Laborator 04 - (X)HTML, CSS

- Obiective:
  - Înțelegerea structurii HTML și a diferențelor dintre HTML, xHTML și XML
  - Înțelegerea conceptelor fundamentale ale CSS
  - Înțelegerea sintaxei CSS și a modului de folosire a stilurilor în pagini web

HTML

HTML a apărut ca un răspuns la necesitatea de a descrie un set de informații într-un document, delimintând anumite secțiuni precum: titluri, paragrafe, liste, legaturi cu alte documente și suplimentându-le cu formulare interactive, imagini și alte tipuri de obiecte dinamice (ex: Adobe Flash, Java Applets etc). De asemenea, în HTML pot fi incluse secvențe de cod JavaScript care sunt executate de browser.

Este foarte important de reținut că markup-ul trebuie să definească un document **din punct de vedere semantic** și nu din punct de vedere al reprezentării vizuale.

HTML este o instanță a metalimbajului SGML (la fel cum este XML, dar **fără** a fi un descendent al XML). Acest lucru înseamnă că în HTML vom întâlni multe elemente familiare din XML (tag-uri, atribute, structură nested etc.). Schema de mai jos ilustrează relația dintre SGML, HTML, XML și XHTML.

Noțiuni generale despre HTML

HTML are o structură arborescentă al cărei rădacină este tag-ul `<html>`. Copiii acestuia sunt tagurile `<head>` si `<body>`. Tagul `<head>` conține informații generale despre document (meta-informații - "informatii despre informatii") si conținutul acestuia. Tagul <body> contine informația efectivă asociată paginii web.

Există numeroase taguri, cu semnificații diferite. Pentru a vedea o listă cu toate tagurile suportate de standard consultați această referință (nu conține HTML5).

Tagurile HTML pot prezenta și o serie de atribute care îi definesc comportamentul. De exemplu:

- atributul `href` asociat tag-ului `<a>` - specifică adresa documentului către care trimite un link.
- atributul name asociata tagului `<input>` - specifică numele asociat valorii introduse de utilizator.
- atributele `onclick`, `onblur`, `onchange` etc. - sunt utile pentru "embedding"-ul unui limbaj "client-side" si permit specificarea de actiuni particulare pentru diverse evenimente. Pot fi comparate cu listener-i.
- atributul `style` - permite specificarea directă ("inline") a stilurilor CSS pentru formatarea vizuală a elementului curent.

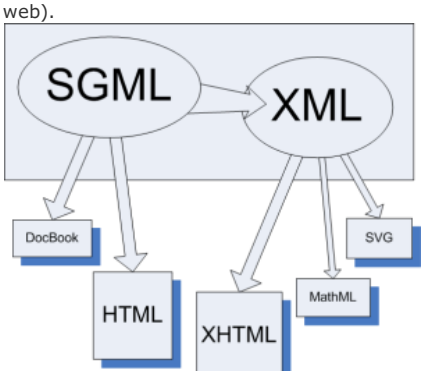
O obsevație importantă este că, deși nu toate elementele suportă aceleași atribute, ultimele două tipuri de atribute de bază (stiluri și evenimente) se regăsesc la toate elementele.

DOCTYPE

Un document HTML valid, trebuie să specifice (chiar la începutul documentului) un `<!DOCTYPE>`. Acest tag specifică browser-ului cum trebuie interpretat documentul (ca HTML 4.0, ca XHTML, ca HTML5 etc.). Pentru a vedea mai multe tipuri de declarații puteți consulta această listă.

În același timp, doctype-ul definește și modul în care documentul trebuie **validat**; validarea este o unealtă foarte folositoare pentru dezvoltatori pentru că poate prinde erori subtile în structura documentului și este o bază foarte bună de pornire în "relația" cu browserul (dacă un document se validează și browserul implementează corect standardele, comportamentul documentului în browser devine previzibil).

Ca validator, este recomandat să folosiți cel oferit de w3c (organizația responsabilă cu producerea diverselor standarde pentru web).



