

# Tehnici Web

## CURSUL 9

Semestrul II, 2022-2023  
Carmen Chirita

<https://sites.google.com/site/fmitehniciweb/>

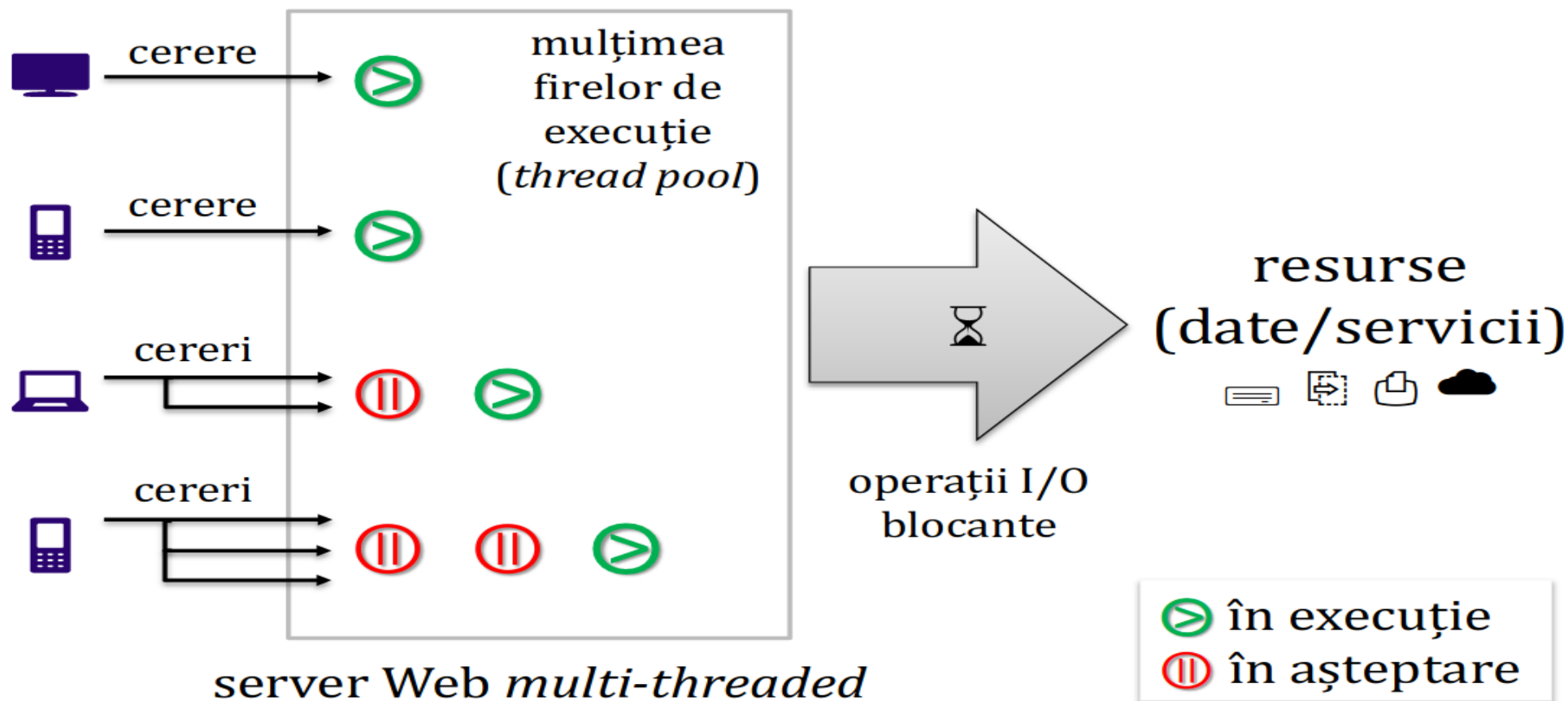
## Server Web

- **Server Web** = program care ruleaza pe un calculator conectat la Internet si care furnizeaza clientilor la cerere diverse resurse Web
- Tim Berners-Lee în 1990 concepe primul **server Web** ruland pe calculatoare NeXT
- *Apache WebServer*—cel mai popular server Web. Prima versiune a fost lansata în 1995
- Alte servere web: Internet Information Services, Lighttpd, NGINX,...

## Serverul Web -caracteristici

- deservește cereri multiple provenite de la clienți pe baza protocolului HTTP
- fiecare cerere e considerată independentă de alta, chiar dacă provine de la același client Web
- se creeaza un număr de fire de execuție (threads) la inițializare, fiecare fir interacționand cu un anumit client

# Serverul Web



cererile multiple de la diverși clienți nu pot fi deservite simultan  
(numărul firelor de execuție asociate unui proces este limitat)

# Browser Web

- **Browser Web** = program software (client) ce permite utilizatorilor să se conecteze la un server Web în vederea explorării resurselor găzduite de acesta (text, grafică, video, etc.)
- Protocolul utilizat: **HTTP**
- Resursele sunt identificate printr-un Uniform Resource Locator (URL)
- Cele mai populare browsere Web: Google Chrome, Safari, Firefox, Internet Explorer, Opera

# Browser Web-caracteristici

- Posibilitatea de a realiza interogări multiple către server
- Asigurarea securității transmițerilor de date
- Stabilirea de liste a site-urilor web favorite
- Memorarea istoricului navigării (history)
- Posibilitatea de a folosi mai multe ferestre de navigare
- Asigurarea suportului pentru diverse limbaje de programare folosite la realizarea paginilor Web dinamice (CSS, JavaScript)

# Protocolul HTTP

- Protocolul HTTP(HyperText Transfer Protocol) = set de reguli de comunicare între un server și browser web.
- Dezvoltat în 1990 de Tim Berners-Lee
- Protocol de tip **cerere/raspuns**
- Portul standard de acces: **80**

# Localizarea resurselor WEB

URI = sir de caractere care identifica o resursa prin nume sau locatie

URN = Uniform Resource Name

(identificare prin nume)

URN: ISBN: 9780062515872

URL = Uniform Resource Locator

(identificare prin locatie)

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html>



# URL

protocol:// host:port /location?query#fragment

HTTP (port 80)

<http://webdesign.about.com/>

<http://search.about.com/?q=URL>

HTTPS = HTTP + securitate (port 443)

