Tartalomjegyzék

Bevezetés

Az internet eredete

1957. október 4-én rendkívüli esemény történt: a Szovjetunió sikeresen Föld körüli pályára állította az első műholdat. Ennek az eseménynek a közvetlen hatására jött létre az Amerikai Védelmi Minisztérium ARPA (*Advanced Research Projects Agency*) részlege. A legismertebb projektjük az internet létrehozása volt. Azonban a különböző hálózati protokollok elszaporodása nagyon hamar probléma lett, amint megpróbálták az egyes hálózatokat összekapcsolni. Viszont már volt egy megoldás a láthatáron, mégpedig Robert Kahn által, aki egy műholdas csomagalapú hálózati projekten dolgozott.. Olyan szabályokat alkotott, amelyek egy sokkal nyíltabb hálózati architektúrát tettek lehetővé, mint az addig használt ARPANET-es protokollok. Később csatlakozott hozzá Vinton Cerf, és ketten létrehoztak egy olyan rendszert, amely egy új szabvány segítségével eltüntette az egyes hálózatok közti különbségeket. A szabványt az *Internet Transmission Control Program* névre keresztelték. A specifikáció háttérbe szorította a hálózatok szerepét, és az adatátvitel integritásának felügyeletét a gazdaszámítógép hatáskörébe helyezte. Így végre lehetségessé vált szinte az összes hálózat egyszerű összekapcsolása. 1981-re megjelent a végleges specifikáció, majd 1982-ben az USA-n kívüli kapcsolatokat átalakították az új „TCP/IP” protokoll használatára. Az általunk is ismert internet megérkezett.

A World Wide Web létrehozása

A kilencvenes évek elején a Gopher egy népszerű adatkezelő rendszer volt, amelyben menükön keresztül lehetett fájlokhoz, különböző erőforrásokhoz vagy más menükhöz hozzáférni. A Gophert a Minnesotai Egyetem készítette. 1993 februárjában bejelentették, hogy licencdíjat fognak kérni az általuk készített Gopher kiszolgáló kódja után, amelyet referenciaként lehetett használni. Ennek hatására sok szervezet alternatívát kezdett keresni. Tim Berners-Lee egy információkezelő rendszeren dolgozott, amelyben a szövegek hivatkozásokat tartalmazhattak más munkákra, így az olvasó könnyen átugorhatott egyik dokumentumról a másikra. Az általa csak hiperszövegként (*hypertext*) elnevezett tartalmak megjelenítéséhez készített egy kiszolgálót, amelyet WorldWideWeb-nek neveztek el. A szoftver már 1991-ben megjelent, de még szükség volt két dologra, mielőtt elterjedhetett

volna, és teljesen lecserélte volna a Gophert. 1993. április 13-án a CERN szabadon hozzáférhetővé tette a WorldWideWeb forráskódját, így bárki felhasználhatta és módosíthatta a kódot anélkül, hogy fizetnie kellett volna érte.

Még ugyanebben az évben az NCSA (*National Center for Supercomputing Applications*) Mosaic névvel kiadott egy alkalmazást, amely egy webböngésző és egy Gopher kliens ötvözete volt. Eredetileg csak Unix gépeken volt elérhető és csak forráskód szinten, de 1993 decemberében az új verziója már Macintosh és Windows telepítőkkel érkezett. A Mosaic és vele együtt a web népszerűsége rohamosan nőtt.

A böngészők háborúja

Később a web népszerűsége kereskedelmi érdeklődést is kiváltott. Marc Andreessen elhagyta az NCSA-t, és Jim Clarkkal közösen megalapították a Mosaic Communicationst, amelyet később Netscape Communications Corporationnek neveztek el, majd elkezdtek dolgozni a később Netscape Navigatorként ismert böngészőn. Ennek az első verziója 1994 decemberében jelent meg.

A Netscape és a Microsoft közötti harc hamarosan elérte a tetőpontját, és mindketten megpróbáltak új funkciókkal előrukkolni, hogy megnyerjék maguknak a fejlesztőket. Ez később a „böngészők háborúja” néven vált ismertté. Az Operának ezalatt egy kisméretű, de stabil pozíciója volt, és innovatív fejlesztésekkel, valamint a szabványok támogatásával próbáltak fennmaradni.

A webes szabványok eljövetele

A böngészőháború alatt a Microsoft és Netscape főleg az új funkciók fejlesztésére koncentrált ahelyett, hogy az alapfunkciókat javították volna ki rendesen. Saját, védett funkciókat építettek be a böngészőkbe, emellett olyanokat valósítottak meg, amelyek már léteztek a többi böngészőkben, de nem voltak kompatibilisek egymással.

A webfejlesztők ekkor egyre inkább arra voltak kényszerítve, hogy a valódi fejlesztés helyett az ilyen zavaros helyzetek megoldásával foglalkozzanak. Néha arra volt szükség, hogy két változatot készítsenek egy oldalból a két nagy böngésző számára, amelyek gyakorlatilag teljesen megegyeztek. Egyesek egyszerűen csak az egyik böngészőt támogatták, és a másik böngésző felhasználóit letiltották az oldalról. A munka így rémálommá vált. következmények.

A W3C megalakulása

1994-ben Tim Berners-Lee megalapította a W3C-t (*World Wide Web Consortium*), a CERN, a DARPA (ez lett az ARPA új neve), valamint az Európai Bizottság támogatásával. A W3C a különböző protokollok és technológiák szabványosítását tűzte ki célul maga elé egy olyan internet felépítésének érdekében, amelyen az információ a világ lehető legnagyobb részére elérhetővé válhat. A gyártók csak akkor kell betartsák a W3C dokumentumokban foglaltakat, ha szeretnék, hogy a termékük megfeleljen a W3C feltételeinek.

A Web Standards projekt

1998-ban a böngészőpiacot az Internet Explorer 4 és a Netscape Navigator 4 uralta. Megjelent az Internet Explorer 5 béta verziója, amely már egy új, saját fejlesztésű HTML-t támogatott. Ennek eredményeképpen a professzionális webfejlesztők és webdesignerek egy csoportja összeállít, és együtt megalakították a WaSP-ot (*Web Standards Project*). Az ötlet alapjában az volt, hogy a W3C dokumentumait szabványnak nevezzék ajánlások helyett, így talán sikerülhet meggyőzni a Microsoftot és a Netscape-et arról, hogy támogassák őket.

A webes szabványok felemelkedése

2000-ben a Microsoft kiadta az Internet Explorer 5 Macintosh verzióját. Ez az Opera megfelelő CSS és HTML támogatásával együtt már megadta a lehetőséget a webfejlesztők és webdesignerek számára, hogy először tapasztalhassák meg a szabványos weboldalak fejlesztésének kényelmét. 2003-ban Dave Shea elindította a „CSS Zen Garden” oldalt. Ez valószínűleg nagyobb hatást ért el a webfejlesztőknél, mint addig bármi más, ugyanis képes volt azt bemutatni, hogy egy oldal teljes designját meg lehet változtatni egyszerűen a stílus lecserélésével, miközben a tartalom pontosan ugyanaz marad. Azóta a webfejlesztők körében a webes szabványok alapvetővé váltak. A szoftverfejlesztés világában a webalkalmazás egy program, melyet a weben keresztül érünk el az *interneten*, vagy *intranet* hálózaton.

A w*ebalkalmazások* népszerűségének oka, hogy az őket használó webböngésző kliensek szinte minden gépen rendelkezésre állnak. Egy webalkalmazás karbantartható a kliens gépek szoftverének változtatása nélkül. A legelterjedtebb webalkalmazás a Webmail, Webáruház, online aukció, fórum, blog, de ezeken kívül számtalan létezik belőlük.

A webalkalmazások eredete

A korábbi *kliens-szerver* architektúráknál minden egyes alkalmazásnak saját kliens programja volt, ami felhasználói felületként szolgált, és amit minden számítógépre fel kellett külön telepíteni. Egy *szerver-oldali* frissítés általában a *kliens-oldal* szoftverének frissítésével járt, ami kevésbé hatékony megoldás. Ezzel ellentétben a webalkalmazások dinamikusan generálnak szabványos formátumú *Web dokumentumokat (HTML, XHTML)* , amit a webböngészők támogatnak. A kliens-oldali szabványos nyelvű szkriptek, mint a JavaScript a böngészőkben már eleve megtalálható. Minden weboldal statikus dokumentumként érkezik a klienshez, viszont interaktív felhasználói élményt nyújt a beágyazott *Web formokon* keresztül. A kapcsolat közben a webböngésző értelmezi és megjeleníti az oldalakat, és egy univerzális kliensként működik minden webalakalmazás számára.