- Repartizare teme
- Catalog PW

Laboratoare

- Laborator 01 - Introducere
- Laborator 02 - BD in PHP
- Laborator 03 - Arrays, Magic Methods
- Laborator 04 - (X)HTML, CSS
- Laborator 05 - Formulare HTML, Persistența Datelor
- Laborator 06 - Securitate I
- Laborator 07 - Securitate II
- Laborator 08 - JavaScript
- Laborator 09 - DOM scripting
- Laborator 10 - AJAX
- Laborator 11 - Web Frameworks

Teme

- Tema 01 - API pentru BD
- Tema 02 - Template System & Controller
- Tema 03 - Generator de formulare
- Tema 04 - Tree Web UI

Login

XHTML

Explozia industriei web și "lejeritatea" cu care browserele parseaza și interpretează codul HTML (în multe situații diferit), și faptul că o pagina incorectă va fi totuși afișată (în unele cazuri acceptabil sau chiar corect), au condus la o situatie (actuală) în care majoritatea paginilor web conțin erori.

XHTML este o incercare de a contracara acest fenomen. Spre deosebire de ultimele versiuni HTML, XHTML ca si standard-de-limbaj are reguli stricte.
Enumeram cateva dintre ele:

- taguri ca `<b>`,`<i>`,`<font>` nu mai sunt admise, pentru că definesc layout și nu semantica documentului
- tagurile trebuie sa fie intotdeauna inchise; tagurile simple trebuie inchise cu `/>` (spre exemplu, tagul `<br>` din HTML 4.0 este valid scris in XHTML astfel: `<br />`)
- toate tagurile sau atributele acestora trebuie scrise lowercase iar atributele trebuie să aibă valorile prinse între "" (tagurile `<A HREF=…>` nu sunt valide in XHTML; varianta validă este `<a href="…">`).

HTML 5

HTML5 este urmatorul standard propus pentru a inlocui HTML 4.01 si XHTML 1.0. Scopul sau este de a reduce nevoia de aplicatii bazate pe plug-in-uri precum Adobe Flash, Microsoft Silverlight, Apache Pivot, si Sun JavaFX.

Acest standard este inca in progres de dezvoltare insa unele parti au fost deja implementate la nivel de browser.
Diferentele majore fata de versiunea anterioara sunt:

- Reguli noi de parsare axate pe flexibilitate și compatibilitate, ce nu mai sunt bazate pe SGML
- Abilitatea de a folosi inline SVG și MathML în text/html
- Elemente noi – `article`, `aside`, `audio`, `canvas`, `command`, `details`, `datalist`, `dialog`, `embed`, `figure`, `footer`, `header`, `hgroup`, `keygen`, `mark`, `meter`, `nav`, `progress`, `output`, `rp`, `rt`, `ruby`, `section`, `source`, `time`, `video`
- Tipuri noi de controale pentru formulare – dates and times, email, url, search
- Atribute noi – ping (pe `a` și `area`), charset (pe `meta`), `async` (pe `script`)
- Atribute globale (ce pot fi aplicate pentru fiecare element in parte) – `id`, `tabindex`, `hidden`, `data-*` (custom data attributes)
- Formulare vor primi suport pentru metodele PUT și DELETE nu doar pentru GET și POST
- Se renunță la elementele deprecated – `center`, `font`, `frameset`, `strike` etc.

Error handling

Un browser HTML5 (`text/html`) va fi flexibil in parsarea sintaxei și va corecta eventualele erori. HTML5 este creat astfel incat vechile browsere să ignore construcțiile proaspăt introduse. În contrast cu HTML 4.01, specificația HTML5 oferă reguli detaliate pentru lexing și parsare, cu scopul unificării comportamentului diferitelor browsere în cazul sintaxei incorecte.

Gasiti aici câteva exemple ale forțelor proaspete aduse de HTML5.

HTML și PHP

Paginile clasice HTML au extensia `.html` sau `.htm`. În instalarea default de (L/W)AMP, serverul este configurat să treacă prin interpretorul PHP doar paginile care au extensia `.php`. Pentru a putea interpreta codul php inserat in interiorul unei pagini scrise in format HTML, extensia paginii trebuie modificata in `.php` sau serverul trebuie configurat pentru a trece și paginile `.html` prin interpretorul de php. Codul php integrat in pagina trebuie cuprins intre taguri php, astfel:

```
<head>
    ...
</head>

<body>
    ...
    <?php
        ... cod php ...
    ?>
    ...
</body>
```

Referințe:

- Standardele: Specificația HTML 4.01, Specificația HTML5
- Sitepoint HTML reference
- Galerie HTML5, HTML5 Demos

CSS

Limbajul HTML nu a fost conceput pentru formatarea vizuală a documentului. Taguri precum `<p>` asociat cu un paragraf sau `<h1>` asociat cu un heading nu au decât funcția de a delimita portiuni logice ale documentului.