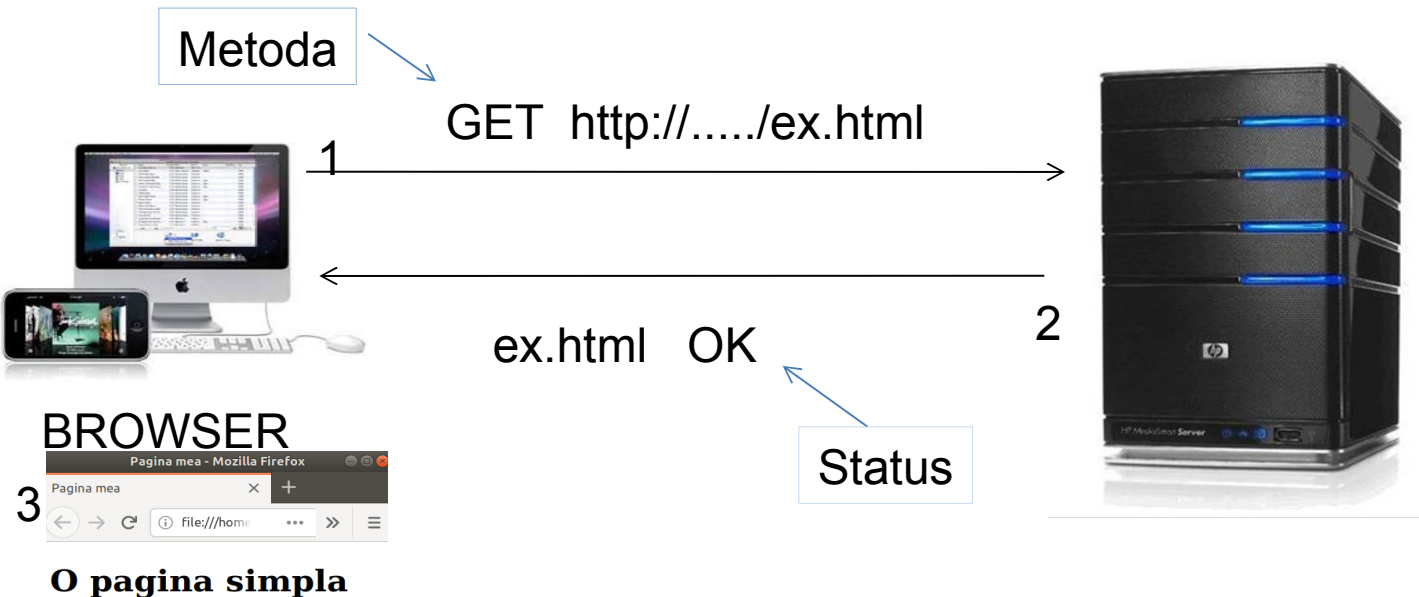
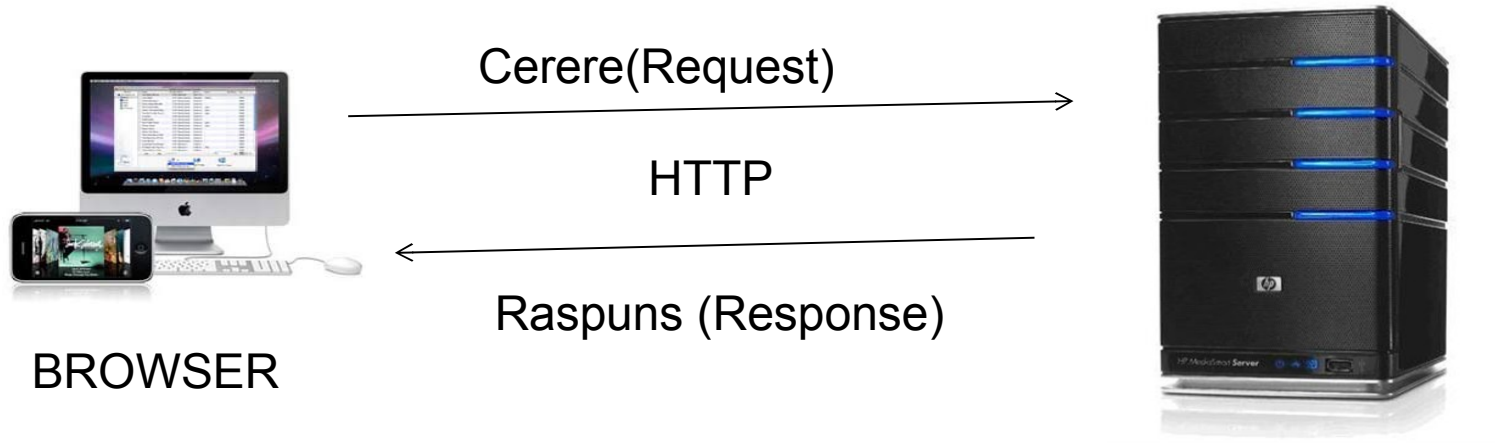
<https://web.stanford.edu/class/cs142/lectures/HTTP.pdf>

File URL = legatura la un fisier local

[file:///home/carmen/TEHNICI\\_WEB\\_CURSURI/date.xml](file:///home/carmen/TEHNICI_WEB_CURSURI/date.xml)

# Client

# Server



# HTTP Request

## Sintaxa unei cereri HTTP

METHOD /path-to-resource HTTP/version-number

Header-Name-1: value

Header-Name-2: value

[ optional request body ]

# HTTP Request-Exemplu

[http://fmi.unibuc.ro/ro/pdf/2019/orar/orar\\_profesori\\_2019-2020\\_s1.pdf](http://fmi.unibuc.ro/ro/pdf/2019/orar/orar_profesori_2019-2020_s1.pdf)

**GET** /...orar\_profesori\_2019-2020\_s1.pdf **HTTP/1.1**

**Host:** fmi.unibuc.ro

**User-Agent:** Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:70.0) Gecko/20100101 Firefox/70.0

**Accept:** text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

**Accept-Language:** ro-RO,ro;q=0.8,en-US;q=0.6,en-GB;q=0.4,en;q=0.2

**Accept-Encoding:** gzip, deflate

**Connection:** keep-alive

Campuri-antet



# Metode HTTP

**GET** - solicită un document/resursă de pe server

**HEAD** - solicită informații despre un document/resursă

**POST** - metodă utilizată pentru a transmite date către server  
și a primi un raspuns

**PUT** - metodă utilizată pentru a actualiza/depune o resursă  
pe server

**DELETE**- metodă utilizată pentru a șterge un  
document/resursă pe server

# Metode HTTP

- tradițional, browser-ul Web permite doar folosirea metodelor **GET** și **POST**
- o metoda este **sigură (safe)** când nu produce modificari în datele serverului
- GET și HEAD sunt *safe*
- POST, PUT, DELETE nu sunt *safe*

# HTTP Response

## Sintaxa unui raspuns HTTP

HTTP/version-number status-code message

Header-Name-1: value

Header-Name-2: value

[ response body ]

# HTTP Response-Exemplu

[http://fmi.unibuc.ro/ro/pdf/2019/orar/orar\\_profesori\\_2019-2020\\_s1.pdf](http://fmi.unibuc.ro/ro/pdf/2019/orar/orar_profesori_2019-2020_s1.pdf)

HTTP/1.1 200 OK

**Date:** Mon, 18 Nov 2019 18:23:27 GMT

**Server:** lighttpd/1.4.26

**Content-Type:** application/pdf

**Content-Length:** 1406

**Last-Modified:** Mon, 25 May 2015 15:34:17 GMT

**Accept-Ranges:** bytes



# Coduri de stare

- 200 OK
- 301 Moved Permanently
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error
- 503 Service Unavailable

# JSON = JavaScript Object Notation

<http://www.json.org/>

Ofera o modalitate de reprezentare a datelor.

Bazat pe JavaScript, este in prezent un format independent de limbaj.

Multe limbaje pot prelucra date in format JSON.

Este folosit pentru schimbul de informații cu serverul.

Elemente de baza:

*Object:* {"cheie1":val1, "cheie2":val2}

*Array:* [val1, val2, val3]

*Value:* string, number, object, array, true, false, null

date.json

```
[{"pers": {"nume": "Ion", "varsta": 42} },  
{"pers": {"nume": "Maria", "varsta": 30} } ]
```

# Sintaxa JSON

Câmpul **cheie** trebuie să fie scris cu ghilimele

`"nume": "Ana"`

Câmpul **valoare** poate fi:

`string, number, obiect (JSON), array, boolean, null`

Obiectele JSON sunt reprezentate între acolade

`{"nume": "Ana", "varsta": 30, "porecla": null }`

Elementele array sunt reprezentate între paranteze drepte

`[ "Ana", "Mihai", "Maria" ]`

Valoare nu poate fi

`function`

`date`

`undefined`

*Valoare:* string, number, object, array, true, false, null

JSON String:    { "nume": "Andrei" }

JSON Number:    { "varsta": 30 }

JSON Object:    { "pers": { "nume": "Ion", "varsta": 42 } }

JSON Array:    { "studenti": [ "Ionut", "Mihai", "Dana" ] }

JSON Boolean:    { "promovat": true }

JSON null:    { "porecla": null }

## Obiecte JSON

```
myObj = {"cheie1":val1, "cheie2":val2, "cheie3":val3};
```

Accesarea obiectelor: `myObj.cheie1` sau `myObj["cheie1"]`

Iterarea proprietatilor unui obiect

```
<script>
var myObj = { "student":"Popescu", "grupa":231,
"promovat":true };
for (x in myObj) {
    document.getElementById("prop").innerHTML += x + "<br>";
}
</script>
```

```
.....
<p id="prop">
```

Paragraful va contine

student
grupa
promovat

# Obiecte JSON

Iterarea valorilor proprietatilor unui obiect

```
<script>
var myObj = { "student":"Popescu", "grupa":231,
"promovat":true };
for (x in myObj) {
    document.getElementById("val").innerHTML += myObj[x]+ "
";
}
</script>
.....
<p id="val"></p>
```

Paragraful va contine

Popescu 231 true

# Obiecte JSON încorporate

```
var myObj = { "student":"Ionescu",  
              "grupa":30,  
              "note": {"nota1":8,"nota2":9,  
"nota3":10}  
              }
```

## Accesarea obiectelor încorporate:

```
myObj.note.nota2 // 9
```

```
myObj.note["nota2"] // 9
```

Modificarea valorilor: `myObj.note.nota1="10";`

Stergerea proprietatilor: `delete myObj.note.nota1;`

# JSON Arrays

[val1, val2, ..., valn]

val1,...,valn pot fi string, number, object, array, boolean or null.

