

Tehnici Web

CURSUL 11

Semestrul II, 2019-2020
Carmen Chirita

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

Utilizarea view engine-urilor pentru template-uri (EJS-Embedded JavaScript Templates)

- EJS este un view engine utilizat pentru a genera marcaj HTML cu ajutorul JavaScript (template-uri)
- instalare: `npm install ejs --save`
- un document EJS (de obicei cu extensia 'ejs') conține cod HTML și cod JavaScript care referă attributele transmise de la nivelul de logică a aplicației (de obicei serverul), codul JS fiind încadrat între delimitatorii `<%` și `%>`.

<https://ejs.co/>

Alte exemple de Node view engines

- Pug (fostul Jade)
- Mustache
- Dust
- Handlebars (express-handlebars)
- React

<https://expressjs.com/en/resources/template-engines.html>

EJS-Embedded JavaScript Templates

Pentru ca platforma Express să poată reda fișiere de tip template, este necesar să se specifice anumite configurații:

- fișierele de tip template trebuie salvate într-un director numit **views**
- trebuie specificat motorul (*view engine*) care va fi utilizat pentru redarea acestor fișiere:

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

Delimitatori folosiți în cadrul documentelor EJS pentru a încadra codul sursă JavaScript

- <%** utilizat pentru controlul fluxului de execuție al programului ca delimitator de început; nu produce nimic în HTML
- <%=** utilizat pentru evaluarea rezultatului expresiei conținute și plasarea acestuia în șablonul obținut, înlocuind caracterele speciale HTML prin codul lor
- <%-** utilizat pentru evaluarea rezultatului expresiei conținute și plasarea acestuia în șablonul obținut, fără a înlocui caracterele speciale HTML prin codul lor
- <%#** utilizat pentru comentarii, codul sursă JavaScript nu este executat și nici nu produce vreun rezultat
- <%%** utilizat pentru a reda secvența de caractere '<%' la nivelul paginii HTML care este generată pe baza sa
- %>** utilizat pentru controlul fluxului de execuție al programului ca delimitator de sfârșit
- %>** utilizat pentru eliminarea caracterelor '\n', în cazul în care acestea sunt conținute în codul sursă JavaScript
- <%_ , _%>** utilizat pentru eliminarea spațiilor de dinainte de (respectiv de după) el

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 parțiale) 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>
```

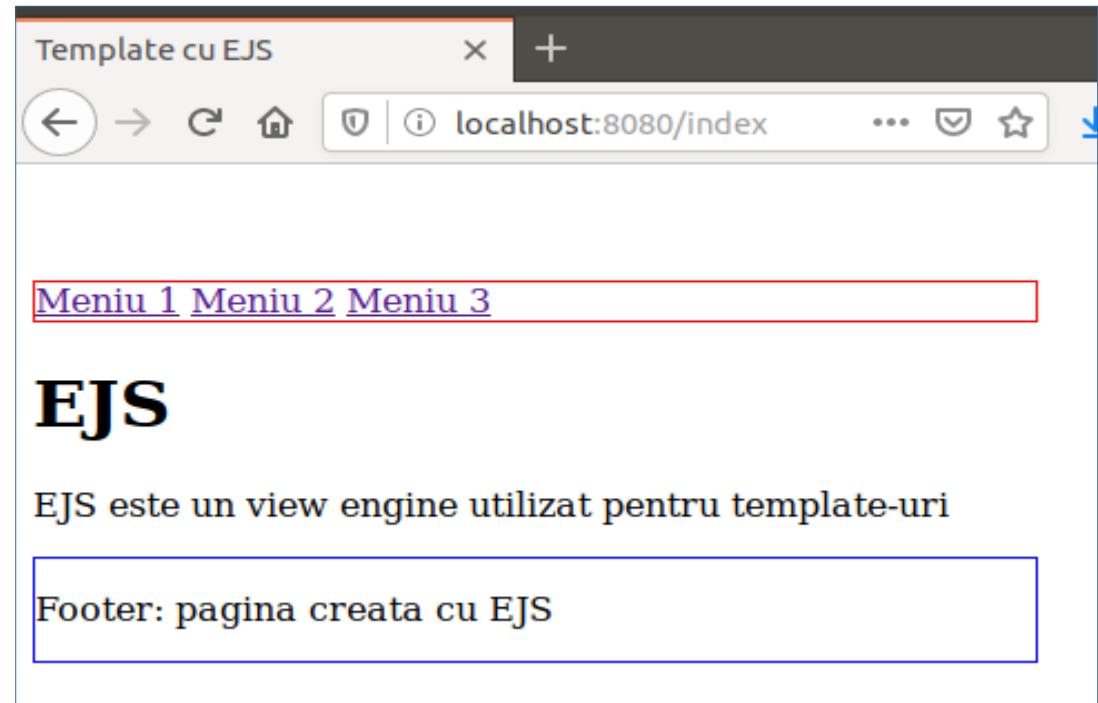
Metoda **render()**

-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

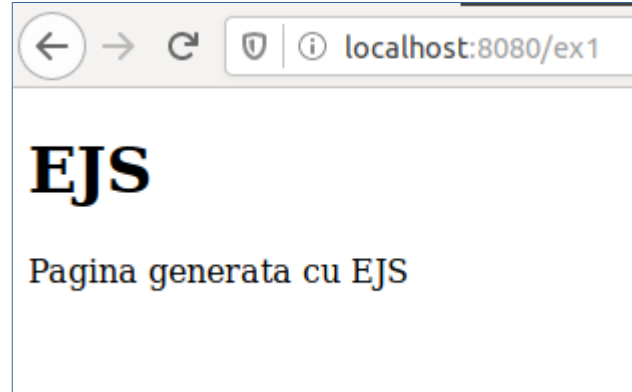
```
....  
app.get('/index', function(req,res){  
  res.render('pagini/index');});  
.....
```



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'});});  
.....
```