Felhasználói felület

A Web felület kevés korlátot állít a kliens funkcionalitása felé. *Java, JavaScript, DHTML, Flash* és más technikákkal lehetséges képernyőtartalmat megjeleníteni, hangot lejátszani, egérhez és billentyűzethez hozzáférni. Ezeket kombinálva egy operációs rendszerhez hasonló megjelenést tudunk elérni, melyet a felhasználó már jól ismer. Általános technikákat is támogat, mint pl. a drag and drop. A webfejlesztők gyakran használnak kliens-oldali szkripteket, hogy bővítsék a funkcionalitást, például oldal frissítése, újratöltése nélkül jelenítenek meg tartalmat. Az új technológiák a szerver-oldali nyelvek (pl. PHP) segítségével irányítják a kliens-oldali szkripteket. Ilyen technológia az AJAX, mely számos különböző technikákat alkalmaz a felhasználói élmény javítása érdekében.

Technikai megfontolások

A webalkalmazások jelentős elönye, hogy függetlenek a kliens gép operációs rendszerétől, illetve verziójától. Ahelyett, hogy a kliens programokat minden operációs rendszer számára külön megírnánk, az alkalmazást egyszer kell kifejleszteni, és szinte minden platformon működőképes. Azonban következetlen HTML, CSS, és DOM implementációk és a browser specifikációk különbözősége problémákat jelenthet. Eltérő felhasználói böngészőbeállítások (szkript futtatás tiltása, eltérő betűtípus beállítás, stb.) szintén zavaró az alkalmazások következetes implementációjánál. Egy másik szemlélet az Adobe Flash és Java

kisalkalmazások (Java applet) használata. A legtöbb browser manapság már támogatja ezeket, ezért az ilyen alkalmazások bevezetése is rendkívül egyszerű. Jobb irányítást tesznek lehetővé, megoldják a böngésző beállításainak problémáit. Architekturálisan azonban a hagyományos kliens-szerver alkalmazásokra hasonlítanak, ezért sokan vitatják a helyüket a Web alkalmazások között, ezért inkább „Rich Internet Application” (RIA) alkalmazásoknak hívják öket.

A webalkalmazások felépítése

A Web alkalmazásokat általában három rétegre lehet bontani:

- webböngésző
- középső réteg: egy motor, mely dinamikus web tartalmat használ (pl. PHP, CGI, ASP, JSP)
- adatbázis réteg

A webböngészők kéréseket küldenek a motornak, ami kiszolgálja őket azáltal, hogy lekérdezéseket és módosításokat végez az adatbázisban, majd megjelenítik a felhasználói felületet.

Üzleti felhasználás

A szoftverfejlesztő cégek egy feltörekvő stratégiája, hogy web-hozzáférést biztosítsanak a már létező helyi alkalmazásokhoz. Ez lehetséges oly módon is, hogy egy teljesen más böngésző alapú felületet fejlesztenek, de adaptálhatják a meglévő alkalmazást is másféle megjelenítéssel. Azokat a cégeket, melyek ezt a stratégiát követik, hálózati alkalmazás szolgáltatóként (application service provider, ASP) említünk. Ezek a cégek egyre nagyobb figyelmet kapnak.

Webalkalmazások fejlesztése

Sok különböző webalkalmazás-keretrendszer létezik, amely elősegíti a gyors alkalmazás fejlesztést, lehetővé téve a programozónak, hogy magas szintű leírást adjon a programról. Web alkalmazás keretrendszerek használata egyszerűsítheti a kódot, csökkentheti a hibák számát. A keretrendszerek elősegíthetik a legjobb technikák használatát, pl. GET a POST után. A Web Application Security Consortium (WASC) és OWASP projekteket azzal a céllal fejlesztették és dokumentálták, hogy elkerüljék a Web alkalmazások biztonsági problémáit. A Web Application Security Scanner pedig egy olyan speciális szoftver, ami segít észrevenni a webalkalmazások hibáit, lehetővé téve ezzel hibamentesebb szoftverek elkészítését.

Bevezető a webes szabványokba

Miért használjuk a webes szabványokat?

- **Hatékony kód**
- **Egyszerű karbantartás**
- **Hozzáférés**
- **Kompatibilitás**
- **Webes keresők és keresőmotorok**

A fenti előnyök ellenére a legtöbb weboldal mégsem követi a szabványokat, és rengeteg fejlesztő még ma is régi, elavult módszerekkel dolgozik szerte a világon. Hogy miért? Ennek sok oka van - sokszor az oktatás hiányosságára vagy a cégük házirendjére hivatkoznak, esetleg hogy nincs szükségük megtanulni a webes szabványokat, mert anélkül is megkapják a fizetést, túl nehéz megtanulni a szabványok használatát... Ez eléggé elszomorító.

A HTML dokumentumokról

A HTML formátumú dokumentumok megtekintése egy ún. webböngésző programmal lehetséges. Mi is pontosan a HTML? A HTML egy olyan szövegfájl, amely a szövegen kívül tartalmaz ún. HTML tag-eket - formázóutasításokat -, valamint a megjelenítendő objektumokra történő hivatkozásokat is. Ezek a HTML formázóutasítások befolyásolják a dokumentum megjelenítését. Ezeket az utasításokat a böngészőprogram értelmezi és végrehajtja. Ezen okból a formázóutasítás mindig megelőzi azt a részét a dokumentumnak, amelyre vonatkozik.

A dokumentumkészítéshez használható HTML utasítások köre állandóan bővül, a nyelv fejlődik. A szabványosítás csak lassan követi a fejlődést. Ezért nem minden böngészőprogram tudja a HTML utasítások mindegyikét értelmezni. Egy böngésző, ha számára értelmetlen utasítással találkozik, akkor kihagyja, így nem okoznak problémát az újabb keletű - még szabványosítatlan - utasítások a régebbi kiadású, valamint a kezdetleges WWW-böngészőknek sem.

Sajnos a fentiek miatt ugyanazt a dokumentumot két különböző program nem biztos, hogy azonos formában fogja megjeleníteni. Ennek más oka is van. A WWW-n kalandozónál kicsi a valószínűsége annak, hogy rendelkezésére áll ugyanaz a betűtípus, mint a WWW-oldalt fejlesztőnek vagy képek esetén semmi garancia nincs arra, hogy minden böngészőprogram ugyanazon felbontásban és színszámmal tudja megjeleníteni a képet, stb.
A HTML-ben mégis az a nagyszerű, hogy nagymértékben megközelíti a platformfüggetlenséget. Egy HTML dokumentum - ha nem is teljesen azonos módon - mindenki számára megtekinthető.

Néhány szóban a HTML alapvető szabályairól

A HTML dokumentum hagyományos szövegfájl. Bármely szövegszerkesztővel létrehozható, ill. módosítható, amely nem használ különleges fájlformátumot vagy ha létezik TEXT formátumú mentési lehetőség benne. A HTML utasításokat a szövegben **<** és **>** jelek közé kell zárni. Egy-egy utasítás - HTML parancs, HTML elem - hatását általában a záró utasításpárja szünteti meg, amely megegyezik a nyitó utasítással, csak a / jel vezeti be. Az utasítások nagy része opcionális elemeket is tartalmazhat, melyek csak a nyitóutasításban szerepelhetnek, a záróban nem. Az opciók értékadásánál az idézőjel nem mindig kötelező, csak ajánlott. A HTML utasítás kulcsszavaiban nem különböztetjük meg kisbetűket és nagybetűket.

A HTML dokumentum elvi felépítése

Minden HTML formátumú szövegfájl a **<HTML>** utasítással kezdődik és a **</HTML>** záróutasítással végződik. Ezen elemek közé kell zárni a teljes dokumentumot - formázóutasításokkal és hivatkozásokkal együtt.

A HTML dokumentumot két részre lehet bontani: fejlécre és dokumentumtörzsre. A dokumentumot a fejlécelemek vezetik be, melyek kezdetét a **<HEAD>** utasítás jelzi.

A fejlécelemek között szokás a dokumentumcímet megadni, mely címet a **<TITLE>** és a **</TITLE>** utasítások közé kell zárni. A fejlécet a **</HEAD>** utasítás zárja. Ezt a részét a dokumentumnak általában az ablak címsorában jelenítik meg a böngészőprogramok.

