

Tehnici Web

CURSUL 13

Semestrul II, 2019-2020
Carmen Chirita

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

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/

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)
- codul JavaScript rulat pe partea de server așteaptă și tratează cereri provenite de la client/clienti
- pentru a rula aplicația în 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 odata 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
```

Modulul http

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

- include functionalitati HTTP de baza
- permite receptionarea si transferarea datelor prin HTTP
- 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
```

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)
- 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)

Modulul url

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

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

url.parse(urlString[, parseQueryString[, slashesDenoteHost]])

\\transforma un URL string într-un URL obiect

href													
protocol	auth		host		path		hash						
			hostname	port	pathname	search							
						query							
"	https:	//	user	:	pass	@ sub.example.com	:	8080	/p/a/t/h	?	query=string	#hash	"
						hostname	port						
protocol		username	password	host									
origin				origin		pathname	search	hash					
href													

- urlObject.auth
- urlObject.hash
- urlObject.host
- urlObject.hostname
- urlObject.href
- urlObject.path
- urlObject.pathname
- urlObject.port
- urlObject.protocol
- urlObject.query
- urlObject.search
- urlObject.slashes

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('fisier.txt','utf8', function (err, data) {
    if (err) throw err;
    console.log(data);
});
console.log('citire asincrona');
```

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

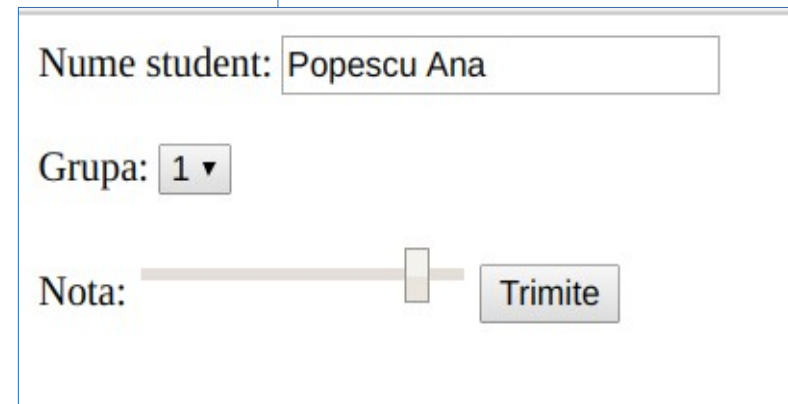
```
var fs = require('fs');
var data =fs.readFileSync('dummyfile.txt', 'utf8');
console.log(data);
console.log('citire sincrona');
```

Exemplu cu node:

- 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: Popescu Ana

Grupa: 1 ▼

Nota: Trimite

```

.....
const server=http.createServer(function(request, response){
  var body="";
  var url_parts=url.parse(request.url,true);

  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>');
```

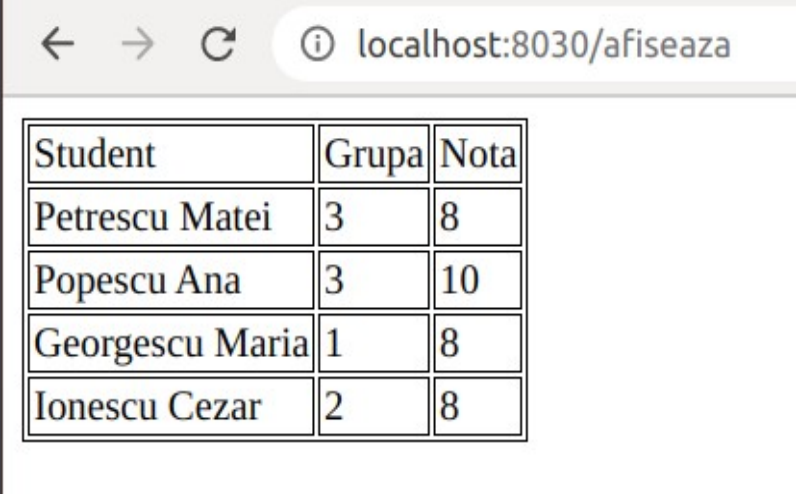
```
        }
```

```
        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

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

Funcția 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`

- fișierele statice sunt fișiere pe care clienții le descarcă așa cum sunt de pe server
- în mod implicit, Express nu poate servi fișiere statice de pe server
- pentru a-l activa să 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