Array în interiorul obiectelor JSON

```
var ob = { "student":"Ionescu",  
           "grupa":30,  
           "note": [7, 8 , 9]  
         }
```

**Accesarea valorilor:** ob.note[0] // 7

**Iterarea valorilor în Array:**

```
for (i în ob.note)  
{  
    x += ob.note[i];  
}
```

sau

```
for (i=0; i< ob.note.length; i++)  
{  
    x += ob.note[i];  
}
```



## Exemplu (w3schools) (array incorporat în array)

```
<p id="demo"></p>
```

```
<script>
```

```
var myObj, i, j, x = "";
```

```
myObj = {
```

```
  "name": "John",
```

```
  "age": 30,
```

```
  "cars": [
```

```
    { "name": "Ford", "models": [ "Fiesta", "Focus", "Mustang" ] },
```

```
    { "name": "BMW", "models": [ "320", "X3", "X5" ] },
```

```
    { "name": "Fiat", "models": [ "500", "Panda" ] }
```

```
  ]
```

```
}
```

```
for (i in myObj.cars) {
```

```
  x += "<h2>" + myObj.cars[i].name + "</h2>";
```

```
  for (j in myObj.cars[i].models) {
```

```
    x += myObj.cars[i].models[j] + "<br>";
```

```
  }
```

```
}
```

```
document.getElementById("demo").innerHTML = x;
```

```
</script>
```

### **Ford**

Fiesta

Focus

Mustang

### **BMW**

320

X3

X5

### **Fiat**

500

Panda

# Obiectul JSON in JavaScript

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/JSON](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON)

```
JSON.stringify(valoare) // transforma un obiect JavaScript intr-un string JSON  
JSON.parse(text)       //transforma un string JSON într-un obiect JavaScript
```

Exemplu:

```
var o1 = {pers: {nume:"Ion", varsta:42}},  
    o2 = {pers: {nume:"Maria", varsta:30}},  
    o = [o1,o2];  
  
var s = JSON.stringify(o);  
// "[{"pers":{"nume":"Ion","varsta":42}},{"pers":{"nume":"Maria","varsta":30}}]"  
  
localStorage.setItem("myarray", s);  
var st = localStorage.getItem("myarray");  
  
var jo = JSON.parse(st);
```

Poate fi folosit pentru memorare  
in localStorage si sessionStorage

# Node.js

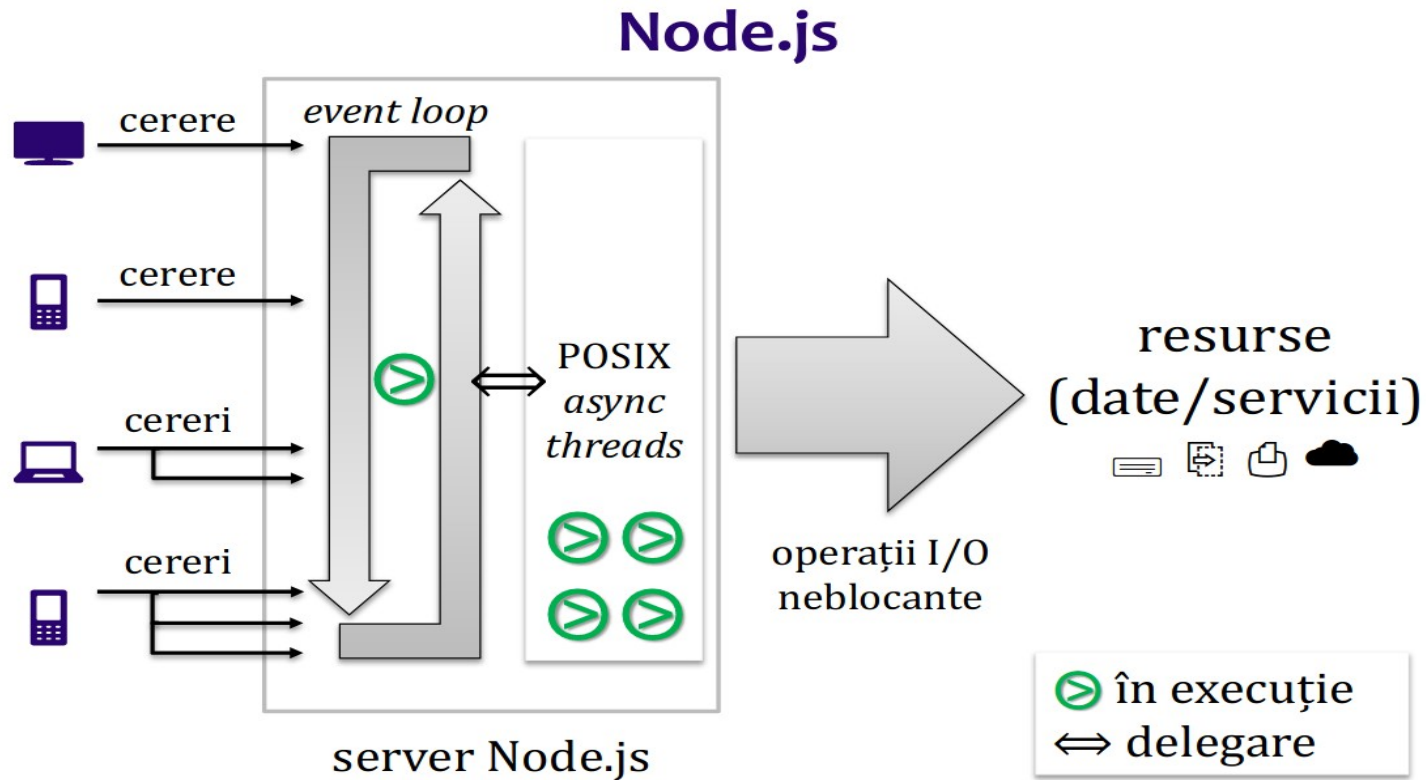
- permite dezvoltarea de aplicații Web la nivel de server în limbajul JavaScript
- creat de Ryan Dahl în 2009 și disponibil gratuit -open source- pentru platformele UNIX/Linux, Windows, MacOS
- trateaza în mod asincron diverse evenimente de intrare/iesire

[nodejs.org/en/download/](https://nodejs.org/en/download/)

# Node.js

o aplicație node.js rulează într-un singur proces

deosebire esențială față de serverele de aplicații Web tradiționale ce recurg la servere multi-process/threaded



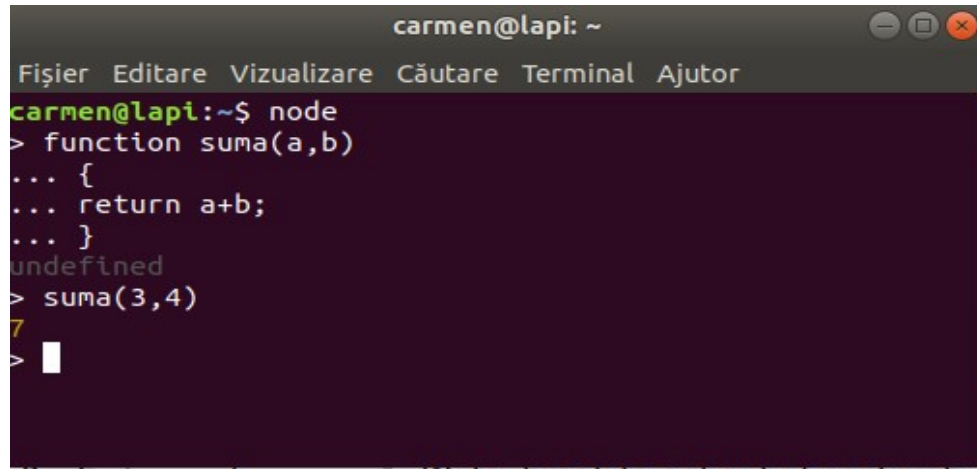
cererile multiple de la diverși clienți pot fi deservite simultan

## Node.js și modulele asociate

- Node.js poate genera continut dinamic pe pagina
- Poate crea, deschide, citi, scrie, sterge si inchide fisiere pe server
- Node.js poate colecta date din formular
- Poate adauga, sterge, modifica date intr-o baza de date
- Poate crea sesiuni
- Poate face criptare/decriptare

# Node.js

- folosind consola **REPL** (Read-Eval-Print-Loop) a node-ului se poate testa cod JavaScript/Node
- lansare REPL: **node** (în linia de comanda)

A screenshot of a terminal window titled 'carmen@lapi: ~'. The window has a menu bar with 'Fișier', 'Editare', 'Vizualizare', 'Căutare', 'Terminal', and 'Ajutor'. The terminal shows the following commands and output:

```
carmen@lapi:~$ node
> function suma(a,b)
... {
...   return a+b;
... }
undefined
> suma(3,4)
7
> 
```

# Node.js

- codul JavaScript rulat pe partea de server  
așteaptă și trateaza cereri provenite de la client/clienti
- pentru a rula aplicația in node: **node aplicatie.js**

# Module și npm (Node Package Manager)

- aplicatiile in node folosesc module (**modul**: librărie JavaScript care are asociat un obiect având proprietăți și metode care pot fi invocate)
- exista module predefinite care se instaleaza odată cu Node ([http](#), [url](#), [fs](#), [querystring](#), [crypto](#), etc.)
- functionalitati suplimentare sunt oferite de module administrate cu **npm** ([nodemailer](#), [express](#), [formidable](#), [cookie-parser](#), [ejs](#), [express-session](#), [socket.io](#), etc.)