Începând cu specificația HTML 3.2 au fost introduse tagurile `<font>` și întregul set de atribute de formatare (culori, dimensiuni, fonturi, etc), provocând un haos la nivelul dezvoltatorilor de site-uri. Pentru a obține formatarea dorită, fiecare porțiune de document trebuia să primească atributele necesare (de culoare, de font, etc), chiar daca ele erau deseori aceleași.

Pentru a rezolva aceasta problema, a fost introdus un *nivel de prezentare* care să izoleze elementele de formatare/stil și să le elimine din documentul HTML propriu-zis. Acest lucru s-a realizat prin introducerea CSS (Cascading Style Sheets). CSS permite definirea stilurilor vizuale separat - în fisiere separate (*.css), între taguri `<style>`, în pagină sau în atributul `style="…"` al elementului destinație.

Un exemplu de clasă în css ar fi:

```
/* Se aplică pe elemente ca <p class="content">Text-ul meu<p> */
.content {
    padding: 50px 0 0 15px;
    font-family:Tahoma;
    color:#333333;
    font-size:11px;
    font-weight:bold;
}
```

Stilul content formateaza un tag HTML astfel: impune un padding de 50 de pixeli în partea de sus și 15 pixeli în partea stângă (mai multe detalii la secțiunea box-model), apoi seteaza fontul default pentru tagurile care au acea clasă, culoarea acestuia, dimensiunea fontului, si in final specifica in plus ca fontul redat sa fie bold.

Includerea unui fisier de stiluri .css intr-un document HTML se face prin tagul `<link>`. Acesta trebuie introdus în interiorul tagului <head> al documentului HTML.

```
<link rel="stylesheet" type="text/css" href="[nume-css].css" >
```

Aplicarea stilurilor pe elemente. Selectori.

Un fisier CSS poate conține mai multe seturi de reguli.

Există mai multe tipuri de selectori în sintaxa CSS. Selectorii au rolul sugerat de nume - specifică modul de selecție al elementelor asupra cărora se va aplica regula. Primul tip de selector este cel utilizat mai sus (se observă prefixul .). El asociază un set de reguli cu o "clasă". Toate tagurile ce au setat atributul class="content" vor fi formatate în consecință.

Al doilea tip de selector asociază un set de reguli cu toate elementele HTML de un anumit tip din pagină, de exemplu:

```
/* se aplică pe toate elementele <p> din pagină */
p {
    width: 750px;
    font-family: Tahoma;
}
```

A treilea tip de selector asociază regulile elementului care are id-ul specificat. Ea funcționează astfel:

```
/* se aplică pe elementul ca <a id="noLine">Link fara subliniere</a> */
#noLine {
    text-decoration:none;
}
```

Mai există două tipuri de selectori: pseudo-clase - `:hover`, `:visited` etc. și pseudo-elemente - `:before`, `:after`, `:first-letter` etc. Aceste elemente nu sunt tratate în laborator, însă puteți trece în revistă referințele dacă doriți mai multe informații.

Se observă similaritatea cu prima metodă: proprietatea id inlocuiește class, iar prefixul . este inlocuit cu #.

Combinarea selectorilor

Pe langa aceste variante, selectorii se pot combina. De exemplu:

```
.myClass {
 /* se aplică tuturor elementelor care au clasa myClass */
}

a.myClass {
 /* se aplică doar elementelor <a> care au clasa myClass */
}

p.myClass {
 /* se aplică doar elementelor <p> care au clasa myClass
    a se observa că stilurile de aici pot fi complet diferite
    de cele pentru <a class="myClass">
 */
```

```
}

#contactForm .hidden {
 /* se aplică elementelor care au clasa hidden și sunt descendenți ai
 (în interiorul) elementului cu id-ul contactForm */
}

#errorBox p span.error.highlight {
 /* se aplică elementelor span, care au atașate atât clasa error cât și clasa
 hilight (class="error hilight") și se află în interiorul unor tag-uri <p> care
 la rândul lor se află în interiorul elementului cu id-ul errorBox */
}
```

Diferența între atributele id și class

Atributele `class` și `id`, deși sunt similare din punct de vedere al sintaxei CSS, sunt totuși diferite. Atributul class se referă întotdeauna la "o clasă de taguri", ce pot avea (de exemplu) acelasi stil. Atributul id se refera la un tag UNIC. Desi browserele tolerează acest aspect în afisarea paginilor web, este considerat eronat a avea doua taguri html cu același id. Așadar, este recomandat să folosim class; id se poate folosi doar atunci când suntem siguri că elementul respectiv va fi unic. Utilitatea proprietății id va fi mai clară în laboratoarele ce urmeaza (și în legătura cu JavaScript).