A dokumentumtörzs a fájl **<BODY>** és **</BODY>** utasítások közötti része. Ezen elemek között kell elhelyezni mindent: a szöveget, hivatkozásokat, képeket, stb. (A keretek és a JavaScript kódok kivételével!) Az alábbi egyszerű példa tartalmaz fejlécet, törzsét pedig egy elsőszintű alcím és egy normál bekezdés alkotja:

```
<HTML>
<HEAD>
<TITLE>A dokumentum neve</TITLE>
Fejléc elemek ...
</HEAD>
<BODY>A tulajdonképpeni dokumentumtörzs következik ...
<H1>Ez itt egy alcím</H1>
<P>Ez itt egy normál bekezdés
</BODY>
</HTML>
```

A HTML nyelvet fejlesztői a kezdetekben tartalom egyszerű közlését megoldó leírónyelvnek tervezték. A nyelv megjelenése egyre több ember számára tette lehetővé, hogy publikálhasson a weben, és az internet terjedése is egyre több és több új felhasználót hozott. A felhasználók elkezdtek egyre többre vágyni, az évek során felmerült annak az igénye, hogy lehessen a nyomdai megjelenéshez hasonlóan befolyásolni a weboldalak kinézetét. Így a HTML kiegészült bizonyos formázási lehetőségekkel.

Az új formázási lehetőségek sokmindent lehetővé tettek, de a nyelv ezzel elvesztette az egyszerűségét, a dokumentumok a megjelenés miatt egyre bonyolultabbá és összetettebbé váltak. A Ekkor jött a CSS szabvány, melyet a böngészők az utóbbi években egyre egységesebben értelmeznek, s sokkal szabadabban, rugalmasabban tudjuk vele befolyásolni HTML oldalaink megjelenését, mint azt korábban bármikor is tehettük.

A technológia már viszonylag elég régóta létezik, a CSS szabvány leírása 1996. december 17-n látott napvilágot a W3C oldalán. A szabvány azóta több kiadást ért meg, illetve 1998. május 12-n napvilágot látott a CSS 2 szabvány leírása is (a CSS 3 kidolgozása pedig folyamatban

van). Fontos, hogy a széles körben használt böngészőprogramok viszonylag jól támogatják a CSS szabványt, de könnyen előfordul, hogy valamit félreértenek. A CSS specifikációját a World Wide Web Consortium felügyeli.

Miért CSS?

Egy példán keresztül nézzük meg, miért lesz egyszerűbb a dolgunk a CSS használatával. Vegyünk egy általános oldalt, ahol több címsor és bekezdés található. Szeretnénk, ha a címsoraink betűi nagyok és sötétvörösek lennének, míg a bekezdéseink betűi kisebbek és sötétzöldek. Ezt HTML formázással a következőképpen tudjuk megvalósítani:

```
<html>
<head>
  <title>CSS példa</title>
</head>
<body>
<h1><font size="4" color="#a00000">Bevezetés</font></h1>
<p><font size="2" color="#00a000">A vers...</font></p>
<h1><font size="4" color="#a00000">Tárgyalás</font></h1>
<p><font size="2" color="#00a000">A költő...</font></p>
<h1><font size="4" color="#a00000">Összefoglalás</font></h1>
<p><font size="2" color="#00a000">Végezetül...</font></p>
</body>
</html>
```

Hogyan alakul ugyanez stíluslapok használatával?

```
<html>
<head>
  <title>CSS példa</title>
  <style type="text/css"><!--
  h1 { font-size: 20px; color: #a00000; }
  p { font-size: 12px; color: #00a000; }
  --></style>
</head>
<body>
<h1>Bevezetés</h1>
<p>A vers...</p>
<h1>Tárgyalás</h1>
<p>A költő...</p>
<h1>Összefoglalás</h1>
<p>Végezetül</p>
</body>
</html>
```

A trükk a fejlécben látható `style` elemeken belül van. Meghatározzuk, hogy a HTML állományban levő összes címsor (`h1`) elem és bekezdés (`p`) elem a fentiekben meghatározott

kívánságainknak megfelelően jelenjen meg. Ezt csak egyszer kellett megtenni A pontos méretet is meghatározhattuk, ezúttal pixelben megadva azt, de további mértékegységek is rendelkezésünkre állnak. Amennyiben valamiért utólag módosítanunk kell a megjelenést, egyből csak egy helyen kell ezt megtennünk, de mindenhol megváltozik. Az egyszerűbb szerkeszthetőségen és karbantarthatóságon kívül megvan az az előny, hogy elkészített dokumentum sokkal kisebb méretű is lesz, gyorsabban letöltődik, kisebb adatforgalmat generál. Ha egy külső állományban helyezzük el a stílusdefiníciókat, akkor több dokumentum megjelenését is befolyásolni tudjuk egyszerre, továbbá ha a böngésző gyorsítótárazza a meghatározásokat, az adatforgalom tovább csökken. További előny, hogy az újabb böngészők egyre inkább támogatják a szabványokat, s ha mi is tartjuk hozzá magunkat, akkor egyre kisebb munkával egyre több látogató fogja pontosan ugyanúgy látni a honlapunkat.

CSS hozzákapcsolása a HTML-hez

Beágyazott stíluslap

Ezt láthattuk a fenti példában.

Külső stíluslap

A stíluslapunkat elhelyezhetjük egy külső állományban is, így lehetővé téve, hogy az több HTML állományra is érvényes legyen. A külső stíluslapokra az oldal fejlécében tudunk hivatkozni, egy `link` elem segítségével:

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

A kulso.css tartalma csak ennyi:

```
h1 { font-size: 20px; color: #a00000; }
p { font-size: 12px; color: #00a000; }
```

A beágyazott stíluslapokhoz hasonlóan az így meghatározott stílus az egész dokumentumra érvényes.

Elemhez rendelt stíluslap

Bár kevés alkalommal van rá szükség, de akkor jól jön, hogy az egyes HTML elemekhez helyben is tudunk stílust meghatározni. Egy elem stílusának a meghatározásához egy `style` attribútum szükséges:

```
<h1 style="font-size: 20px; color: #a00000;">Bevezetés</h1>
```

Az így definiált stílus csak az adott elemre, illetve az adott elemen belül lesz érvényes.

Importált stíluslap

További lehetőségünk, hogy egy stílus meghatározáson belül egy másik, külső stílus meghatározásra hivatkozzunk Ez a következőképpen történik:

```
<style type="text/css"><!--
@import url(http://www.honlapunk.hu/stilusok/masik.css);
--></style>
```

A külső stílus hivatkozásnak meg kell előznie minden más definíciót, amennyiben már szerepel előtte más, figyelmen kívül lesz hagyva!

Stílusok formátuma

Egy önálló stílus definíciós állomány, vagy egy beágyazott stílus több meghatározást tartalmazhat. Egy-egy meghatározás két részből áll, egy kiválasztó és egy tulajdonság részből.

A következőképpen épül fel tehát egy stílusdefiníció:

```
kiválasztó { tulajdonság }
kiválasztó { tulajdonság }
kiválasztó { tulajdonság }
```

Az egyes definícióknak, de még a kiválasztónak és a tulajdonságnak sem szükséges új sorban lenniük, gyakorlatilag szabadon ránk van bízva, hogy a fenti tartalmat milyen elrendezésben valósítjuk meg. A következő formátumok mind helyesek:

```
- kiválasztó { tulajdonság } kiválasztó { tulajdonság }
- kiválasztó
  {
-  tulajdonság
  }
```

Öröklődés

Példa:

```
<html>
<head>
<title>Öröklődés</title>
<style type="text/css">
P { color: blue }
</style>
</head>
<body>
<p>Ez a rész kék,<em> de ez is, az öröklődés miatt...</em></p>
</body>
<html>
```

A CSS elemek öröklődése úgy történik, hogy a gyökérelemtől (itt a bekezdés) kezdve, (minden elem, ami a gyökérelemen belül van) a gyökérelemet lezáró tagig formázódik. Az öröklés nem minden attribútum esetében történik meg automatikusan. Azt, hogy melyik attribútumok öröklődnek és melyek nem, a CSS specifikációból tudhatjuk meg.