# Module și npm

- pentru a crea o aplicație în node este recomandat să se creeze un folder al aplicației (numit folder rădăcina) în care se vor găsi toate fișierele aplicației

- pentru a instala un modul care nu este predefinit:

```
npm install nume_modul --save
```

- pentru a folosi un modul trebuie să-l includem în aplicația node cu funcția `require()`

```
var module = require('module_name'); \ \ întoarce un obiect asociat  
modulului respectiv
```

# NPM (Node Package Manager)

- utilitar pentru administrarea pachetelor (instalare, update, dezinstalare, publicarea modulelor, etc.)

- se instaleaza odată cu Node.js

- comenzi specifice pentru operatii asupra modulelor

npm(site oficial) <https://www.npmjs.com/>

## CLI documentation > CLI

- > **access** Set access level on published packages
- > **adduser** Add a registry user account
- > **audit** Run a security audit
- > **bin** Display npm bin folder
- > **bugs** Bugs for a package in a web browser maybe
- > **build** Build a package
- > **bundle** REMOVED
- > **cache** Manipulates packages cache
- > **ci** Install a project with a clean slate
- > **completion** Tab Completion for npm
- > **config** Manage the npm configuration files
- > **dedupe** Reduce duplication
- > **deprecate** Deprecate a version of a package
- > **dist-tag** Modify package distribution tags
- > **docs** Docs for a package in a web browser maybe
- > **doctor** Check your environments
- > **edit** Edit an installed package
- > **explore** Browse an installed package
- > **help-search** Search npm help documentation
- > **help** Get help on npm
- > **hook** Manage registry hooks
- > **init** create a package.json file
- > **install** Install a package
- > **link** Symlink a package folder
- > **logout** Log out of the registry
- > **ls** List installed packages
- > **npm** javascript package manager
- > **org** Manage orgs
- > **outdated** Check for outdated packages
- > **owner** Manage package owners
- > **pack** Create a tarball from a package
- > **ping** Ping npm registry
- > **prefix** Display prefix
- > **profile** Change settings on your registry profile
- > **prune** Remove extraneous packages
- > **publish** Publish a package
- > **rebuild** Rebuild a package
- > **repo** Open package repository page in the browser
- > **restart** Restart a package
- > **root** Display npm root
- > **run-script** Run arbitrary package scripts
- > **search** Search for packages
- > **shrinkwrap** Lock down dependency versions for publication
- > **star** Mark your favorite packages
- > **stars** View packages marked as favorites
- > **start** Start a package
- > **stop** Stop a package
- > **team** Manage organization teams and team memberships
- > **test** Test a package
- > **token** Manage your authentication tokens
- > **uninstall** Remove a package
- > **unpublish** Remove a package from the registry
- > **update** Update a package
- > **version** Bump a package version
- > **view** View registry info
- > **whoami** Display npm username

# package.json

conține meta-date (numele modulului, versiune, autor,...) + informații privind dependențele de alte module

<https://docs.npmjs.com/files/package.json>

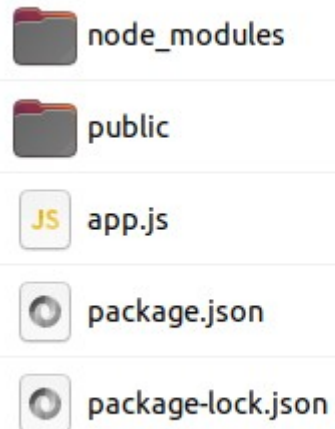
După crearea unui folder pentru proiect (aplicatie)

**npm init** //creaza fișierul package.json în folderul rădăcina al proiectului

**npm install nume-modul --save** //instaleaza un modul ca dependentă a proiectului (va apărea în package.json)

# Exemplu

folderul **aplicatie\_express**



```
Deschide package.json ~/aplicatie_server Salvează
1 |
2   "name": "aplicatie_express",
3   "version": "1.0.0",
4   "description": "web",
5   "main": "app.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8 },
9   "author": "",
0   "license": "ISC",
1   "dependencies": {
2     "cookie-parser": "^1.4.5",
3     "express": "^4.17.1",
4     "formidable": "^1.2.2",
5     "nodemailer": "^6.4.16"
6   }
7 |
```

# Modulul http

<https://nodejs.org/api/http.html>

- include functionalitati HTTP de baza
- permite receptionarea si transferarea datelor prin HTTP
- conține 5 clase:
  - http.Agent
  - http.ClientRequest
  - http.Server
  - http.ServerResponse
  - http.IncomingMessage

# Modulul http

<https://nodejs.org/api/http.html>

metode specifice:

```
var http=require("http");  
var server = http.createServer(handler); //crearea unui server  
Web; întoarce un obiect din clasa http.Server
```

```
server.listen(port, host, backlog, callback) //pornirea serverului  
pe portul specificat
```

portul: numeric  
hostul: string

nr. de cereri acceptate  
in paralel (implicit 511)

funcție care se executa când  
pornește serverul

# Obiectele request si response

funcție care se executa atunci când clientul face o cerere

```
http.createServer(function(request,response){...});
```



- obiectul **request** conține datele cererii primite de la client (ex: putem prelua prin metode specifice datele trimise de client la submiterea unui formular; obiect din clasa **IncomingMessage**)
- obiectul **response** reprezinta răspunsul HTTP emis de server (conține metode pentru setarea campurilor antet, pentru setarea status codului întors de server, pentru scrierea datelor în răspuns, finalizarea raspunsului; obiect din clasa **ServerResponse**)



# Modulul http

cerere emisă de client – clasa `http.ClientRequest`

metode uzuale:

`write()` `end()` `setTimeout()`

evenimente ce pot fi tratate:

`response` `connect` `continue` etc.

# Modulul http

clasa `http.ClientRequest`

**`request.write(chunk[, encoding][, callback])`** //trimite o parte din date  
din corpul cererii

**`chunk`**: string; **`encoding`**: string (implicit „utf8”); **`callback`**: funcție care se va  
executa după ce datele au fost trimise

**`request.end([data[, encoding]][, callback])`** //incheie trimiterea cererii

**`data`**: string; **`encoding`**: string (implicit „utf8”); **`callback`**: funcție  
care se va executa după trimiterea cererii

# Modulul http

răspuns emis de server – clasa `http.ServerResponse`

metode uzuale:

`writeHead()` `getHeader()` `removeHeader()` `write()` `end()`

evenimente ce pot fi  
tratate:

`close` `finish`

proprietăți folosite:

`statusCode` `headersSent`

# Modulul http

clasa `http.ServerResponse`

**response.write(chunk[, encoding][, callback])**

\\trimite o parte din date către client

**chunk:** string; **encoding:** string (implicit „utf8”); **callback:** funcție care se va executa după ce datele au fost trimise

**response.writeHead(statusCode[, statusMessage][, headers])**

\\trimite un antet de răspuns

**statusCode:** numeric; **statusMessage:** string; **headers:** obiect

**response.end([data[, encoding]][, callback])** \\răspunsul e complet

**data:** string; **encoding:** string (implicit „utf8”); **callback:** funcție care se va executa după ce răspunsul de la server este finalizat

salut.js

**// Un program JavaScript care răspunde cu un mesaj de salut la toate cererile adresate de clienti Web**

```
var http=require('http'); // folosim modulul 'http' predefinit
```

```
var server=http.createServer( // cream un server Web  
  // functie anonima ce trateaza o cerere si trimite un raspuns  
  function(request,response){  
    console.log("Am primit o cerere..");
```

```
  // stabilim valori pentru diverse campuri-antet HTTP  
    response.writeHead(200, {"Content-Type" : "text/html"});
```

```
  // emitem raspunsul propriu-zis conform tipului MIME (cod HTML)  
    response.end('<html><body><h1>Salutari din Node.js</h1></body></html>');  
  }));
```

```
server.listen(7000, function(){console.log ('Serverul creat asteapta cereri la  
http://localhost:7000/');}); // serverul este pornit si asculta cereri la portul 7000 al masinii locale
```

carmen@lapi: ~/node-exemple

Fișier Editare Vizualizare Căutare Terminal Ajutor

```
carmen@lapi:~$ cd node-exemple  
carmen@lapi:~/node-exemple$ node salut.js  
Serverul creat asteapta cereri la http://localhost:7000/  
Am primit o cerere..
```