Tipuri de proprietăți

CSS suportă un număr mare de tipuri de reguli ce se pot aplica elementelor. Cele mai uzuale sunt:

- `color` - foreground color, culoarea textului din elementul specificat
- `background` - fundalul unui element, compune mai multe proprietăți (`background-color`, `background-image`, `background-position`, `background-repeat`)
- `float`, `position`, `display` - modul de poziționare al elementului (detalii în secțiunea următoare)
- `top`, `right`, `bottom`, `left` - poziția efectivă a elementului în context
- `margin`, `padding`, `border`, `width`, `height` - dimensiunile unui element din punct de vedere al box model-ului (detalii în secțiunea următoare)
- `font` (`font-family`, `font-size`, `font-weight` etc.), `text-transform`, `text-decoration`, `text-align`, `text-shadow`.

Puteți vedea lista completă a proprietăților CSS aici.

Poziționarea elementelor

Fiecărui element i se asociază de către browser la randare un spațiu - bloc - în care vor fi afișate componentele sale interne. Poziționarea acestui bloc depinde de proprietățile `display` și `position` ale elementului.
`display` poate lua următoarele valori:

- `block` - implicit pentru elemente ca `<div>`, `<p>`, `<blockquote>` - elementul suportă definirea explicită a dimensiunilor și își "rezervă" spațiul pe orizontală forțând o linie nouă
- `inline` - implicit pentru elemente ca `<span>`, `<img>`, `<strong>`, `<em>` - dimensiunile sunt determinate implicit în funcție de conținut și nu forțează o linie nouă
- `inline-block` - permite specificarea dimensiunilor, dar fără a forța o linie nouă

Există mai multe moduri de a poziționa un element în pagină, folosind proprietățile `position`, `display`, `float` și `clear`.

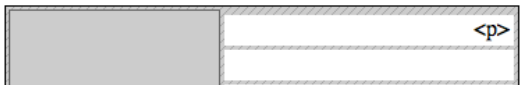
`position` poate lua următoarele valori:

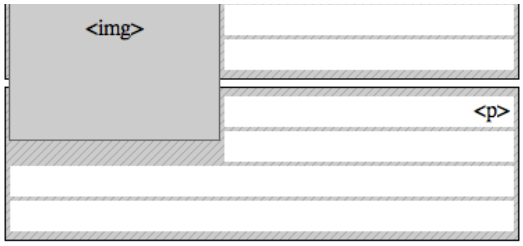
- `absolute` - poziționează elementul absolut în funcție de cel mai apropiat părinte care are `position absolute` sau `relative`
- `relative` - poziționează elementul relativ la poziția în care trebuia așezat inițial; celelalte elemente se poziționează relativ la poziția inițială a elementului
- `fixed` - poziționează fix elementul (indiferent de scroll sau alte evenimente din pagină)
- `inherit` - copiază `position` de la părinte
- `static` (valoarea implicită) - nu modifică în niciun fel poziționarea elementului, el este așezat în poziția firească din pagină

Proprietatea `float` poate lua următoarele valori: `left`, `right`, `none`. Dacă este `left` sau `right`, elementul este împins în stânga sau în dreapta până atinge marginea elementului care îl conține sau marginea unui alt element cu `float`. Considerați următorul cod:

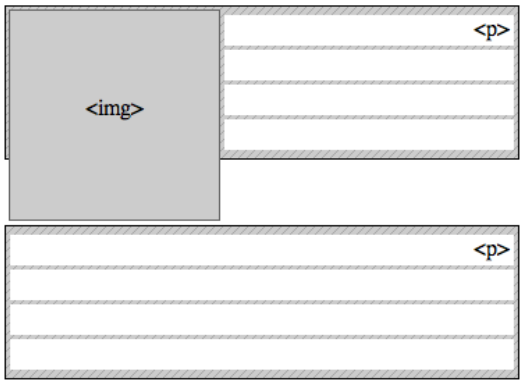
```
<p>
  <img src="image.png" alt="">
   text text … text text
</p>
<p>text text … text text</p>
```

