Debreceni Egyetem

Informatikai kar

Web 2.0 alkalmazásfejlesztés

Diplomamunka

Témavezető:

Adamkó Attila

Egyetemi tanársegéd

Készítette:

Borsós Gyula Attila

programtervező matematikus

Debrecen

2007.

Bevezetés

Manapság, az internet elterjedésével egyre több ember kerül kapcsolatba a világhálóval. Már nemcsak az életünk egy plusz tényezőjeként kell tekintenünk rá, hanem életünk részeként, ezt jól példázza, hogy a Time magazin minden évben kiadott „év embere” díját 2006-ban az „internetező embert” kapta meg. Változik az internetes szolgáltatások célközössége, már nem egy réteget kell, hogy kiszolgáljanak.

A megnövekedett információ halmaz és szolgáltatásigény viszont szükségessé tette a már megszokott „szolgáltató-igénylő” architektúra változását is. Ki kell használnunk azt az erőt, hogy milliók használják az internetet és be kell vonnunk őket a tartalmak bővítésébe, hiszen mennyivel gazdaságosabb, ha nem egyvalakinek kell létrehoznia valamit, hanem mindenki hozzáad egy kicsit.

Szükséges tehát az internet fejlődése, melynek különlegessége abban áll, hogy nem egy kontrollált entitás, nincs semmilyen felügyelő szervezet, amely teljességgel megszabhatná a fejlődés irányát és ütemét. Vannak persze irányelvek, de ezek elfogadása az internetes közösségen múlik.

A jelenlegi irányelv, a dolgozatom témája, Web 2.0, melyet úgy tűnik mind a felhasználók, mind a fejlesztők egyre inkább elfogadnak és próbálnak e fejlődési irány szerint haladni. Dolgozatomban első részében szeretném ismertetni a Web 2.0 fogalmát, legfőbb jellemzőit, előnyeit, hátrányait, az emberek viszonyát az új szemléletű világhálóhoz. Elemzek néhány népszerű Web 2.0-s alkalmazást, majd a kapcsolódó technológiákról is szót ejtek.

A második részben a Web 2.0-s alkalmazások legfőbb implementációs módszerét, az Ajax-ot ismeretem, kitérve az alapvető implementálási technikára, legfőbb programozási sajátosságokra és mintákra.

Végül az alapoktól kezdve ismertetem egy Web 2.0-s webalkalmazás megtervezését, implementálást. Maga a program témájának kitalálásakor fontos volt, hogy illeszkedjen a Web 2.0 által meghatározott irányelvekbe. Választásom egy zenei tudástárra esett, melyben a felhasználók, egy közösség részeként, az egymás által bevitt és kategorizált adatok között kereshetnek. Legvégül megvizsgálom, hogy az elkészített program mennyiben felel meg az alapelveknek. A program ismertetésénél feltételezem, hogy az olvasó ismeri a Java, JSP, SQL, HTML, XML és JavaScript technológiákat, így csak azokra az implementációs technikákra térek ki, melyek az alkalmazás szempontjából sajátosak.

A Web 2.0 jelenség

A „web 2.0” fogalom még ma is rengeteg vita tárgya. Valakik szerint nem lehet definiálni, mások szerint nem is létezik, sokak pedig egyáltalán nem tudják mit is takar.
Maga a „web 2.0” kifejezés az O’Reilly Media nevű médiatársasághoz kötődik, közismert pedig az általuk 2004-ben tartott „Web 2.0 konferencia” után lett. Itt olyanok vettek részt, mint Jerry Yang a Yahoo-tól, Louis Monier, az AltaVista kereső-motor alapítója, Halsey Minor a CNET-től, John McKinley az American Online-tól, Lawrence Lessig, akinek a Creative Commons-t köszönhetjük, Cory Doctorow író, Marc Andreessen, a Mosaic böngésző egyik alkotója (melyet a Microsoft miután megvásárolt, kisebb-nagyobb módosítások után átnevezett Internet Explorer-re), Mary Meeker a Morgan Stanley-től, Jeff Bezos az Amazon-tól és persze Tim O'Reilly.
A fogalom jelentése időről időre változik, lassacskán letisztázódni látszik végleges jelentése. Abban mindenki egyetért (még a web 2.0-t fogalmat ellenzők is, sőt. nekik ez a legfőbb érvük), hogy a 2.0-s „verziószám” nem azért került a szó mögé, mert új (verziójú) világhálóról van szó. Igazán új technikai vagy technológiai újítás nem következett be (az Ajax komponensei is megvoltak már jó ideje). Az viszont látszik, hogy az emberek viszonya a világhálóhoz megváltozott. Azt megmondani, hogy az új „web2-es alkalmazások” létrejötte miatt változott meg az emberek viszonya, vagy az emberek új hozzáállása szülte a web2-es alkalmazásokat, talán lehetetlen. Maguk az weboldalak megváltoztak, szebbek, felhasználó-barátiabbak lettek, mégsem jelenthetjük ki, hogy emiatt kezdtek el az emberek blog-ot írni, online enciklopédiát használni és közösségi oldalakon régi ismerősöket keresni.
Megfogalmazhatjuk a változást tehát úgy, hogy már nem egy bizonyos réteg szerkeszti a web-et (a weboldal készítők, akik értenek a HTML-hez, vagy „bonyolult” szerver oldali technológiákhoz), hanem egy jóval bővebb halmaz, amiben azok az emberek is helyet kapnak akik nem értenek az informatikához, a programozáshoz, de szerkesztik az enciklopédiákat, videókat töltenek fel, részét képzik zenei statisztikáknak, weboldalakat készítenek (anélkül hogy akár a HTML-t is ismernék).