EJS-exemple

Afisarea unui vector de stringuri sub forma unei liste ordonate (în HTML) și a unui vector de obiecte sub forma de tabel (în HTML)

app.js

```
app.get('/for_vector', function(req, res) {
  aranjamente_flori=["bucet mic","bucet mare", "coroana", "salba","ghiveci"]
  flori=[
    { nume:"lalele", culoare:"rosu", nr_fire_bucet: 5, imagine:"lalele.jpg" },
    { nume:"orhidee", culoare:"roz", nr_fire_bucet: 10,
imagine:"orhidee.jpg" },
    { nume:"garoafe", culoare:"alb", nr_fire_bucet: 9, imagine:"garoafe.jpg" }
  ];
  res.render('for_vector', {vector_simplu: aranjamente_flori,
vector_obiecte:flori});
```

EJS-example

for_vector.ejs

```
<ol>
  <% for (var i = 0; i < vector_simplu.length; i++) { %>
    <li><%= vector_simplu[i] %></li>
  <% } %>
</ol>
```

- fiecare secventa de cod JavaScript se adauga intre <% %>
- <%= afiseaza valoarea variabilei scrise dupa el (vector_simplu[i])
- acoladele de la for (si alte instructiuni repetitive/conditionale) sunt **obligatorii**

EJS-exemple




for_vector.ejs

```
<table id="flori">
  <thead><tr><th>Nr</th><th>Nume</th><th>Culoare</th><th>Fire in
bucchet</th><th>Imagine</th></tr></thead>
  <tbody>
    <% for (var i = 0; i < vector_obiecte.length; i++) { %>
      <tr>  <td><%= i+1 %></td>
        <td><%= vector_obiecte[i].nume %></td>
        <td><%= vector_obiecte[i].culoare %></td>
        <td><%= vector_obiecte[i].nr_fire_bucchet %></td>
        <td></td>  </tr>
      <% } %>
    </tbody></table>
```

Lista creata cu EJS

1. buchet mic
2. buchet mare
3. coroana
4. salba
5. ghiveci

Tabel creat cu EJS

Nr	Nume	Culoare	Fire in buchet	Imagine
1	lalele	rosu	5	
2	orhidee	roz	10	
3	garoafe	alb	9	