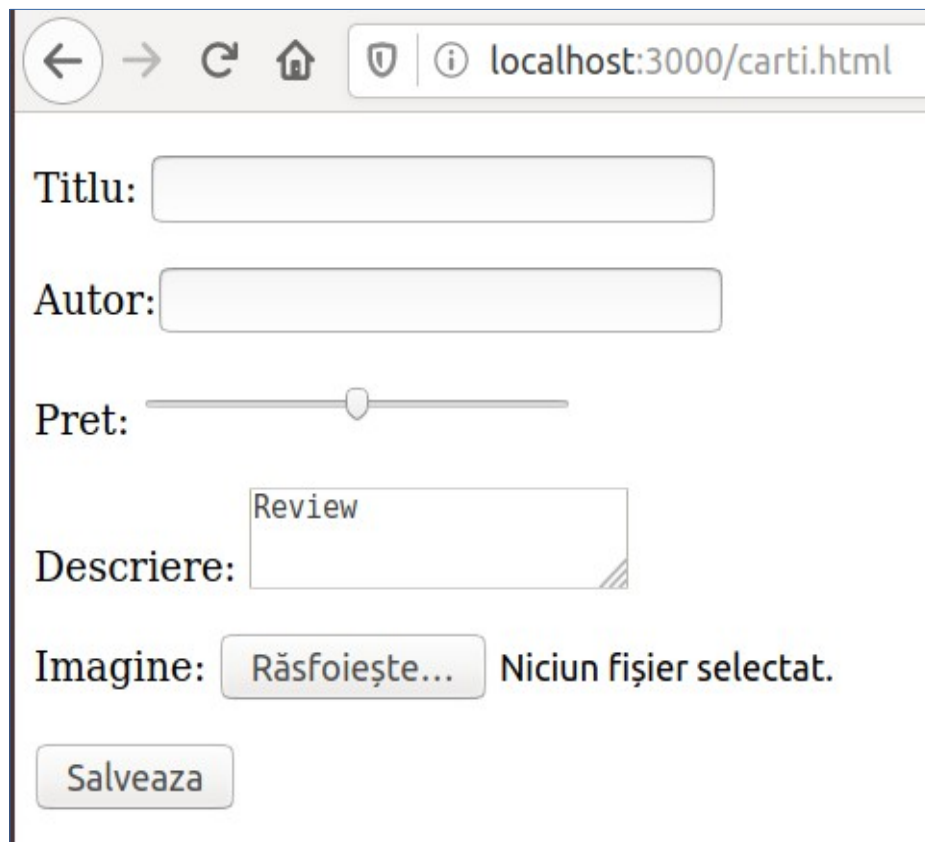
Fisiere și directoare statice - middleware-ul `express.static`

```
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

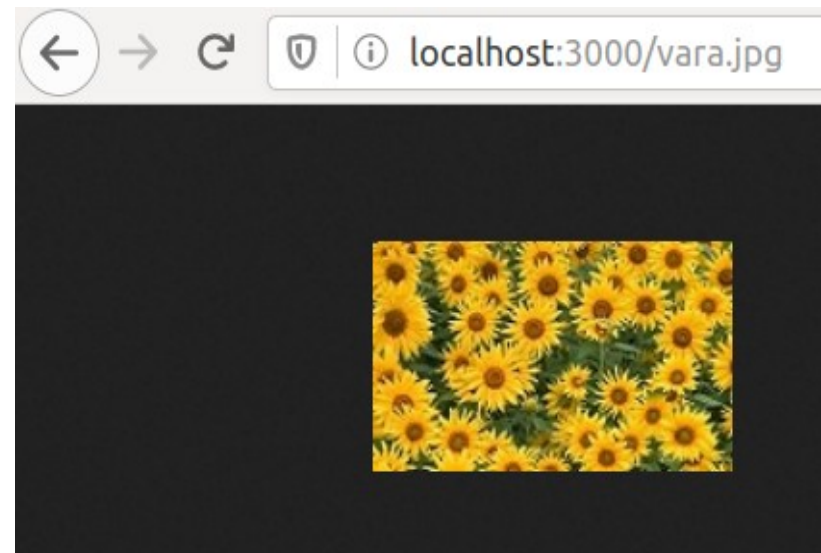
`cărți.html` este un fisier din directorul `html`

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



A screenshot of a web browser window displaying a form for adding a book. The browser's address bar shows `localhost:3000/carti.html`. The form contains the following fields and controls:

- Titlu:** A text input field.
- Autor:** A text input field.
- Pret:** A range slider control.
- Descriere:** A text area with the placeholder text "Review".
- Imagine:** A button labeled "Răsfoiește..." followed by the text "Niciun fișier selectat."
- Salveaza:** A button at the bottom left of the form.



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

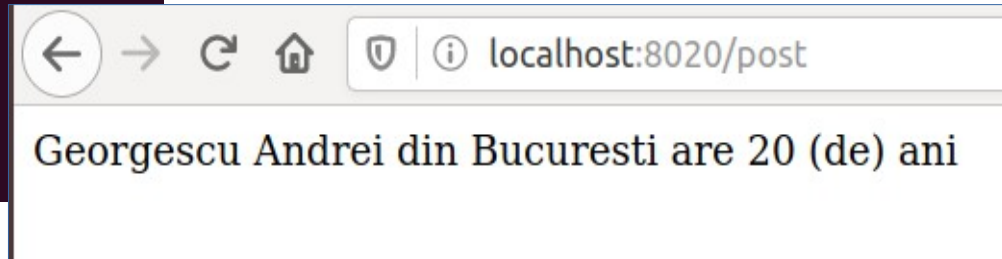
Middleware-ul `express.urlencoded`

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' }
```



```
<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 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 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.num);
      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");});
```

