REST (REpresentational State Transfer). A REST webes szabványokon alapuló architektúra és HTTP protokollt használ. Az erőforrás áll a középpontban, ahol minden összetevő erőforrás, és egy erőforráshoz egy közös interfész fér hozzá HTTP szabványos módszerekkel. A REST-t először Roy Fielding vezette be 2000-ben.

A REST szerver egyszerűen hozzáférést biztosít az erőforrásokhoz, és a REST kliens hozzáfér az erőforrásokhoz és módosítja azokat a HTTP protokoll használatával. Itt minden erőforrást URI / globális azonosítóval azonosítanak. A REST különféle reprezentációkat használ egy erőforrás, például szöveg, JSON, XML ábrázolására, de a JSON a legnépszerűbb.

HTTP metódusok:

GET - erre az erőforrásra csak olvasható hozzáférést biztosít.

PUT - Ez egy új erőforrás létrehozására szolgál.

DELETE - erőforrás eltávolítására szolgál.

POST - Ezt egy meglévő erőforrás frissítésére vagy új erőforrás létrehozására használják.

Példa:

user.json

```
{
   "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
   }
}
```

```
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)
})
```

Felhasználó hozzáadása:

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

var user = {
   "user4" : {
      "name" : "mohit",
      "password" : "password4",
      "profession" : "teacher",
      "id": 4
   }
}

app.post('/addUser', function (req, res) {
   // First read existing users.
```

```
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function
(err, data) {
      data = JSON.parse( data );
      data["user4"] = user["user4"];
      console.log( data );
      res.end( JSON.stringify(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)
})
```

Felhasználó listázása id alapján:

```
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) {
      var 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)
})
```

Felhasználó törlése:

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

var id = 2;

app.delete('/deleteUser', function (req, res) {
   // First read existing users.
```

```
   fs.readFile( __dirname + "/" + "users.json", 'utf8', function
(err, data) {
      data = JSON.parse( data );
      delete data["user" + 2];

      console.log( data );
      res.end( JSON.stringify(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)
})
```