EJS-exemple

if... else

<p>

```
<% var x=Math.random();
```

```
  if (x<0.5)    {%>
```

```
    <%= (x+"e mai mic decat 0.5") %>
```

```
<% } else {%>
```

```
    <%= (x+"e mai mare decat 0.5") %>
```

```
<% } %>
```

</p>

EJS-example

```
<p>
<%   culori=["red","green","blue"]
      var ind = Math.trunc(Math.random()*culori.length); %>
      Culoarea este: <span style='color:<%= culori[ind]
%>'>
      <%   switch (ind){
          case 0: %>   rosie
          <% break;
          case 1:%>   verde
          <% break;
          case 2:%>   albastra
          <% } %>
      </span>
</p>
```

EJS - diverse

daca vrem sa obtinem intr-un string (in loc sa fie afisat direct in pagina) umplerea unui template ejs cu date :

```
var date = {titlu:"Titlu", culori:["red","green","blue"]}
var template = fs.readFileSync("./template.ejs", "utf-8"); //modulul fs
var template_completat = ejs.render ( template , date );
pe care il putem apoi salva intr-un fisier html (care astfel devine static):
fs.writeFileSync("./html/fisier_completat.html", template_completat , 'utf8');
```

daca vrem sa folosim extensia html in loc de ejs putem pune in app.js:

```
app.set('view engine', 'html');
ejs=require('ejs')
app.engine('.html', ejs.renderFile);
```

daca vrem sa setam alt folder de views decat cel default:

```
app.set('views', 'templateuri');
```


Modulul **cookie-parser**

- folosit pentru definirea cookie-urilor
- date stocate în browser și trimise de server împreună cu răspunsurile la cererile clientilor
- pot fi utilizate pentru întretinerea sesiunilor
- instalare: **npm install cookie-parser --save**
- mai întâi modulul trebuie inclus în aplicația express:

```
var cookieParser = require('cookie-parser');  
app.use(cookieParser());
```

Crearea unui cookie

```
response.cookie(ume-cookie, valoare-cookie)
```

Exemplu:

```
var date_user={'ume':'Andrei','varsta':18}  
response.cookie('user', date_user);
```

Stergerea unui cookie

```
response.clearCookie(ume-cookie)
```

Accesarea cookie-urilor

```
request.cookies
```

Exemplu: la submiterea formei se creaza un cookie cu optiunea selectata iar la reincarcarea paginii se va păstra selectia

```
<html lang="us">  
<body>
```

form.ejs

```
<p>limba selectata: <%= selectedLang %></p>
```

```
<form action="formpost" method="post">  
  <select name="limba">
```

```
<% langs.forEach(function(lang) {%>
```

```
  <option <%= lang == selectedLang ? 'selected' : " %>><%= lang %></option>
```

```
<% }); %>
```

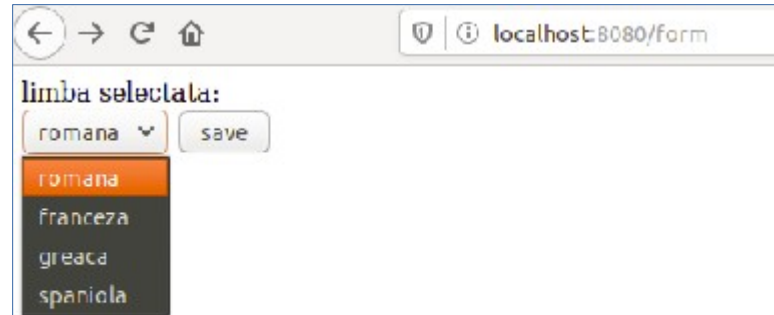
```
</select>
```

```
<button type="submit">save</button>
```

```
</form>
```

```
</body>
```

```
</html>
```



Exemplu: la submiterea formei se creaza un cookie cu optiunea selectata iar la reincarcarea paginii se va păstra selectia

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

var cookieParser = require('cookie-parser');
app.use(cookieParser());

app.set('view engine', 'ejs');