Mozilla Firefox

localhost:7000/

localhost:7000

# Salutari din Node.js

# Clasa URL

<https://nodejs.org/api/url.html>

- procesarea adreselor Web  
(împarte o adresă web în părți care pot fi citite)

```
const myURL= new URL(input[, base]) //creaza un obiect cu url-ul parsat
```

```
myURL=new URL("http://example.com/foo/?p1=a&p2=b"); SAU
```

```
myURL=new URL("/foo/?p1=a&p2=b","http://example.com/");
```

```
URL {
  href: 'http://example.com/foo/?p1=a&p2=b',
  origin: 'http://example.com',
  protocol: 'http:',
  username: '',
  password: '',
  host: 'example.com',
  hostname: 'example.com',
  port: '',
  pathname: '/foo/',
  search: '?p1=a&p2=b',
  searchParams: URLSearchParams { 'p1' => 'a', 'p2' => 'b' },
  hash: '' }
```

- Class: URL
  - new URL(input[, base])
  - url.hash
  - url.host
  - url.hostname
  - url.href
  - url.origin
  - url.password
  - url.pathname
  - url.port
  - url.protocol
    - Special schemes
  - url.search
  - url.searchParams
  - url.username
  - url.toString()
  - url.toJSON()

# Clasa URLSearchParams

-ofera acces la partea query a unei adrese url

```
const myURL = new URL('https://example.org/?abc=123');
```

myURL: obiect din clasa URL

myURL.**searchParams** : obiect din clasa URLSearchParams

```
console.log(myURL.searchParams.get('abc')); //123
```

- Class: URLSearchParams
  - new URLSearchParams()
  - new URLSearchParams(string)
  - new URLSearchParams(obj)
  - new URLSearchParams(iterable)
  - urlSearchParams.append(name, value)
  - urlSearchParams.delete(name)
  - urlSearchParams.entries()
  - urlSearchParams.forEach(fn[, thisArg])
  - urlSearchParams.get(name)
  - urlSearchParams.getAll(name)
  - urlSearchParams.has(name)
  - urlSearchParams.keys()
  - urlSearchParams.set(name, value)
  - urlSearchParams.sort()
  - urlSearchParams.toString()
  - urlSearchParams.values()
  - urlSearchParams[Symbol.iterator]()

// Program ce ilustreaza procesarea URL-urilor

```
var adresa = new URL(
'http://TehniciWeb:8080/anulIII/grupa232/nume_student=Ionescu&nota_student=10');
console.log (adresa);
if (adresa.searchParams.get('nota_student') >= 5) {
    console.log ('Ai promovat examenul cu nota ' + adresa.searchParams.get('nota_student'));
} else {
    console.log ('Nu ai promovat examenul');
}
```

```
URL {
  href:
    'http://tehniciweb:8080/anulIII/grupa232/?nume_student=Ionescu&nota_student=10',
  origin: 'http://tehniciweb:8080',
  protocol: 'http:',
  username: '',
  password: '',
  host: 'tehniciweb:8080',
  hostname: 'tehniciweb',
  port: '8080',
  pathname: '/anulIII/grupa232/',
  search: '?nume_student=Ionescu&nota_student=10',
  searchParams:
    URLSearchParams { 'nume_student' => 'Ionescu', 'nota_student' => '10' },
  hash: '' }
Ai promovat examenul cu nota 10
```



## Tratarea erorilor în JavaScript

JavaScript dispune de un mecanism de tratare a erorilor (exista erori implicite și erori definite de utilizator cu ajutorul obiectului *Error*)

- lansarea unei erori

**throw** *expresie* //expresie este un argument de tip Error dar poate fi de alt tip (ex. string)

- prinderea unei erori

**try** { //instrucțiuni care pot genera erori }

**catch**(e) { // instrucțiuni de testare a tipului de eroare generat și de tratare a erorii }

**finally** { //cod care se executa la final }

## Tratarea erorilor în JavaScript

```
<script>

function f(n){
var pn=parseInt(n);
if(Number.isNaN(pn))
    throw "not a number";
var s=pn*(pn+1)/2;
alert(s);
}

window.onload=function(){
var nr=prompt("Introduceti un numar:");
try{
    f(nr);
} catch(err) {
    alert("A aparut o eroare: " + err);
}
}
</script>
```

# Modulul fs

- permite operatii cu fisiere/directoare pe server (citire, creare, adaugare date, stergere, etc. )

Metode:

`readFile()` `writeFile()` `appendFile()` //variantele asincrone  
`readFileSync()` `writeFileSync()` `appendFileSync()` //variantele sincrone

```
fs.readFile(fileName [,options], callback)
```

```
var fs = require('fs');  
fs.readFile('file.txt', 'utf8', function (err, data) {  
    if (err) throw err;  
    console.log(data);  
});  
console.log('citire asincrona');
```

```
fs.readFileSync(fileName [,options])
```

```
var fs = require('fs');  
var data = fs.readFileSync('file.txt', 'utf8');  
console.log(data);  
console.log('citire sincrona');
```

## file.js

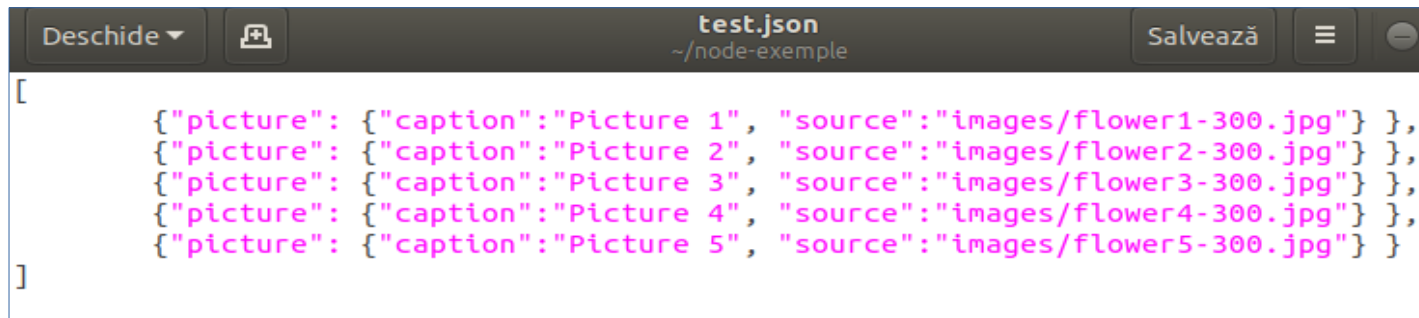
//citire date dintr-un fisier json și adaugarea lor într-un fisier html

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

Date în format JSON

```
fs.readFile('test.json',function (err, data) {  
    if (err) throw err;  
    var json=JSON.parse(data);    //transformare din string JSON într-un array JavaScript  
    fs.writeFileSync('test.html','<html><body>');  
    for(var i=0; i<json.length;i++)  
    fs.appendFileSync('test.html','<img src='+json[i].picture.source +'>');  
    fs.appendFileSync('test.html','</body></html>');  
  
    console.log('Operatie completa.');
```

```
});
```