A `CLASS` szelektor

A `CLASS` szelektor nem teljesen a csoportosításhoz tartozik, de fontos megemlíteni, mert ezzel még rugalmasabban tudunk stíluslapokat rendelni a dokumentumunkhoz. A lényege az, hogy a szelektorokhoz hozzárendelünk egy nevet, amire a dokumentum törzs részében bármikor hivatkozhatunk (`CLASS="szelektorhoz rendelt nev"`). Ez akkor jöhet nagyon jól, ha mondjuk nem minden bekezdést szeretnénk kék színűre változtatni, hanem csak egy részét.

Jelen feladat az, hogy készítsünk 2 bekezdést, az egyikhez rendeljünk hozzá egy `CLASS`-t, ami kék színt eredményez, a másikhoz ne rendeljünk semmit!

```
<html>
<head>
<title>Class szelektor példa</title>
<style type="text/css">
P.kek {color: blue}
</style>
</head>
<body>
<p class="kek">Ez a bekezdés kék színű.</p>
<p>Ez már nem kék színű, mivel nincs hozzárendelve a Class.</p>
</body>
</html>
```

Az ID szelektor

Nem csak a CLASS szelektorral készíhetünk csak bizonyos HTML elemekre deklarált stílusokat, hanem az ID szelektorral is, amit a Class szelektorhoz hasonlóan kell használni. A deklarációs részben a szelektor helyére írjunk egy #-et, azután írjuk le az általunk választott szelektor-nevet, majd deklaráljuk ugyanúgy, ahogy az előzőekben. Az adott HTML tag-ben az ID="megadott_nev" segítégével rendelhetjük hozzá a deklarált szelektort.

Példa:

```
<html>
<head>
<title>Az ID szelektor használata</title>
<style type="text/css">
#azonositonev { color : blue }
</style>
</head>
<body>
<p ID="azonositonev">Ebben a bekezdésben kék lesz a
szöveg..</p>
<p>Ebben a bekezdésben már nem, ugyanis nem rendeltük a taghez
az ID szelektort...</p>
</body>
</html>
```

Az ID szelektort vonatkoztathatjuk csak egy bizonyos HTML elemre úgy, hogy a # elé beírjuk azt a HTML taget, amire vonatkoztatni akarjuk a formázást. Ha ezt az ID azonosítót egy másik (nem az, amit a deklarációnál megadtunk) HTML tagnél rendeljük hozzá, akkor a formázás arra nem fog vonatkozni:

```
<html>
<head>
<style type="text/css">
em#azonositonev { color : blue }
</style>
</head>
<body>
<p ID="azonositonev">Ez a rész fekete, ugyanis nem a bekezdésre
vonatkozatattuk a deklarációnál megadott ID szelektort, de <em
ID="azonosito">ez a rész már kék, ugyanis ezt a tag-et
deklaráltuk...</em>Ez a rész már megint nem kék, mert az "<em>"
tag le lett zárva.</p>
</body>
</html>
```

A CSS segítségével ezenkívül még rengeteg jellemzőt állíthatunk, mely taglalása rengeteg oldalt megtöltene, így most nem mennék bele.

JSP

A **JSP (Java Server Pages)** egy technológia, melynek segítségével a szoftverfejlesztő dinamikusan tud generálni HTML, XML vagy egyéb dokumentumokat a HTTP kérésekre reagálva. A JSP tekinthető a szervlet réteg feletti absztrakciós szintnek. A JSP oldalból java servlet forráskód generálódik. A JSP 2006 májusa óta a J2EE specifikáció része.

Történet

Az első hivatalos JSP specifikációt 1999 júniusában adták ki. A későbbi változatok felülről kompatibilisek ezzel az 1.0 verzióval. Szintén még 1999-ben jelent meg a JSP 1.1 verzió decemberben, amely már lehetőséget adott a *custom tag libraries*, magyarul elemkönyvtárak használatára. A JSP 1.2 verzió 2001 szeptemberében látott napvilágot. Ezt követte a JSP 2.0 verzió, amelyben már szerepelt a *kifejezésnyelv* (angolul *Expression Language* vagy röviden *EL*) és a JSPX dokumentumok támogatása is. A legfrissebb JSP specifikáció a JSP 2.1.

Szintaxis

Egy JSP oldalban a következő nyelvi elemek lehetnek:

- statikus adat, például HTML kód
- direktívák
- szkriptelemek és változók
- akciók
- elemkönyvtárakban definiált *tag*-ek

Direktívák

A direktívák tekinthetők a JSP konténernek szóló utasításoknak is. Attribútumaik is lehetnek. A direktívák általános alakja a következő:

```
<%@ direktívanév attr_1="érték_1" attr_2="érték_2" … %>
```

Három féle direktíva használható JSP oldalakban: **include**, **page** és a **taglib.**

include

Az **include** direktívával be lehet illeszteni egy teljes fájl tartalmát az adott JSP oldalba. Ez hasonlatos a *#include* direktívájához. Statikus tartalmat illeszt be a fordítás előtt. (Dinamikus, futásidejű beillesztés a *jsp:include* akcióval lehetséges.) A beillesztett fájlok kiterjesztése általában *.jspf*, ami a **JSP** Fragment:

```
<%@ include file="valami_file.jspf" %>
```

page

A **page** direktíva hatása a kapott opcióktól függ, melyeket az alábbiakban részletezünk:

-import

Stringértékű argumentum, a generált szervlet kódban import utasítássá konvertálódik:

```
<%@ page import="java.util.*" %>
```

-contentTypes

A tartalom típusát adja meg. Ezt akkor érdemes használni, ha a a generált tartalom nem HTML vagy ha az alapértelmezett karaktertáblától különbözőt kívánunk használni:

```
<%@ page contentType="text/html;charset=ISO-8859-2" %>
```

-errorPage

Azt adja meg, hogy mely oldalra irányítódjon át a felhasználó, ha kivétel történik a HTTP kérés kezelése során:

```
<%@ page errorPage="hibalap.jsp" %>
```

-isErrorPage

Logikai értéket vehet fel. Akkor igaz az értéke, ha az adott lap hibalap:

```
<%@ page isErrorPage=false %>
```

-isThreadSafe

Logikai értéket vehet fel. Értéke akkor legyen *true*, ha az adott oldal szálbiztos, értve ezalatt azt, hogy ki tud szolgálni több kérést párhuzamosan. Ha értéke *false*, akkor a JSP container sorbaállítja a kéréseket: **<%@ page isThreadSafe=true %>**

-info

String értékű argumentum, mely az oldal rövid leírását tartalmazhatja. Az így definiált leírást adja vissza a generált Servlet osztály `getServletInfo()` metódusa:

```
<%@ page info="Ez itt, kéremszépen, fontos servlet információ." %>
```

-extends

Azt adja meg, hogy a JSP oldalból generált osztály mely osztálynak legyen a leszármazottja. Ritkán használják, hiszen általában megfelel az alapértelmezett öröklődés.

```
<%@ page extends="ValamiOsztaly" %>
```

-session

logikai értékével azt határozható meg, hogy akar-e a szoftverfejlesztő sessiont használni. Az alapértelmezett érték `true`. Ha értéke `false`, akkor nem használható a `session` implicit objektum.

```
<%@ page session=false %>
```

-buffer

Megadható a kimeneti puffer minimális mérete. Az alapértelmezett érték 8kb. Értéke lehet `none` is, ekkor nincs pufferelés, és minden közvetlenül íródik a HTTP válasz `PrintWriter` objektumára:

```
<%@ page buffer=32kb %>
```

```
<%@ page buffer=none %>
```

-autoFlush

Logikai érték, mely azt határozza meg, hogy mi történjek, ha a kimeneti puffer megtelik. Ha értéke `true`, akkor a puffer automatikusan ürítődik. Ha értéke `false`, akkor kivétel váltódik a puffer megtelése esetén. Alapértelmezett értéke `true`:

```
<%@ page autoFlush=false %>
```

taglib

A **taglib** direktívával JSP elemkönyvtárakat lehet használni. Paraméterként meg kell adni egy URI-t, ahol az elemkönyvtár leírófájlja található, valamint egy prefixumot, amivel később hivatkozni lehet az elemkönyvtár elemeire. Például így:

```
<%@ taglib prefix="kedvencprefixumom" uri="taglib/kedvenc.tld" %>
```

Szkriptelemek

Háromféleképpen lehet java forráskódot elhelyezni a JSP oldalban: deklarációk, szkriptrészletek és kifejezések formájában:

Deklarációk

A *deklarációs tag*-ben definiált kód teljes egészében bemásolódik a generált java servlet osztály forráskódjába. Például adattagokat lehet vele definiálni:

```
<%! int serverInstanceVariable = 1; %>
```

A deklarációs tag metódust is tartalmazhat, amely szintén a generált szervlet osztályba kerül be:

```
<%!
 public void increaseServerInstanceVariable() {
     serverInstanceVariable++;
 }
%>
```

Szkriptrészletek

A *szkriprészlet* (angolul *scriptlet*) tagben megadott kódrészlet a generált servlet osztály `_jspService()` metódusába másolódik be.