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

app.get('/form', function(req,res){
  res.render('pagini/form',{selectedLang:req.cookies.limba,
  langs : "['romana', 'franceza', 'greaca', 'spaniola']"});
});

app.post('/formpost', function(req,res){
  res.cookie('limba', req.body.limba);
  res.send('saved');
});
```

app.js

Gestiunea evenimentelor

- modulul `events` permite utilizarea de evenimente definite de utilizator
- pentru utilizarea funcționalității oferite de acest modul, este necesar să creăm o instanță a unui obiect din clasa `EventEmitter`

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

```
var EventEmitter = new events.EventEmitter();
```

- metoda `on()` este utilizată pentru înregistrarea handlerului evenimentului

```
eventEmitter.on('nume_event', eventHandler);
```

- metoda `emit()` este utilizată pentru declanșarea evenimentului

```
eventEmitter.emit('nume_event',[argumente])
```

Proprietăți ale metodelor on() și emit()

- putem înregistra mai multe funcții handler pentru un eveniment

```
eventEmitter.on('myevent', eventHandler1);
```

```
eventEmitter.on('myevent', eventHandler2);
```

- metoda `emit()` permite transmiterea unui set de argumente arbitrare funcțiilor handler asociate metodei `on()`

Exemplu:

```
eventEmitter.emit('event1',10,'a');  
eventEmitter.on('event1',function(arg1,arg2)  
{  
    for(i=0;i<arg1;i++) console.log(arg2);  
});
```

Modulul **formidable**

- folosit pentru upload de fisiere
- instalare: **npm install formidable --save**
- mai întâi modulul trebuie inclus în aplicația express:

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

Intr-o cerere de tip post sau get din app.js se creaza un obiect care conține datele din formular:

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

Modulul **formidable**

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

Pentru a accesa datele, trebuie parsate cu metoda

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

err //setat dacă exista vreo eroare (null altfel)

fields // un obiect din care preluam proprietatile
corespunzatoare campurilor formei cu excepția inputurilor de tip
„file” (proprietatea=valoarea atributului name).

Modulul **formidable**

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

```
<form>
```

```
<input type="text" name="nume">
```

```
<input type="number" name="varsta">
```

```
<input type="file" name="poza">
```

```
<input type="file" name="cv">
```

```
<button type="submit">Trimite</button>
```

```
</form>
```

Exemplu: fields.nume, fields.varsta
files.poza, files.cv

files.poza.name //numele fisierului (ex: floare.jpg)

files.poza.path //calea de upload (care trebuie setata în
evenimentul fileBegin)

Modulul **formidable**

Evenimente care trebuie tratate în cererea realizata cu post sau get

fileBegin //se declanseaza la începutul incarcarii fisierului

form.on("fileBegin", function(name,file){  ob cu proprietatile fis

//se seteaza calea de upload

file.path='cale_director' + file.name; //ca sa păstrăm numele initial al
fisierului din file.name
}

file //se declanseaza la sfârșitul incarcarii fisierului

form.on("file", function(name,file){

//aici putem:

verifica dimensiunea fisierului cu file.size
numele fisierului: file.name
calea: file.path }

Sesiuni

În momentul creării unei sesiuni, un user primește un session ID. Informațiile referitoare la sesiunea activă pot fi stocate la client, de obicei într-un cookie, sau la server, folosind un memory store sau alt tip de storage.

Un memory store este cel mai simplu de implementat, dar presupune stocarea informațiilor într-o memorie volatilă (informațiile de sesiune vor fi șterse la restartarea serverului).

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

Sesiuni

Obiectul sesiune se regaseste in campul session al parametrului request (din cererile get si post)

```
app.get('/pagina1', function(req, res) {  
    //putem sa setam intr-o cerere o proprietate a sesiunii:  
    req.session.vizitat="da";  
    //.....  
});
```

```
app.get('/pagina2', function(req, res) {  
    //si o putem accesa  
    if (req.session.vizitat=="da" ) console.log("a vizitat pagina1");  
    res.render('pagina2', {vizitat: req.session.vizitat}); //putem chiar trimite  
valori din ea catre pagina  
});
```

Sesiuni - login/logout

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

Sesiuni - login/logout

Pentru logare:

```
app.post('/', 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  
  else  
    req.session.username=false;  
  res.render('html/index',{user: req.session.username});  
});
```

Sesiuni - login/logout

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('html/logout');  
});
```