Preluarea datelor dintr-un formular folosind modulul formidable

- avantajul folosirii modulului formidable este ca se pot accesa atât date de tip text, cât și fișiere din inputurile de tip file

Pasii:

- se include modulul în aplicația express folosind funcția require:

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

- în funcția care va procesa cererea se va crea un obiect de tip formular folosind clasa IncomingForm a modulului formidable:

```
var form = new formidable.IncomingForm();
```

- pentru parsarea datelor din cererea post se folosește metoda `parse` a obiectului formular:

```
form.parse(req,function(err,fields,files){...});
```

// funcția parse primește ca parametrii obiectul cerere (req) și o funcție callback care va prelucra datele după parsare (în obiectul fields vom avea campurile formei în afara celor de tip file, în obiectul files vom avea campurile de tip file)

- accesarea proprietatilor campurilor formei: `fields.nume`, `fields.grupa`, `files.cv`, `files.poza`

Upload de fisiere

- în interiorul funcției care va procesa cererea, vom seta calea la care uploadăm fișierele; există două moduri:

\\la crearea obiectului de tip formular

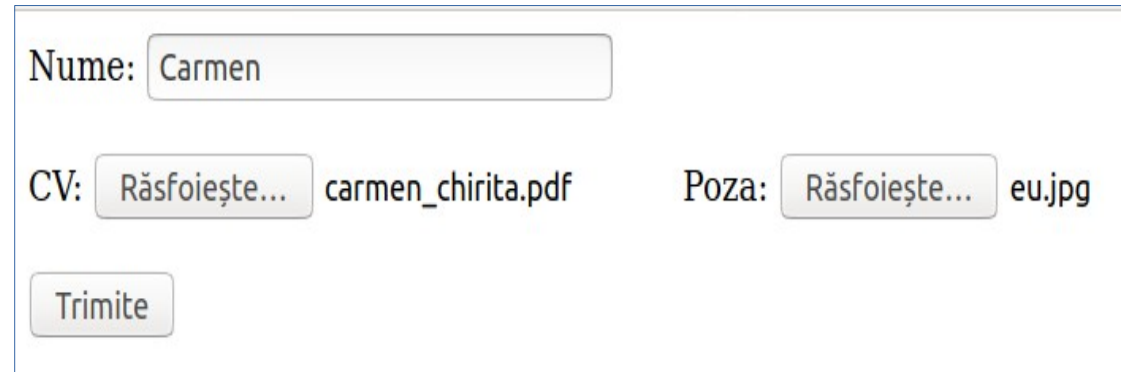
```
var form = new formidable.IncomingForm  
({uploadDir:'cale_director', keepExtensions:true})
```

\\în funcția handler pentru evenimentul `fileBegin` care se declanșează la începutul încărcării fișierului

```
form.on("fileBegin", function(name,file){  
    //se setează calea de upload folosind proprietatea path a ob. fișier  
    file.path='cale_director' + file.name; //ca să păstrăm numele inițial al fișierului din  
file.name  
})
```