Szintaxisa:

```
<% java forráskód %>
```

Példa:

```
<% int lokalisValtozo = 6 * 7 ; out.println("Hatszor hét az " +
lokalisValtozo + "."); %>
```

Kifejezések

A *kifejezés* tag-ben megadott kifejezés futási időben értékelődik ki és az értéke kerül a webszerver által visszaadott HTML kódba. Az alábbi példa $\sqrt{2}$ közelítő értékét generálja:

```
<%= Math.sqrt(2) %>
```

Implicit objektumok

Léteznek ún. implicit objektumok, melyeket a JSP container láthatóvá tesz a JSP oldalak számára. Ezekre hivatkozhat a szoftverfejlesztő a JSP kódban:

-out

A JSPWriter osztály példánya, a HTTP válasz üzenettörzs részét reprezentálja.

-page

Maga a generált java servlet objektum.

-pageContext

A `javax.servlet.jsp.PageContext` osztály példánya. Az egész oldalra vonatkozó adatokat tartalmaz.

-request

Egy `javax.servlet.http.HttpServletRequest` objektum, amely a HTTP kérést reprezentálja.

-response

A HTTP válasz objektum.

-session

A HTTP session objektum, mely felhasználói információk megőrzésére használható.

-config

A servlet konfigurációs adatait tartalmazó `ServletConfig` objektum. Értéke megegyezik a servlet objektum `getServletConfig()` metódusának visszatérési értékével.

-application

Alkalmazásadatokat tartalmazó objektum.

-exception

A dobott kivétel objektumot tartalmazza, amennyiben az adott JSP oldal hibalap, azaz `isErrorPage="true"`.

Megjegyzések

A *megjegyzés* tag-ek formátuma az alábbi:

```
<%-- Megjegyzés szövege --%>
```

Akciók

A JSP tartalmaz standard akciókat is, de a JSP 1.1 verziója óta lehetséges a saját akciók definiálása is saját elemkönyvtárak használatával. Az alábbi lista mutatja a JSP standard akcióit:

-jsp:include

Hasonlatos a szubrutinhíváshoz. Hatására a servlet átadja a HTTP kérés kezelését egy másik oldalnak. Ha a megadott oldal válaszolt, akkor visszakerül a vezérlés a hívó oldalhoz. Ezzel az akcióval az egyszer megírt kód újrahasználható, s így nem kell azt duplikálni. Az a különbség az include direktíva és a jsp:include akció között, hogy míg a direktíva statikusan illeszti be egy másik oldal tartalmát fordítás előtt, addig a jsp:include akció dinamikusan, a HTTP kérés kezelésekor fut le és így dinamikusan generált tartalmat is be tud illeszteni.

-jsp:param

A jsp:include, jsp:forward vagy jsp:params blokkok belsejében használható. A megadott paraméter hozzáadódik a kérés paraméterlistájába

-jsp:forward

Segítségével a HTTP kérés továbbadódik egy másik URL-re. Ellentétben a jsp:include akcióval, itt a hívó oldalhoz soha nem kerül vissza a vezérlés.

-jsp:plugin

A Netscape Navigator és az Internet Explorer korábbi verziói különböző html tag-eket használtak a java appletek beillesztésére. Ez az akció legenerálja a megfelelő böngészőspecifikus *tag*-et.

-jsp:fallback

A megjelenítendő tartalom abban az esetben, ha a böngésző nem támogatja az appleteket.

-jsp:getProperty

Lekérdez egy tulajdonságot a megadott JavaBean-től.

-jsp:setProperty

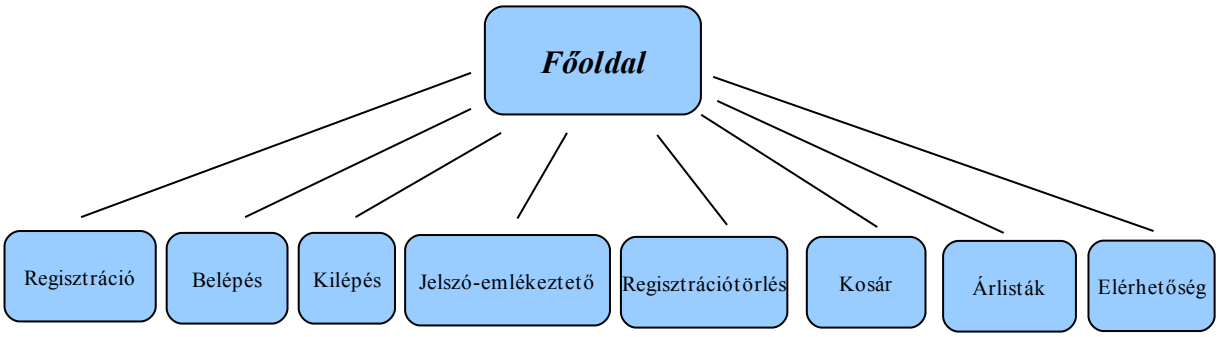
Beállítja a megadott JavaBean egy tulajdonságát.

-jsp:useBean

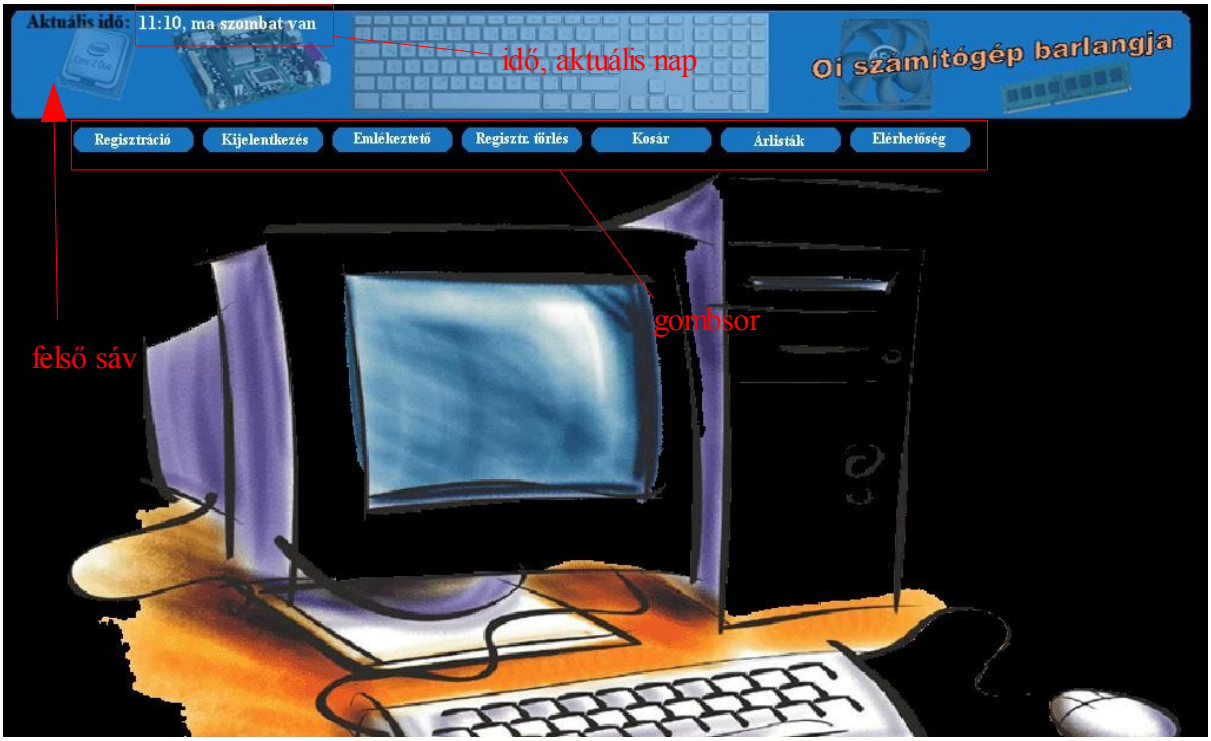
Létrehoz illetve újrafelhasznál egy JavaBeant.