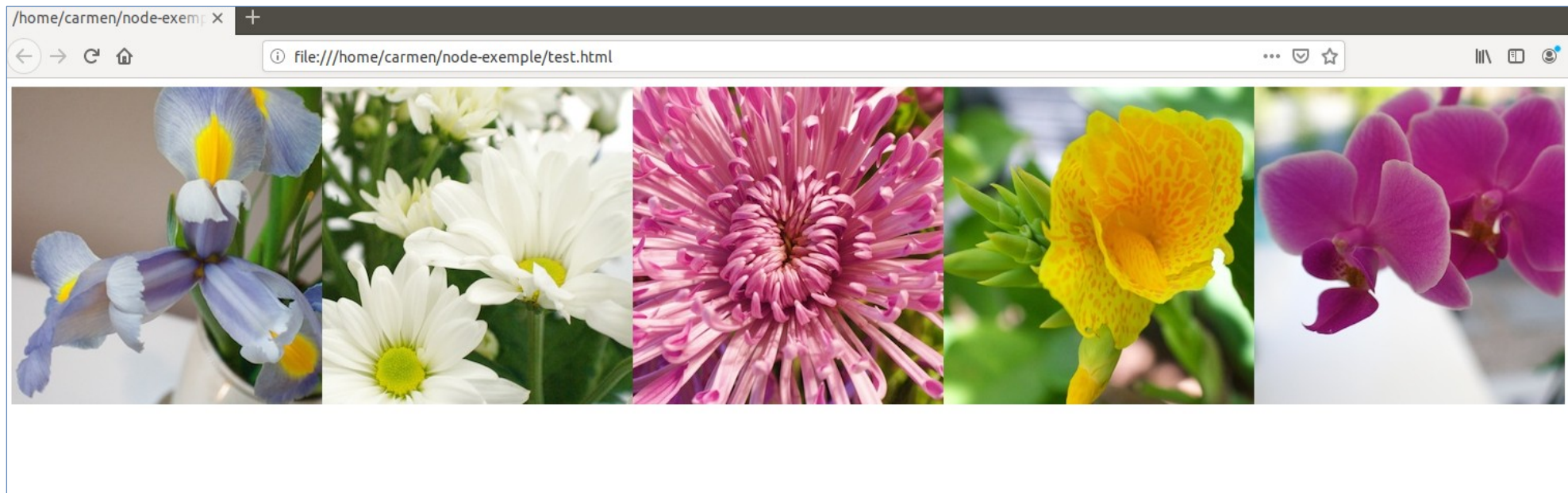
The screenshot shows a code editor window with a dark theme. The title bar indicates the file is 'test.json' located at '~/node-exemple'. The editor contains a JSON array with five objects, each representing a picture. The text is color-coded: strings are in pink, and objects/arrays are in light blue. The JSON structure is as follows:

```
[  
  {  
    "picture": {  
      "caption": "Picture 1",  
      "source": "images/flower1-300.jpg"  
    }  
  },  
  {  
    "picture": {  
      "caption": "Picture 2",  
      "source": "images/flower2-300.jpg"  
    }  
  },  
  {  
    "picture": {  
      "caption": "Picture 3",  
      "source": "images/flower3-300.jpg"  
    }  
  },  
  {  
    "picture": {  
      "caption": "Picture 4",  
      "source": "images/flower4-300.jpg"  
    }  
  },  
  {  
    "picture": {  
      "caption": "Picture 5",  
      "source": "images/flower5-300.jpg"  
    }  
  }  
]
```

```
carmen@lapi: ~/node-exemple
Fișier Editare Vizualizare Căutare Terminal Ajutor
To run a command as administrator (user "root"), use "sudo <command>"
See "man sudo_root" for details.

carmen@lapi:~$ cd node-exemple
carmen@lapi:~/node-exemple$ node file.js
Operatie completa.
carmen@lapi:~/node-exemple$
```

Fișierul creat **test.html**



## Exemplul 1: submiterea datelor dintr-un formular cu metoda GET și salvarea lor într-un fișier text

### form.html

```
<!DOCTYPE html>
<form action="http://localhost:8080/cale" method="GET"
target="_blank">
.....
</form>
```

Nume:

Varsta:

Localitate:

Query string în url

localhost:8080/cale?name=Oana&age=23&city=Cluj

### form.js

```
var http = require('http');
var fs = require('fs');
var server = http.createServer(function (req, res)
{
  console.log("O cerere;");
  var url_parts = new URL(req.url, 'http://localhost:8080/');
  if (url_parts.pathname === '/cale') {
    var query = url_parts.searchParams;
    fs.appendFileSync('date.txt', query.get('name') + ',' + query.get('age') + ',' + query.get('city') + '\n');
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end(query.get('name') + ' din ' + query.get('city') + ' are ' + query.get('age') + ' ani ');
  }
}).listen(8080);
console.log('Serverul creat asteapta cereri la http://localhost:8080/');
```

Deschide ▾ **date.txt**    
 ~/node-exemple/F...   
 Andrei,25,Brasov   
 Maria,10,Bucuresti   
 Bogdan,20,Bucuresti   
 Oana,23,Cluj

## Exemplul 2:

- la requesturi către `/salveaza` (submiterea datelor dintr-un formular cu metoda POST) se va afisa un raspuns în format html cu datele submise și se vor salva datele submise într-un fisier json
- la requesturi către `/afiseaza` se vor citi date dintr-un fișier json și se va afisa răspunsul sub forma unui tabel html cu datele din fisier

```
<form action="http://localhost:8030/salveaza" method="POST">
  <label>Nume student:</label>
  <input type="text" name="nume">
<br><br>
  <label>Grupa:</label>
  <select name="grupa">
    <option value="1" selected>1</option>
    <option value="2">2</option>
    <option value="3">3</option>
  </select>
<br><br>
  <label> Nota:</label>
  <input type="range" name="nota" min=1 max=10>
  <button type="submit" id="buton"> Trimite </button>
</form>
```

form\_node.html

Nume student:

Grupa:

Nota:

```

var querystring = require('querystring');
var server=http.createServer(function(request, response){
  var body="";
  var url_parts= new URL(request.url,'http://localhost:8030/');

  if(url_parts.pathname =='/salveaza'){ //cererea către "localhost:8030/salveaza"

    request.on('data', function(date){
      body+=date;}); //se salvează datele trimise in cerere ca un querystring
                        (ex: body: nume=Popescu+Ana&grupa=1&nota=9)

    request.on('end', function(){
      console.log("Am primit o cerere");
      var ob_body=querystring.parse(body); //se parseaza datele trimise la submiterea formei
                                            (ex. ob_body : { nume: 'Popescu Ana', grupa: '1', nota: '9' })

      console.log(ob_body);
      response.statusCode = 200;
      response.setHeader('Content-Type', 'text/html');
      response.write("<html><body><p> " + ob_body.nume + " din grupa " + ob_body.grupa + " are nota " + ob_body.nota +
        " </p></body></html>");
      response.end();

      if (fs.existsSync("studenti.json"))
        {
          var date= fs.readFileSync("studenti.json");
          ob=JSON.parse(date);
        }
        else
          ob=[];
      ob.push(ob_body);
      fs.writeFileSync("studenti.json",JSON.stringify(ob)); //adaugare in fisier
    });
  }
}); server.listen(8030, function(){console.log("serverul asculta pe portul 8030");});

```

## studenti.json

```

[{"nume": "Petrescu Matei", "grupa": "3", "nota": "8"},
{"nume": "Popescu Ana", "grupa": "3", "nota": "10"},
{"nume": "Georgescu Maria", "grupa": "1", "nota": "8"},
{"nume": "Ionescu Cezar", "grupa": "2", "nota": "8"}]

```



```

if(url_parts.pathname === '/afiseaza'){ //cererea către "localhost:8030/afiseaza"

    fs.readFile("studenti.json",function(err,date){ //citire asincrona din fisier
        if(err) throw err;
        var studenti=JSON.parse(date);
        response.statusCode=200;
        response.write('<html><body><table style="border:1px solid black"><tr><td
style="border:1px solid black">Student</td><td style="border:1px solid black">Grupa</td><td
style="border:1px solid black">Nota</td></tr>');

        for(s of studenti) {
            response.write('<tr><td style="border:1px solid black">');
            response.write(s.num);
            response.write('</td><td style="border:1px solid black">');
            response.write(s.grupa);
            response.write('</td><td style="border:1px solid black">');
            response.write(s.nota);
            response.write('</td></tr>');
        }
        response.write('</table></body></html>');
        response.end();

    });
}

```

Student	Grupa	Nota
Petrescu Matei	3	8
Popescu Ana	3	10
Georgescu Maria	1	8
Ionescu Cezar	2	8