Exemplu: la submiterea unui formular cu post se salvează datele într-un fisier json iar fisierele se vor uploada în directorul 'upload'.

```
const express = require('express');
const app = express();
const fs = require('fs');
const formidable=require('formidable');
app.use(express.static('html'));
app.post('/salveaza',function(req,res){
    var ob;
    if (fs.existsSync("persoane.json"))
    {
        var date= fs.readFileSync("persoane.json");
        ob=JSON.parse(date);
    }
    else
    ob=[];
    var form = new formidable.IncomingForm({uploadDir:'upload', keepExtensions:true});
    form.parse(req,function(err,fields,files){
        var ob_form={'nume':fields.nume,'cv':files.cv.path,'poza':files.poza.path};
        ob.push(ob_form);
        fs.writeFileSync("persoane.json",JSON.stringify(ob));
    });
    res.send("Salvat:");
});
```



Nume:

CV: carmen_chirita.pdf Poza: eu.jpg

Sesiuni

- sunt utile atunci când vrem să păstrăm date de la un request la altul.
- în momentul creării unei sesiuni, un client primește un session ID.
- de câte ori facem din nou un request de la același client, vom avea informații despre sesiune folosind acel ID.

Pentru a folosi sesiuni se va instala modulul **express-session**:

```
npm install express-session --save
```

//se creează obiectul corespunzător modulului

```
const session = require('express-session');
```

Sesiuni

Pentru a crea o sesiune se seteaza middleware-ul de sesiune

```
app.use(session({  
  secret: 'abcdefg', // folosit pentru criptarea session ID-ului  
  resave: true, //sa nu stearga sesiunile idle  
  saveUninitialized: false //nu salveaza obiectul sesiune daca nu am setat un  
camp  
}));
```

- după crearea sesiunii, in obiectele de tip request va fi disponibila o proprietate noua, numita chiar **session** (care este de fapt un obiect in care putem seta proprietati cu valorile pe care dorim sa le salvam in sesiunea curenta).
- aceasta proprietate e "globala" pentru toate rutele, in sensul ca daca setam un camp in session, orice request va vedea de la acel moment incolo noul camp din session cu valoarea lui.

Exemplu: contorizarea numarului de vizitari ale unei pagini

```
var express = require('express');
var app = express();
const session = require('express-session')
app.use(session({
  secret: 'abcdefg',
  resave: true,
  saveUninitialized: false,
  //cookie: { maxAge: 60000 }
}));

app.get('/pagina1', function(req, res){
  if(req.session.vizitat){
    req.session.vizitat++;
  } else {
    req.session.vizitat = 1;
  }
  res.send('Esti pe pagina 1');
  console.log(req.session.vizitat);});

app.get('/pagina2', function(req, res){
  res.send("Ai vizitat pagina 1 de " + req.session.vizitat + " ori");});

app.listen(4000);
```

Sesiunea de logare (login/logout)

Formularul de login dintr-un document html/ejs

```
<form method="post" action="/login">
  <label>
    Username: <input type="text" name="username" value="Ionut">
  </label>
  <label>
    Parola: <input type="password" name="parola" value="">
  </label>
  <input type="submit" value="Submit">
</form>
```

Pentru logare:

```
app.post('/login', function(req, res) {  
  var form = new formidable.IncomingForm();  
  form.parse(req, function(err, fields, files) {  
    user= verifica(fields.username, fields.parola);//verificarea datelor de login  
  });  
  if(user){  
    req.session.username=user;//setez userul ca proprietate a sesiunii  
    res.redirect('/logat');}  
  else  
    req.session.username=false;  
  });  
});  
app.get('/logat', function(req, res) {  
  res.render('pagini/logout',{'nume': req.session.username});  
});
```

Verificarea datelor de login de obicei se face comparand username-ul si parola cu datele stocate intr-un tabel dintr-o baza de date (puteti sa simulati asta printr-un fisier JSON cu userii).

In mod normal parola este criptata si se verifica sirul obtinut prin criptarea parolei data de utilizator la login si verificarea cu sirul deja criptat din tabel.

Pentru delogare:

```
app.get('/logout', function(req, res) {  
  req.session.destroy(); //distrugem sesiunea cand se intra pe pagina de logout  
  res.render('pagini/login');  
});
```

Delogarea presupune distrugerea sesiunii (aceasta operatie va sterge toate proprietatile setate in session).

Presupunem ca avem o pagina de logout la care utilizatorul ajunge dand click pe un link/buton.