Technológiák a gyakorlatban

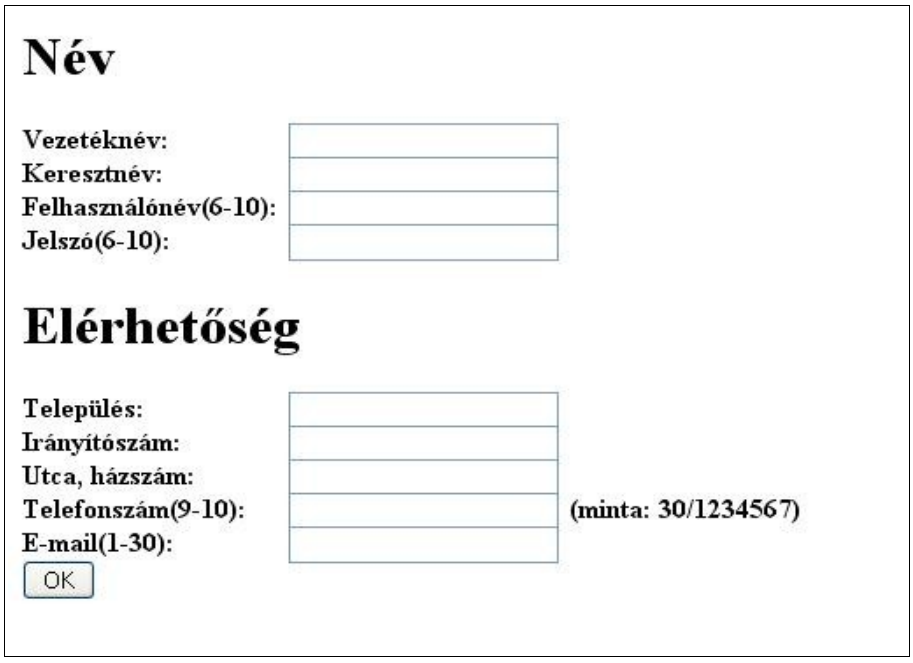
Az előzőekben említett technolóiák illusztrálására egy webshopot készítettem. Ezenkívül MySQL-t is felhasználtam. Ez gyakorlatilag egy fiktív számítógépes bolt, ahol interneten kersztül vásárolunk. Lássuk a szerkezetét!



- **főoldal**: Ezt látja a felhasználó a weblap megnyitásakor. HTML oldalként készült el, a már tárgyalt CSS stíluslap használatával. Külön stílusdefiníciókat hoztam létra a gomboknak, a háttérnek valamint a felső képnek és az időnek. A felső sáv a későbbiekben felhasználható hírek, stb. közlésére.



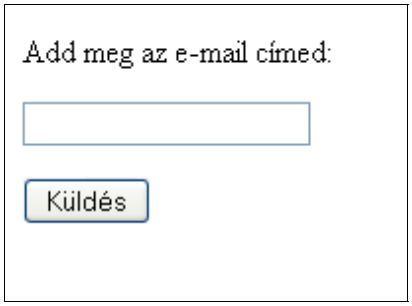
- **regisztráció**: Ez egy regisztrációs űrlap a nem regisztrált felhasználók számára, hogy használhassák a rendszert. Ez egy szabványos HTML oldal, stíluslapok és egyéb szabványok használata nélkül.



A különböző mezők neve maga a mezők előtt található, elnevezésük egyértelmű. A szükséges mezőknél a rendszer ellenőrzést végez a formátum helyessége érdekében. Ilyen pl. hogy a felhasználónévnek és a jelszónak 6 és 10 karakter közé kell esnie, egyébként a rendszer nem regisztrál, hanem felkínálja a lehetőséget az újraszerkesztésre. A rendszer már létező felhasználónév esetén sem regisztrál. Ezt a felhasználókat tároló MySQL adatbázisból állapítja meg az OK gombra kattintás után, amely JSP parancsokat hív meg.

- **belépés**: Bejelentkező képernyő a már regisztrált felhasználók számára. Egy felhasználónevet és egy jelszót kell megadnunk és ezt a rendszer JSecurityCheck segítségével ill. JDBC hívásokkal (felhasználók adatbázisa) ellenőrzi. Ez az oldal csak akkor jelenik meg, ha már pakoltunk termékeket a virtuális kosárba, és szeretnénk azok megrendelését véglegesíteni. Ez azt is jelenti, hogy a kosárba a nem regisztrált ill. nem bejelentkezett felhasználók is tudnak, de a a megrendelés már nem megy ilyen módon. Megrendelés után a rendszer egy-egy értesítő e-mailt küld a felhasználónak és az adminisztrátornak. Az értesítő küldésekor a JavaMail API-t használtam fel.

- **kilépés**: Kijelentkezés a már bejelentkezett felhasználóknak. Ez is ugyanúgy JSP oldal, mint az előző. Ez csupán annyit csinál, hogy zárja az aktuális sessiont.
- **jelszó-emlékeztető**: Elfelejtett jelszó esetén nyújt segítséget, egy e-mailt küld a felhasználó e-mail címére a felhasználónévvel és a jelszóval.



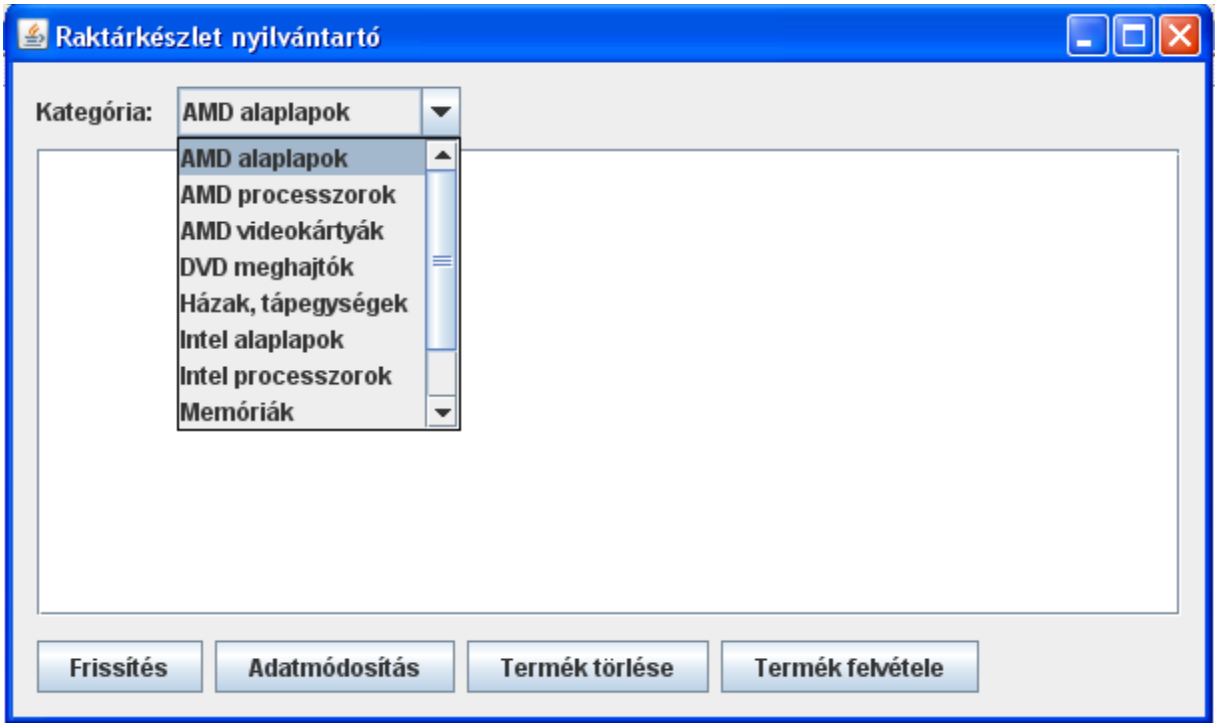
Azt az e-mail címet kell megadnunk, amellyel regisztráltunk, azonban ha az adatbázisban nem szerepel, a rendszer hibajelzést ad.