# Module custom în Node.js

module create de utilizator  
și incluse apoi în aplicație

Cuvântul cheie: **exports**

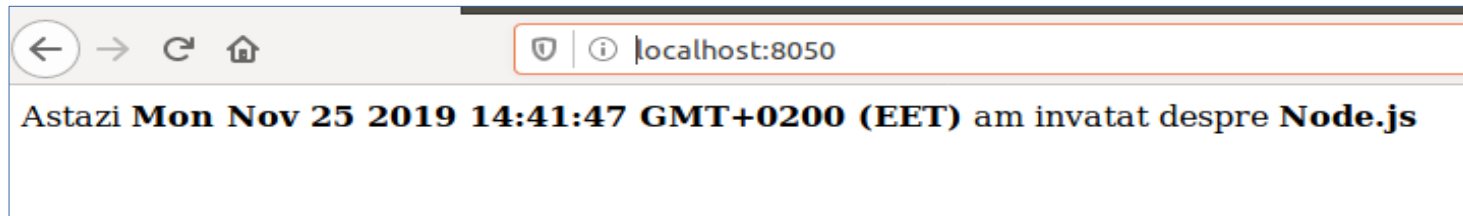
mymodule.js

```
module.exports = {  
  myDate : function () {  
    return Date();  
  },  
  myMessage: function() { return 'Node.js';}  
};
```

```
var http = require('http');  
var date=require('./mymodule');
```

custom.js

```
http.createServer(function(req, res) {  
  console.log('am primit un request');  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.end('<html><body>Astazi <b>' + date.myDate() +  
'</b> am invatat despre <b>' + date.myMessage() + '</b></body></html>');  
}).listen(8050);
```



# Modulul crypto

[https://nodejs.org/api/crypto.html#crypto\\_crypto\\_createcipheriv\\_algorithm\\_key\\_iv\\_options](https://nodejs.org/api/crypto.html#crypto_crypto_createcipheriv_algorithm_key_iv_options)

-ofera metode pentru criptarea și decriptarea datelor  
(ex. pt. securizarea parolelor înainte de a fi stocate în baza de date)

**createCipheriv() createDecipheriv() update() final()**

```
const crypto = require('crypto'); //includem modulul crypto

function encrypt(text){
  var cipher = crypto.createCipheriv('aes128', //creaza un obiect de tip algoritm de cifrare
    'passwordpassword', 'vectorvector1234')
  var crypted = cipher.update(text, 'utf8', 'hex') //criptarea textului
  crypted += cipher.final('hex') //finalizarea criptarii
  return crypted;
}

function decrypt(text){
  var decipher = crypto.createDecipheriv('aes128',
    'passwordpassword', 'vectorvector1234')
  var dec = decipher.update(text, 'hex', 'utf8')
  dec += decipher.final('utf8')
  return dec;
}

console.log(encrypt("Hello world"));
console.log(decrypt("bcf5fd6d5cba937013bb69bcbf68907c"));
```

# Trimiterea de emailuri folosind modulul **nodemailer**

## **npm install nodemailer --save** //instalarea modulului

```
var nodemailer = require('nodemailer'); //folosirea modulului nodemailer
```

```
var transporter = nodemailer.createTransport({ //face autentificarea
  service: 'gmail',
  auth: {
    user: 'my.mail.node@gmail.com',
    pass: 'nodemailer'
  }
});
```

```
var mailOptions = { //optiunile mesajului
  from: 'my.mail.node@gmail.com',
  to: 'carmen_stama@yahoo.com',
  subject: 'Mesaj din Node.js',
  text: 'Hello!'
};
```

```
transporter.sendMail(mailOptions, function(error, info){ //trimite mail
  if (error) {
    console.log(error);
  } else {
    console.log('Mail trimis: ' + info.response);
  }
});
```

<https://nodemailer.com/usage/>

# Trimiterea de emailuri folosind modulul **nodemailer**

## Setari în Google Account



<https://nodemailer.com/usage/>

# Express.js

**Express** este un framework cu ajutorul caruia se implementeaza mai ușor (cod mai simplu și mai clar) aplicatiile server

- este integrat cu diferite module pentru procesarea de cereri și de răspunsuri HTTP (express-session, cookie-parser, nodemailer, etc.)
- ofera metode pentru crearea de rute prin intermediul cărora se stabileste modul de procesare al cererii în funcție de resursa solicitată precum și de metoda folosită
- permite redarea dinamica a paginilor HTML pe baza unor template-uri (ejs)
- furnizează acces la informațiile stocate în diferite surse de date

Instalare: `npm install express --save`

## Creare server cu **express**

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

//importam modulul express; obținem o funcție pe care o apelam pentru a crea o aplicație express

```
var app = express(); // obiectul app corespunzător aplicației  
                        express are metode pentru:
```

- definirea rutelor corespunzătoare cererilor HTTP
- redarea HTML (template-uri folosind EJS)
- accesul la resurse statice (middleware-ul `express.static` )

```
app.listen(5000); //pornirea serverului la portul specificat
```

<https://expressjs.com/en/5x/api.html>

# Crearea rutelor

- rutele create în express reprezintă modul de procesare al cererii în funcție de tipul ei (GET, POST, PUT, DELETE) și a resursei cerute
- rutele se definesc folosind metode ale obiectului aplicației Express (notat **app** in exemple) care corespund metodelor HTTP

Sintaxa unei rute: **app.metoda(cale\_ruta, callback)**

metoda: **get, post, put, delete**    cale\_ruta: **expresie regulata**

callback: **function(request,response,next){..}**



Obiectul request

Obiectul response

Functia middleware urmatoare

În general, dacă în funcția callback răspunsul emis este complet, se folosesc doar primii doi parametri (obiectele request si response)



## Crearea rutelor - exemple

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

```
//ruta către rădăcina (cereri get) (http://localhost:5000/)  
app.get( "/", function(req,res){ res.send('root'); });
```

```
//ruta către pagina1 (http://localhost:5000/pagina1)  
app.get( "/pagina1", function(req,res){ res.send('cerere către pagina 1'); });
```

```
//ruta către toate paginile care se termina cu .html (http://localhost:5000/index.html)  
app.get("/*.html", function(req,res){ //procesarea cererii });
```

```
app.listen(5000);
```

Obiectul de tip **request** <https://expressjs.com/en/5x/api.html#req>

- conține proprietati pentru procesarea cererii

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

req.**query** - obiect continand parametrii din query

(ex. ?name=Corina&age=10&city=Bucuresti ⇒

{name: 'Corina', age: '10', city: 'Bucuresti'} )

req.**body** - obiect continand body-ul parsat

req.**path** - partea din url numita path

localhost:5000/cale?name=Corina&age=10&city=Bucuresti

url-ul cererii pentru forma  
submisa cu metoda GET

Obiectul de tip **response** <https://expressjs.com/en/5x/api.html#res>

- conține metode pentru setarea răspunsului HTTP

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

res.**write(content)** - scrie în conținutul răspunsului

res.**status(code)** - setează status codul răspunsului

res.**end()** - încheie răspunsul

res.**end(msg)** - încheie răspunsul cu un conținut

res.**send(content)** - write() + end()

res.**redirect(url)** - redirectionare către alt url

# Funcțiile middleware

- funcțiile **middleware** sunt utilizate atunci când sunt necesare mai multe procesari pentru a răspunde la o anumită solicitare (de ex. pentru a parsa body-ul pentru formularele submise cu metoda post, pentru a parsa anumite headere ale cererii, pentru a furniza resurse dintr-un folder, pentru crearea unei sesiuni)
- sunt funcții care primesc ca argumente obiectele **request, response** și următoarea funcție (notată de obicei **next**) din ciclul cerere-raspuns al aplicației
- atunci când este invocată (**next()**) executa funcția middleware succesivă cu middleware-ul curent
- dacă funcția middleware curentă nu încheie ciclul cerere-raspuns trebuie să apeleze **next()** pentru a trece controlul la următoarea funcție middleware; altfel, cererea va rămâne suspendată

## Metoda use()

- este folosită pentru setarea unei funcții middleware
- ordinea de setare a funcțiilor contează, deoarece procesările se fac în ordinea în care au fost definite

```
app.use(function (req, res, next) {...})
```

sau

```
app.use(cale_ruta, function(req,res,next){...});
```

# Metoda use()

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

//functie middleware care se executa la fiecare request catre '/pagina1', înaintea  
functiei handler



```
app.use('/pagina1', function(req, res, next){  
  var data=new Date();  
  console.log("O cerere catre pagina1 a fost primita in " + data);  
  next();  
});
```

```
app.get('/pagina1', function(req, res){ //functie handler care trimite  
                                         raspunsul  
  res.send('Pagina 1');  
});
```

```
app.listen(3000);
```

# Fisiere și directoare statice - middleware-ul `express.static`

- fisierele statice sunt fișiere pe care clientii le descarca așa cum sunt de pe server
- în mod implicit, Express nu poate servi fișiere statice de pe server
- pentru a-l activa sa întoarcă resursele statice dintr-un director se folosește middleware-ul `express.static`

Sintaxa: `app.use(express.static('director'))` sau

`app.use(cale_ruta, express.static('director'));`

- `director` este numele unui director static în folderul rădăcina al aplicației Express
- Express caută fișierele în raport cu directorul static, deci numele directorului static nu face parte din adresa URL