Modulul EJS

- EJS este un view engine utilizat pentru a genera marcaj HTML cu ajutorul JavaScript (template-uri)
- instalare: `npm install ejs --save`

Pentru a folosi modulul ejs in generarea și afisarea paginilor de tip template (de obicei avand extensia ejs), este necesar să se specifice anumite configurații:

- fișierele de tip template trebuie salvate într-un director numit **views**
- trebuie setat motorul (*view engine*) care va fi utilizat pentru redarea acestor fișiere:
`app.set('view engine', 'ejs');`

Instrucțiunea **include**

-insereaza cod din fișierul specificat ca parametru

Sintaxa: `<%- include('cale-fisier') %>`

- se folosește de obicei pentru zonele de cod care apar pe mai multe pagini (header, footer, meniu, informații meta)

-recomandare: sa existe în **views** un subfolder de fragmente de cod de inserat (fișiere partiale) și un subfolder cu paginile aplicației.

Exemplu:

```
<header>
```

```
<%- include('../partiale/header'); %>
```

```
</header>
```

Exemplu

```
<!-- views/pagini/index.ejs -->
```

```
<!DOCTYPE html>
```

```
<html lang="ro">
```

```
<head>
```

```
  <%- include('./partiale/head'); %>
```

```
</head>
```

```
<body>
```

```
<header>
```

```
  <%- include('./partiale/header'); %>
```

```
</header>
```

```
<main>
```

```
  <div>
```

```
    <h1>EJS</h1>
```

```
    <p>EJS este un view engine utilizat pentru template-  
uri</p>
```

```
  </div>
```

```
</main>
```

```
<footer>
```

```
  <%- include('./partiale/footer'); %>
```

```
</footer>
```

```
</body>
```

```
</html>
```

```
<!-- views/partiale/head.ejs -->
```

```
<meta charset="UTF-8">
```

```
<title>Template cu EJS</title>
```

```
<style>
```

```
  body {padding-top:50px;}
```

```
  header{width:80%; border:1px solid red;}
```

```
  footer{width:80%; border:1px solid blue;}
```

```
</style>
```

```
<!-- views/partiale/header.ejs -->
```

```
<nav>
```

```
<a href="">Meniu 1</a>
```

```
<a href="">Meniu 2</a>
```

```
<a href="">Meniu 3</a>
```

```
</nav>
```

```
<!-- views/partiale/footer.ejs -->
```

```
<p>Footer: pagina creata cu EJS</p>
```