- **regisztráció törlés**: Töröltethetjük magunkat a rendszer adatbázisából, azonban biztonsági megfontolásokból felhasználónevet és jelszót is kér, hogy illetéktelenek ne töröltethessenek minket.
- **kosár**: A már említett virtuális kosár, az összeválogatott, kifizetésre illetve megrendelésre váró termékek. Minden termék mellett található egy gomb, mely segítségével sztornózhatjuk az adott tételt, így az nem kerül bele a végleges megrendelésbe. A rendszer a tételek alatt megjeleníti a végösszeget, majd a Megrendelés gombra kattintva véglegesíthetjük szándékunkat.
- **árlisták**: A megvásárolható termékek listája kategóriákra bontva. Ezek az árlisták egy számítógép működéséhez szükséges főbb alkatrészeket tartalmazzák az árral együtt. Minden termék mellett van egy „Kosárba” nevű gomb, mely segítségével a már tárgyalt virtuális kosárba rakhatjuk az adott tételt, jelezve a megrendelési szándékunkat. A rendszer a raktáron nem szereplő termékeket nem jeleníti meg, így a felhasználó biztos lehet abban, hogy hozzájuthat a termékhez.
- **elérhetőség**: A bolt elérhetőségeit mutatja (nyilvánvalóan fiktív elérhetőségeket tartalmaz).

Az üzlet címe: 1238 Budapest, XXXIII. Kossuth u. 65.
E-mail: info@szgbarlang.hu
Telefonszám: 06-80-370-8543

Vissza a főoldalra

Ahogyan az ábrán is látható, minden oldal aljára a könnyebb kezelhetőség kedvéért egy „Vissza a főoldalra linket” tettem, a navigáció megkönnyítése érdekében. A raktárkészlet kezeléséhez egy egyszerű Java alkalmazást készítettem, amely tulajdonképpen csak annyit csinál, hogy a megfelelő műveletekhez legenerálja a MySQL parancsokat.

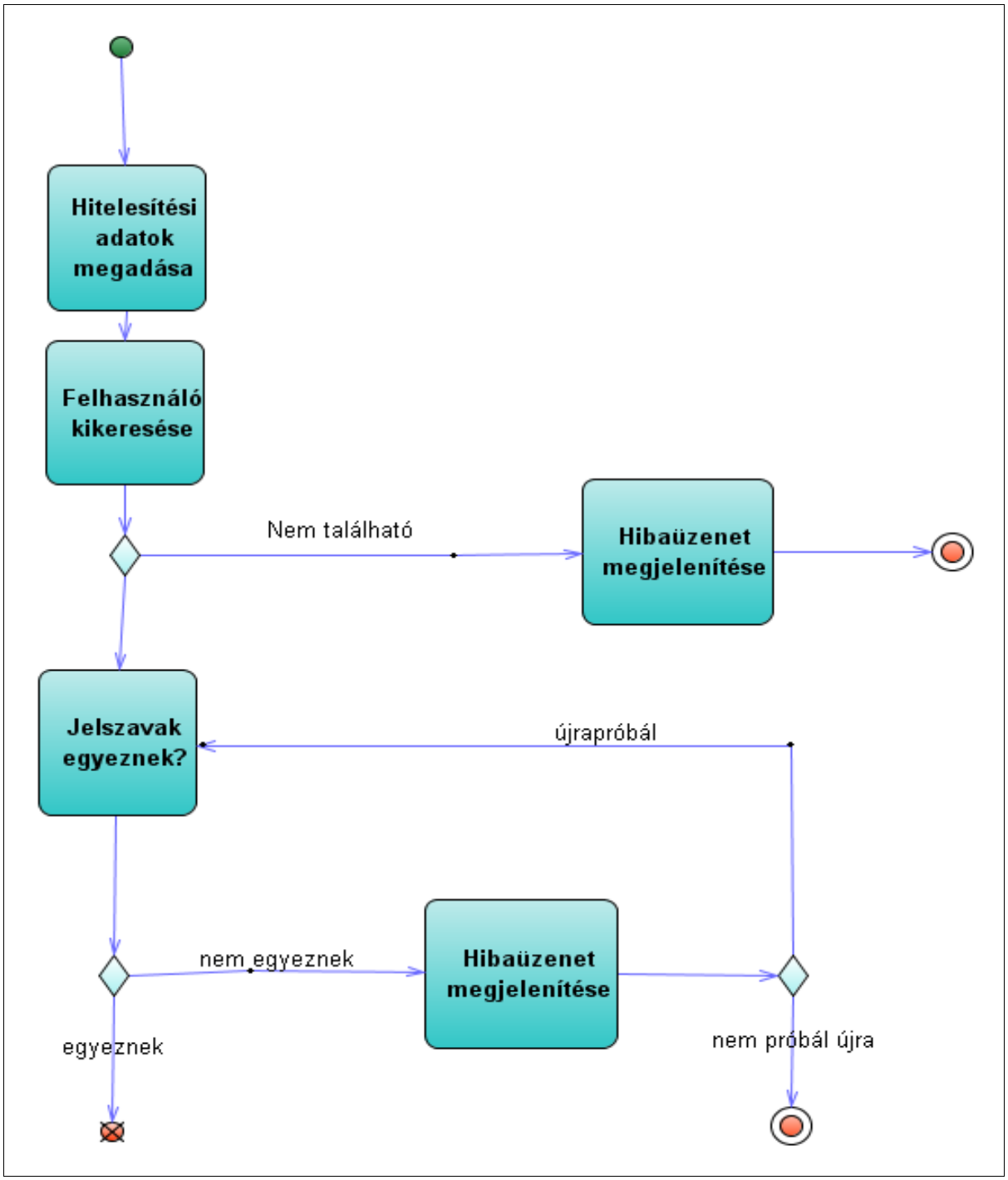


A Kategóriáknál láthatjuk a különböző alkatrész típusokat. Valamelyiket kiválasztva, majd a frissítés gombra kattintva megjelennek az oda tartozó termékek. Az adatmódosítás gombra kattintva módosíthatjuk a kijelölt termék árát, kategóriáját megváltoztathatjuk valamint módosíthatjuk a darabszámot ha például érkezett belőle. A „Termék törlése” gomb segítségével törölhetjük a a rendszerből ha esetleg megszűnne a forgalmazása, a „Termék felvétele” gomb pedig új termékek megjelenése esetén lehet hasznos.

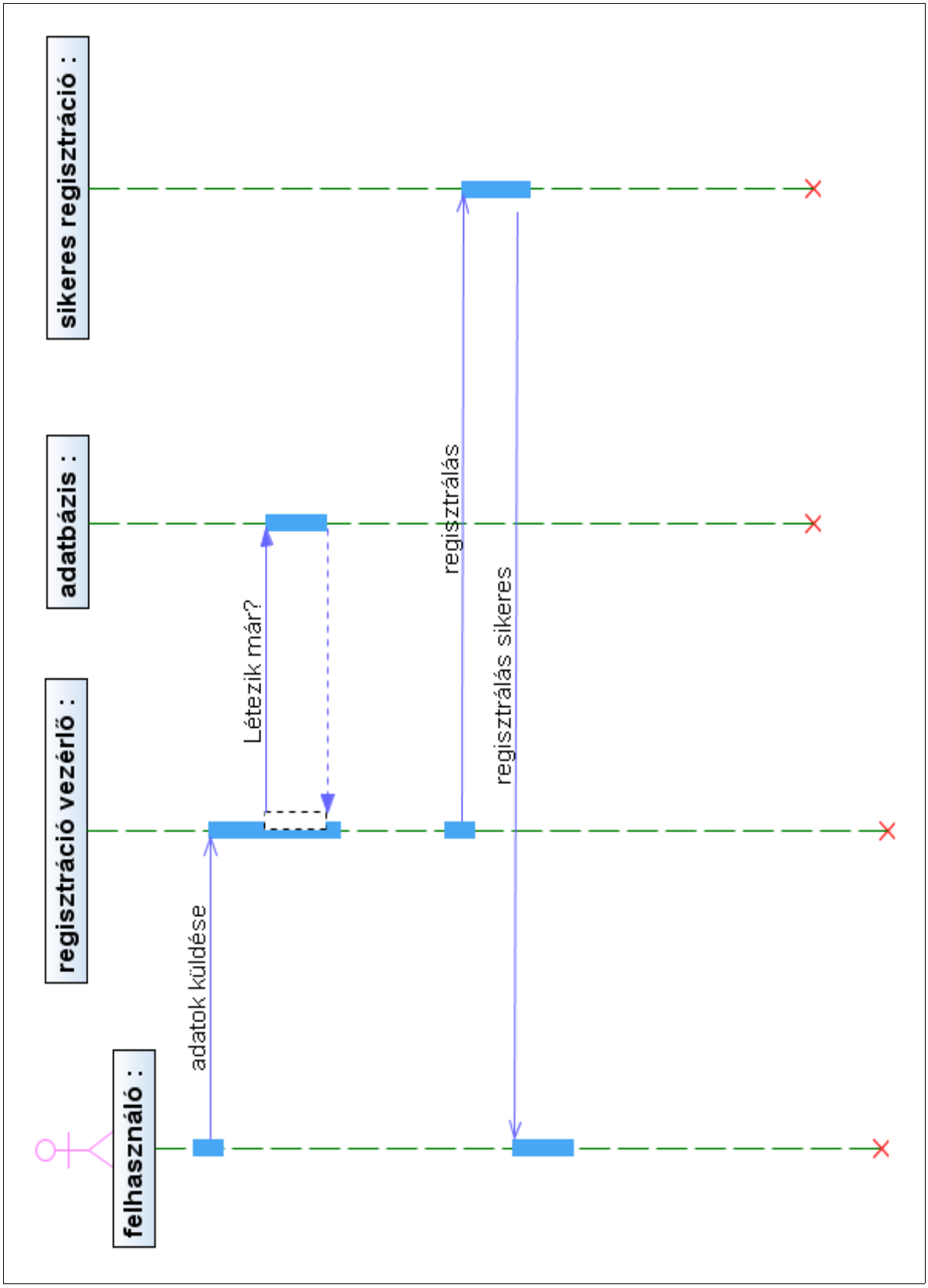
Most a könnyebb átláthatósághoz lássuk a fontosabb folyamatok diagramjait!