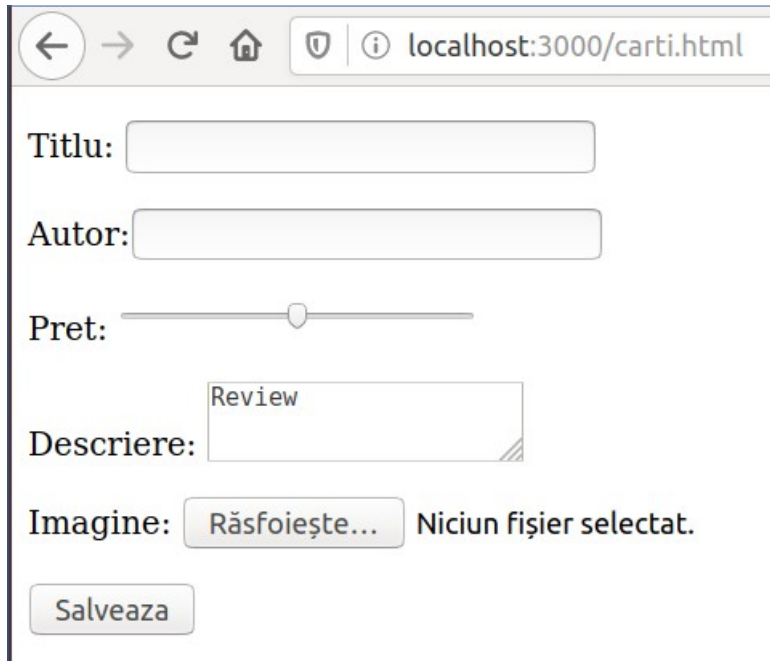
# Fisiere și directoare statice - middleware-ul **express.static**

```
var express = require('express');  
var app = express();  
app.use(express.static('html'));  
app.use(express.static('poze'));  
app.listen(3000);
```

**html** și **poze** sunt directoare în folderul rădăcina al aplicației

**cărți.html** este un fisier din directorul **html**

**vara.jpg** este un fisier din directorul **poze**



← → ↻ 🏠 ⓘ localhost:3000/carti.html

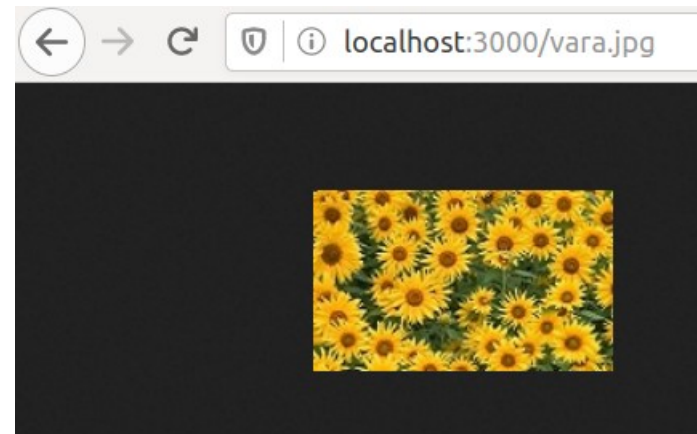
Titlu:

Autor:

Pret:

Descriere:

Imagine:  Niciun fișier selectat.





## Middleware-ul `express.urlencoded`

- parseaza body-ul pentru formulare submise cu metoda post

Sintaxa:

```
app.use(express.urlencoded({extended:true/false})) sau  
app.use(cale_ruta,express.urlencoded({extended:true/false}))
```

`cale_ruta` //calea unde se vor trimite datele submise cu post  
`extended:true` //permite obiecte incapsulate

# Middleware-ul **express.urlencoded**

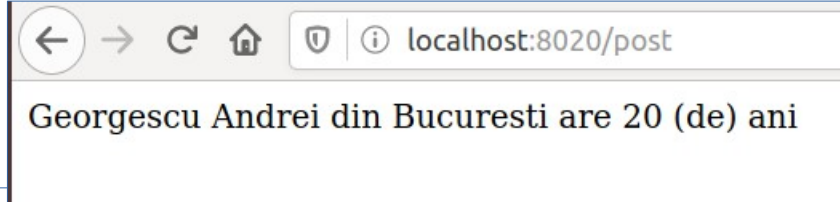
app.js

```
app.use('/post', express.urlencoded({extended:true}));

app.post('/post',function(req, res) {console.log(req.body);
res.send(req.body.persoane.name + ' din '
+ req.body.city + ' are ' + req.body.age + ' (de) ani');})

.....
```

```
carmen@lapi:~/TehniciWeb/node/aplicatie_express$ node app.js
serverul asculta pe portul 8020
{ persoane: { name: 'Georgescu Andrei' },
  age: '20',
  city: 'Bucuresti' }
```



← → ↻ 🏠 🛡 ⓘ localhost:8020/post

Georgescu Andrei din Bucuresti are 20 (de) ani

```
<form id="testform" method="post" action="/post">
<p> <label>Nume:</label> <input type="text" name="persoane[name]"> </p>
<p> <label>Varsta:</label><input type="text" name="age"></p>
<p> <label>Localitate:</label> <select name="city"></p>
  <option value="Bucuresti" >Bucuresti</option>
  <option value="Timisoara" selected>Timisoara</option>
</select>
<p><button type="submit" id="buton"> Trimite </button> </p>
</form>
```

formpost.html

## Exemplul 2 din Node rescris folosind Express

**ex\_express.js**

//la submiterea formei cu post se afiseaza raspunsul în format html  
(continand datele din form) si se salvează datele trimise într-un fisier json

```
var express = require('express');
var app = express();
app.use(express.static('html'));
app.use('/salveaza',express.urlencoded({extended:true}));

app.post('/salveaza',function(request,response){

    response.status = 200;
    response.write("<html><body><p> " + request.body.nume + " din grupa "
    + request.body.grupa + " are nota " + request.body.nota + " </p> </body></html>");
    response.end();

    if (fs.existsSync("studenti.json"))
    {
        var date= fs.readFileSync("studenti.json");
        ob=JSON.parse(date);
    }
    else
    ob=[];
    ob.push(request.body);
    fs.writeFileSync("studenti.json",JSON.stringify(ob));
});
```

## Exemplul 2 din Node rescris folosind Express

**ex\_express.js**

//la cereri get către ruta 'afiseaza' se citesc datele dintr-un fisier json și se trimite răspunsul sub forma unui tabel html continand datele din fisier

```
app.get('/afiseaza', function(request,response){
  fs.readFile("studenti.json",function(err,date){
    if(err) throw err;
    var studenti=JSON.parse(date);
    response.status(200);
    response.write('<html><body><head><link rel="stylesheet" href="stil.css">
    </head><table><tr><td>Student</td><td>Grupa</td><td>Nota</td></tr>');
    for(s of studenti) {
      response.write('<tr><td>');
      response.write(s.nume);
      response.write('</td><td>');
      response.write(s.grupa);
      response.write('</td><td>');
      response.write(s.nota);
      response.write('</td></tr>');
    }
    response.write('</table></body></html>');
    response.end();
  });
});
app.listen(8030, function(){console.log("serverul asculta pe portul 8030");});
```

```
var express = require('express'); //incarc modulul express
```

```
var app = express(); //obiectul aplicației express
```

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

```
app.use('/action', express.urlencoded({extended:true}));
```

app.js

```
app.get('/home',function(req,res){ //inregistrare handler pentru cereri get la calea '/home'
res.send('<html><body>My home page!</body><html>');}) //cu răspuns în format html
```

```
app.get('/get',function(req,res){ //inregistrare handler pentru cereri get la calea '/get' cu
res.redirect("formget.html");}) //redirectionare în raspuns
```

```
app.get('/action', function(req, res) { //inregistrare handler pentru cereri get la calea '/action'
res.send(req.query.name + ' din ' //cu preluare date din form(cu get) in răspuns
+ req.query.city + ' are ' + req.query.age + ' (de) ani');})
```

```
app.get('/post',function(req,res){ //inregistrare handler pentru cereri get la calea '/post'
res.redirect("formpost.html");}) //redirectionare în raspuns
```

```
//inregistrare handler pentru cereri post la calea '/action' cu preluare date din form (cu post) în
răspuns
```

```
app.post('/action',function(req, res) {res.send(req.body.name + ' din '
+ req.body.city + ' are ' + req.body.age + ' (de) ani');})
```

```
app.use(function(req,res){
res.status(404).send("<html><body>Page not found!</body><html>");});
```

```
app.listen(8080, function() {console.log('listening')}); //porneste serverul pe portul 8080
```