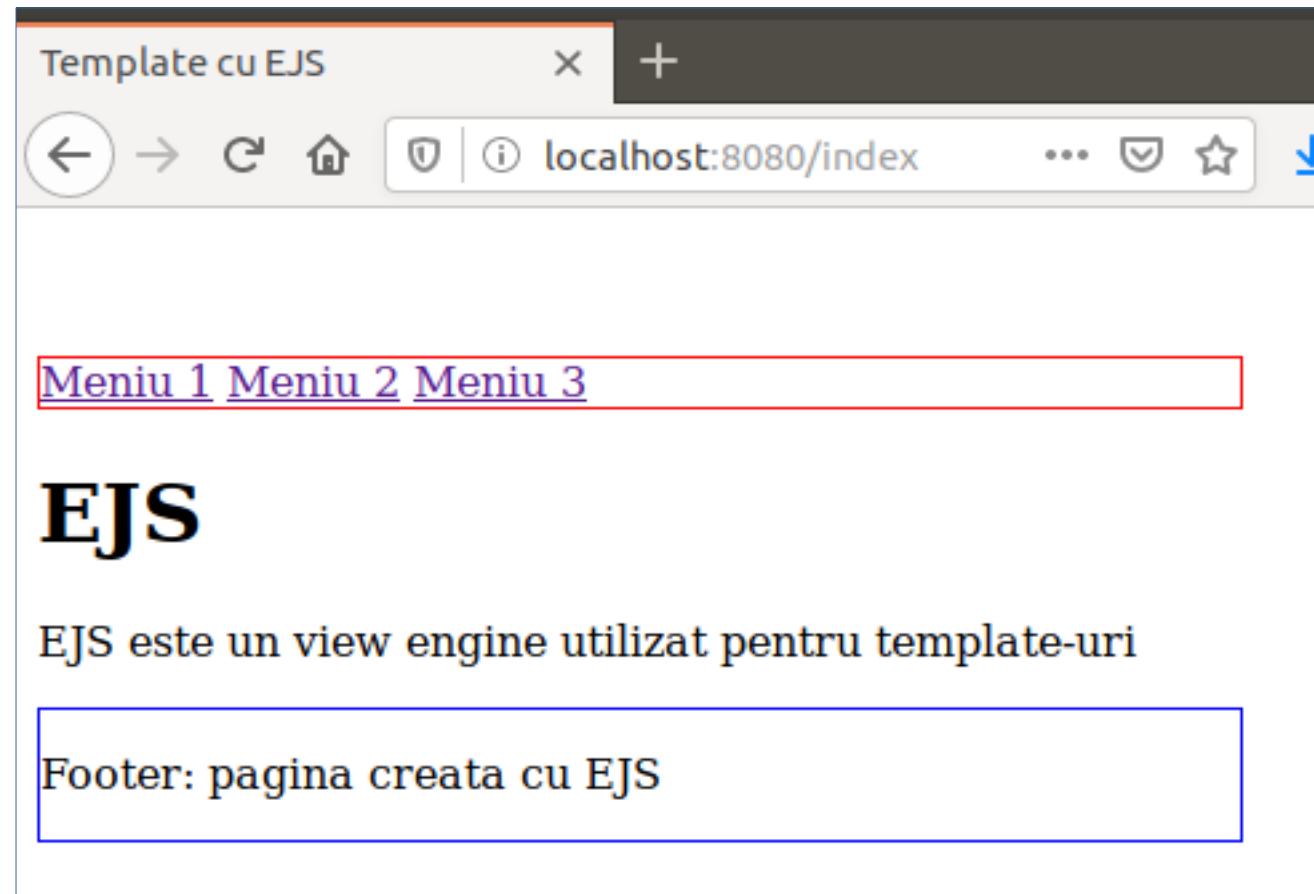
Metoda **render()**

-foloseste view engine-ul setat pentru a genera și a afisa pagina

Sintaxa: **response.render(cale-relativa-fisier, date)**
(calea relativa este relativa la folderul **views**)

app.js

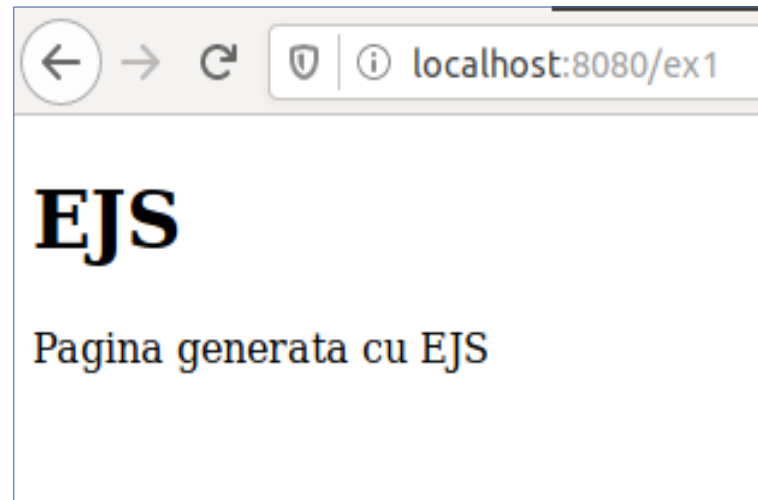
```
....  
app.get('/index',  
function(req,res){  
res.render('pagini/index');});  
.....
```



Metoda **render()**

ex1.ejs

```
.....  
<body>  
<h1><%= titlu %></h1>  
<p><%= continut %></p>  
</body>  
.....
```



app.js

```
.....  
app.get('/ex1', function(req,res){  
  res.render('pagini/ex1',{titlu:'EJS', continut:'Pagina generata cu  
  EJS'});});  
.....
```

Exemplu

form_carti.ejs (din views/pagini)

```
.....  
<body>  
<%- include("../partiale/carti") %>  
</body>  
.....
```

carti.js

```
.....  
app.get('/carti', function(req, res) {  
    var date = fs.readFileSync("carti.json");  
    var comenzi = JSON.parse(date);  
    res.render("pagini/form_carti",{carti:comenzi});  
});  
.....
```

carti.ejs (din views/partiale)

```
<article id="info">  
<h1>Cartile mele</h1>  
<% for(i=0;i<carti.length ;i++) {%>  
<div>  
<h2>Titlul: <%= carti[i].titlu %></h2>  
<p>Autorul: <%= carti[i].autor %></p>  
<p>Descriere: <%= carti[i].descriere %></p>  
<p>Pret: <%= carti[i].pret %></p>  
  
</div>  
<% } %>  
</article>
```