A fontosabb UML diagramok

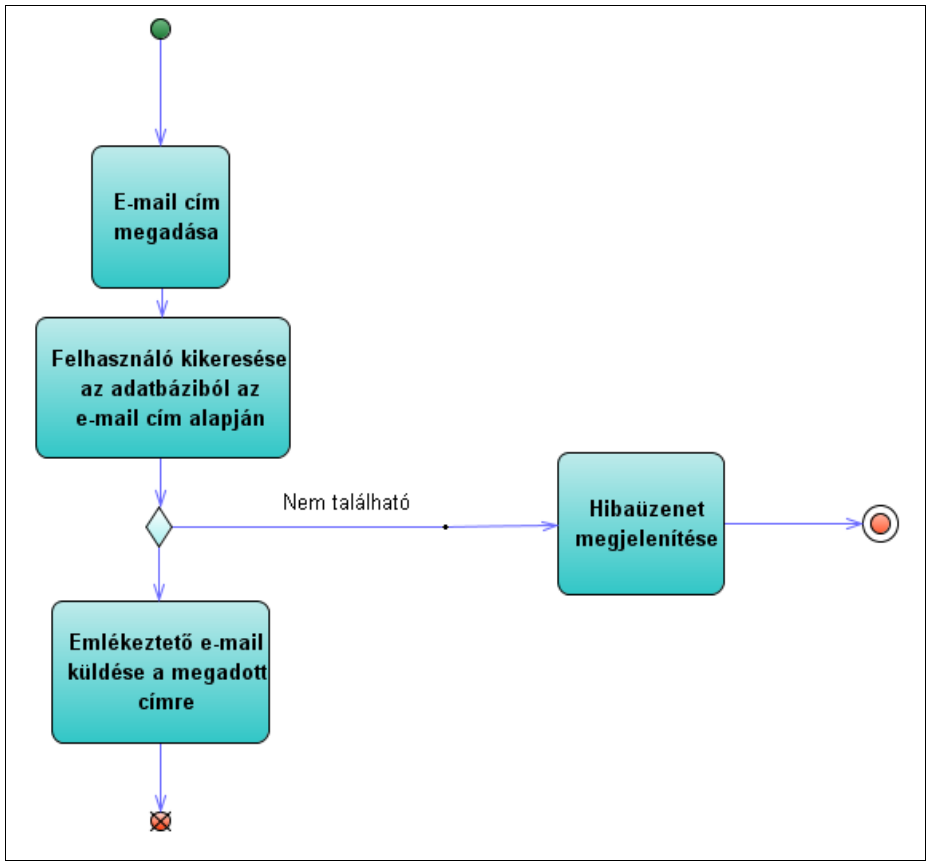
Bejelentkezés:



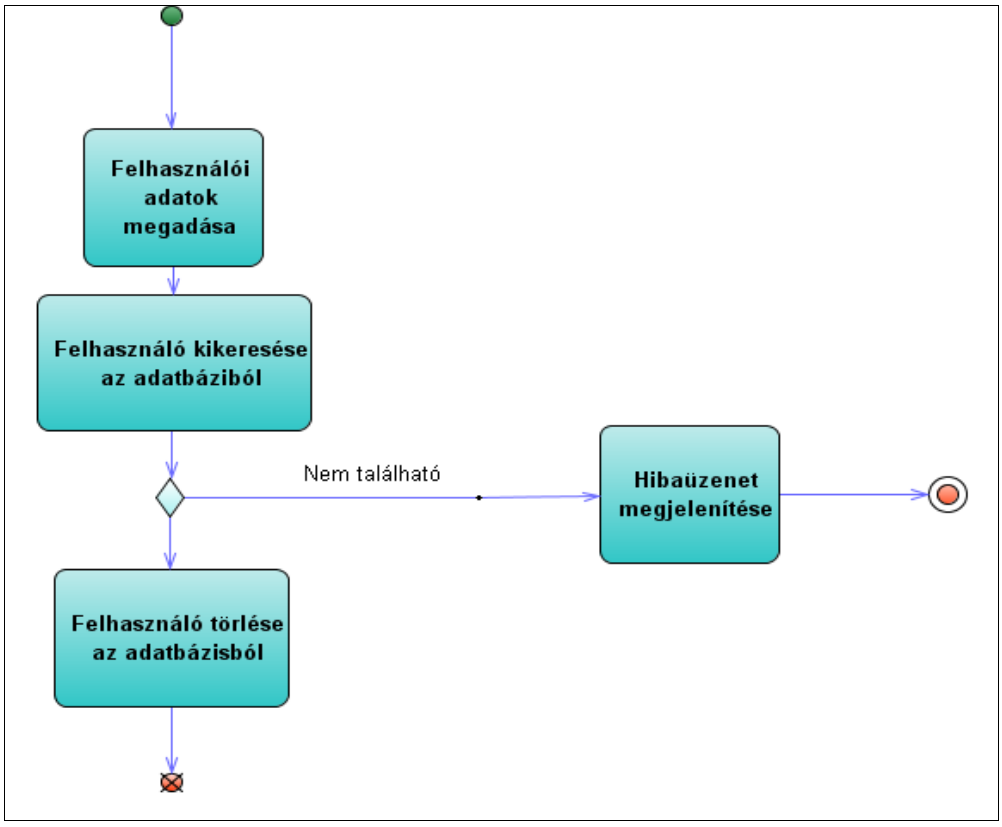
Regisztráció:



Jelszó-emlékeztető:



Regisztráció törlés:



Összefoglalás

Miről is olvashattunk ebben a munkában? Megismerkedhettünk az internet történetével a kezdetektől fogva. Szó volt a cégek háborújáról, a szabványosításra történő igényről. Megismerhettünk szabványokat, közöttük a legismertebbeket, a teljesség igénye nélkül. Láthattunk a használatukra példákat és a gyakorlati feladatomat, megmutatva a technológiák ötvözésének megvalósíthatóságát.

Irodalomjegyzék

- Kris Hadlock: Webalkalmazások fejlesztése Ajax segítségével, *2007* , Kiskapu Kiadó
- Kovács Rudolf, Bártfai Barnabás: Eblapkészítés házilag, 2001, BBS-Info kiadó
- Sikos László: Stíluslapok a weben – CSS kézikönyv, 2005, BBS-Info kiadó
- Debolt, Virginia: HTML és CSS, 2005, Kiskapu kiadó
- László József: Internet a világhálózat, 2007, ComputerBooks kiadó
- Milner, Annalisa: A WEB böngészése, Alexandra kiadó

Köszönetnyilvánítás

Köszönetet mondanék családomnak, barátaimnak a lelki támaszért és hallgatótársaimnak ez elméleti segítségért, hiszen nélkülük ez a munka nem készülhetett volna el.