Dacă `<img>` ar avea float left, layout-ul ar arăta în felul următor:





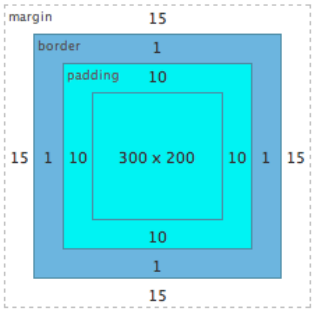
Proprietatea `clear` poate lua aceleași valori ca `float`; forțează elementele cărora li se aplică să se așeze sub float-uri. Dacă al doilea paragraf din exemplul anterior ar avea `clear: left`, atunci layoutul ar fi:



Prezentarea acestor proprietăți a fost simplificată din rațiuni de spațiu, consultați referința pentru informații complete.

Box model

Calcularea dimensiunilor unui element în CSS nu se face doar prin proprietățile `width` și `height`, ci și prin `margin`, `border` și `padding`. Figura de mai jos ilustrează calcularea spațiului ocupat de un element.



Aceasta ar corespunde următoarelor reguli:

```
div {
    width: 300px;
    height: 200px;
    padding: 10px 10px 10px 10px; /* putea fi scris doar ca 10px */
    border: 1px solid #000;
    margin: 15px; /* aici s-a folosit varianta scurtă de a declara toate marginile */
}
```

Observați că dimensiunile efective ale acestui element în document sunt 326x226px și nu 300x200px cum specifică `width` și `height`.

Cascada CSS

După cum ați putut vedea, asupra unui element se pot aplica mai multe seturi de reguli definite în CSS. Pentru a decide ordinea în care trebuie aplicate stilurile, standardul definește cascada (***C**ascading* **S**tyle**S**heets). Procesul de selecție a regulilor este următorul:

1. se găsesc toate declarațiile care se aplică unui anumit element
2. se sortează în funcție de **origine** și de nivelul de **importanță**
3. declarațiile cu același nivel de importanță și cu aceeași origine se ordonează după **specificitate**
4. dacă au aceeași origine, importanță și specificitate se aplică în ordinea în care au fost declarate

Originea stylesheet-urilor și nivelul de importanță

Din punct de vedere al originii, se disting trei tipuri de stylesheet-uri (în ordinea crescătoare a priorității):

1. **User agent stylesheets** - stiluri predefinite de browser. Dacă încercați o pagină fără nici un stil definit, veți observa că există anumite stiluri implicite aplicate elementelor (spre exemplu, linkurile au implicit culoarea albastră și sunt subliniate). Unele dintre aceste stiluri diferă ușor între browsere, de aceea este recomandată folosirea unui stylesheet de reset, pentru a porni de la același bază în orice browser. Pentru mai multe detalii, puteți urmări această discuție.
2. **Author stylesheets** - stiluri definite de autorul paginii web (cele incluse în pagină prin `<link>`, `<style>` sau prin atributul `style="..."`)
3. **User stylesheets** - unele browsere oferă utilizatorilor posibilitatea de a supradefini stilurile paginilor și de a aplica fișierele lor "deasupra" celor definite de autorii paginilor și celor definite de browser.

Din punct de vedere al importanței, avem două niveluri: reguli importante și reguli normale. Regulile importante se disting prin prezența declarației `!important` și au prioritate în fața regulilor normale. `!important` trebuie adăugat la sfârșitul regulii, chiar înainte de `;`. `!important` trebuie folosit doar unde este neapărat necesar, folosirea nejustificată duce la complicarea stylesheet-urilor și le face greu de întreținut și modificat.

```
p {
  color: blue !important;
}
```

În funcție de origine și importanță, ordinea aplicării regulilor este (în ordinea crescătoare a priorității):

1. reguli la nivel de browser (user-agent stylesheets)
2. reguli cu importanță normală din user stylesheets (cele aplicate de utilizator în browser)
3. reguli cu importanță normală din author stylesheets (cele definite de pagina web)
4. reguli importante din author stylesheets (definite de pagina web)
5. reguli importante din user stylesheets (definite de utilizatorul browserului)

Specificitatea elementelor

După cum am menționat anterior, dacă două seturi de reguli au aceeași origine și aceeași importanță, ele sunt ordonate în funcție de specificitate. Standardul definește modul de calcul al specificității. Specificitatea se poate calcula după următorul algoritm:

1. dacă declațiile provin din atributul `style="..."` al elementului (stiluri inline), au cea mai mare specificitate.
2. se numără selectorii de id-uri (`#myElement`) din regulă. Dacă regulile au același număr de selectori de id-uri, treceți la pasul 3.
3. se numără selectorii de clase (`.myRules`) din regulă și numărul de pseudo-clase (`:hover`). Declarația cu numărul total mai mare de astfel de selectori are specificitate mai mare. Dacă au același număr, treceți la pasul 4.
4. se numără selectorii de tag (`div`) și pseudo-elemente (`:first-letter`). Declarația cu numărul total mai mare de astfel de selectori are specificitate mai mare. Dacă au același număr, treceți la pasul 4.
5. dacă în toți ceilalți pași regulile au ieșit egale ca specificitate, se consideră ordinea în care au fost declarate

Pentru a ușura acest proces, se poate considera următoarea convenție: specificitatea unui set de reguli este un număr de forma `xabc`, unde:

- `x` - poate lua valoarea 0 (setul de reguli este declarat în cadrul unui tag `<style>` sau într-un fișier extern) sau 1 (setul de reguli este declarat inline, prin atributul `style` al elementului)
- `a` - numărul de selectori id
- `b` - numărul de selectori de clasă
- `c` - numărul de selectori de tag

Dacă numărul atașat unui set de reguli este mai mare decât al altuia înseamnă că setul are specificitate mai mare și se va aplica în detrimentul celuilalt. De exemplu:

```
#content #myId p.myclass {
 /* Specificitate: 0211 */
 color: red;
}

html body div#myId p {
 /* Specificitate: 0104 */
 color: blue;
}

html body#content div p {
 /* Specificitate: 0104 */
 color: yellow;
}
```

```
<html>
    <head></head>
```

```
<body id="content">
 <div id="myId">
  <p class="myClass" style="color: gray;">Test</p>
 </div>
</body>
</html>
```

Textul va avea culoarea gri, deoarece specificitatea stilului inline e cea mai mare (1000 > 0211 > 0104 > 0104).

Referințe:

- Sitepoint CSS reference

Exercitii

Descărcați această arhivă și plasați fișierele într-un director accesibil de către serverul vostru web.

1. (1p) Creați o pagină simplă HTML 4.01 în care să vă prezentați un hobby. Pagina ar trebui să aibă minim 10 elemente distincte. Nu puneți accent pe aspectul paginii, însă validați structura. Schimbați `<!DOCTYPE>` la XHTML strict și repetați validarea până când nu mai sunt erori.
   Hint: Pentru validare
2. (1p) Creați o pagină simplă HTML5 și folosiți următoarele taguri: `<audio>`, `<video>`, `<header>`, `<footer>`.
   Hint: Dacă rulați în Firefox, țineți cont că nu suportă decât anumite formate de fișiere.
3. (1p) Calculați specificitatea pentru fiecare dintre regulile de mai jos:

```
p.message {
  color: green;
}
#home #warning p.message {
  color: yellow;
}
#warning p.message {
  color: white;
}
body#home div#warning p.message {
  color: blue;
}
p {
  color: teal;
}
* body#home>div#warning p.message {
  color: red;
}
#warning p {
  color: black;
}
```

4. (1p) Creați un user stylesheet și aplicați-l în browser peste un site la alegerea voastră pentru a-i modifica aspectul într-un mod vizibil.
   Hint: userstyles.org - instalati mai intai plugin-ul pentru Firefox sau Chrome de pe aceeasi pagina.
5. (2p) Reparați pagina din directorul `ex5/` pentru a o face să arăte ca în acest screenshot. Trebuie să modificați doar css-ul.
6. (4p) Folosind resursele din directorul `ex6/`, recreați pagina prezentată în acest screenshot. Puteți folosi fie HTML 4.0, fie XHTML 1.0. Pagina trebuie să fie validă și să se apropie cât mai mult vizual de screenshot-ul prezentat. Puteți folosi Firefox sau Chrome ca browser de referință.

Bonus:

1. (1p) Aplicați o tranziție și o transformare din specificația CSS3 peste pagina pe care ați creat-o la punctul anterior.
2. (1p) Modificați markup-ul pentru a folosi cât mai multe elemente specifice HTML5 și validați rezultatul.