A Web 2.0 fogalma

Tim O'Reilly és John Bettelle a következőkben fogalmazta meg a Web 2.0 alapelveit:

- ***a világháló platformként való működése, használata***

  O'Reilly szinteket állapított meg az alkalmazások „web2-ségének" megadására:

  0. szint: Az alkamazás jól működik offline is, ha az összes adat rendelkezésre áll. Ilyen a GoogleMaps, vagy akár egy szótár.

  1. szint: Bár megfelelően működik offline is, de plusz funkciókat kap online működés által. Ilyen lehet egy szövegszerkesztő, vagy napló, ami online is működik.

  2. szint: Olyan alkalmazások melyik működnek offline, de a fő funkciójukat elvesztik ez által. Itt a Flickr képmegosztót említi példaként, melynek fő funkciója fotómenedzselés, de a megosztással új aspektust kap.

  3. szint: Az igazi Web 2-es szint, ezek az alkalmazások csak az interneten működnek. Ezekre az oldalakra jellemző, hogy fő funkciójuk a közösség által létrehozott információ valamilyen szempont alapján történő rendszerezése. Példák: Wikipedia, iWiW, EBay, Last.fm
- ***az adatkihasználás, mint húzóerő***
- ***„hálózat effektus", azaz mindenki jól jár egy új tag/elem belépésével a hálózatba***

  Ez az a tényező ami nagyon sok Web 2.0-s alkalmazásra (pl. az Amazon.com-ra) nem teljesen igaz. A hálózat effektus legjobb példája a telefon, de maga az internet is. Amíg csak két embernek van telefonja addig csak egymást tudják felhívni. De ha belép egy újabb ember a hálózatba, akkor ezzel ők is nyernek, mivel már két embert tudnak felhívni, és így tovább. Last.fm-es példával élve, ha bekerül egy A,B,C zenekarhoz hasonló D zenekar, akkor ezzel A,B,C is jól járnak, mivel a D „hasonló zenekarok" részlegében megjelenik az ő nevük is. Google AdSense-es példa: minél több reklámozó látja, hogy milyen sok kis oldalon/blogon/stb vannak AdSense-es reklámok, annál több reklámozó fog előfizetni a szolgáltatásra, így egyre jobban megéri AdSense-es reklámot weboldalakra kitenni, így még több reklámozó látja, és így tovább.

- ***több rendszer tulajdonságait összefésülve kialakuló új rendszerek***

  Egy alkalmazásba más webalkalmazásokat is beépítünk, avagy más webalkamazásokból nyerünk ki adatokat. Például kapcsolatépítő portálon mindenkinél jelen van egy Google Maps térkép, jelölve rajta a lakhelyét.

- ***„perpetual beta" szerű szoftverek***

  A perpetual beta jelentése, hogy a szoftver sohasem lép ki a beta fázisból. Így a fejlesztők folyamatosan tehetik az új feature-öket az alkalmazásba, akár rendes kitesztelés nélkül. Web2-es jelentősége, hogy az új feature-öket rögtön tesztelik az alkalmazást használó felhasználók, így derítve fel annak hibáit.

A Web 2.0 a gyakorlatban

A **gyakorlatban** általában a következőket mondhatjuk el egy Web 2.0-s alkalmazásról:

- az előbb már említett **„internet, mint platform”** tényező
  az alkalmazás teljes egészében az interneten működik, bárhol elérhetjük ahol van web-böngésző
- **közösségi szellem**
  Gyakran mondják a Web 2.0-s oldalakra, hogy az oldalt nem az implementációja, szerkezete teszi azzá ami, hanem a felhasználók akik a tagjai. A felhasználók sem azt érzik hogy egy programot használnak, hanem azt, hogy egy közösség, egy társaság részei. Gyakran mindenkinek megvannak az oldalhoz köthető „barátai”, akiket itt ismertek meg, és itt folytatják le velük a kommunikációt (privát üzenetek által)
- **címke alapú keresés**
  Nem véletlen, hogy egyre több oldal használ címkézést (ún. tag-eket). Ez azt jelenti, hogyha egy entitás felkerül az oldalra, például egy videó a Youtube-ra, akkor megfelelő címkékkel kell ellátni. Ez többrétegű rendszerezést tesz lehetővé, és jóval hatékonyabban lehet keresni a címkékkel ellátott videók között.
- **felhasználói felület:** felhasználóbarát, könnyen kezelhető (általában minimalista).
  Ez az egyik olyan tényező amelyikben leginkább különböznek a Web 2.0-snek titulált alkalmazások. Általában cél az egyszerű, áttekinthető kinézet, jó példa erre a Wikipedia, vagy a Google technológiái. Mások viszont végletekig csicsásak, ilyen a Myspace. Technológiailag kétfelé lehet bontani, DHTML alapú, avagy Flash alapú. Irányelvnek mondható a DHTML alapú oldalaknál az Ajax technológia használata. A két technológia meglehetősen különböző.
- az **adatokat a felhasználók alakítják ki** (esetekben ők „birtokolják” őket)
  Az adatokat vagy a felhasználók viszik be a rendszerbe (pl. wiki), vagy az ő cselekedeteik által, automatikusan kerülnek kiértékelésre, majd válnak rendszer részévé (pl. last.fm) Gyakran beszélhetünk saját web-es adatokról.
- **mashup**-ok használata: Egy vagy több web-es forrás integrálása saját alkalmazásunkba. Mashup-ok tervezésére újabban már külön editorok állnak rendelkezésünkre, pl. az MS Popfly, vagy a Google Mashup Editor, Yahoo Pipes. Tulajdonképpen minden mashup-nak tekinthető, ami valami külső forrást igénybe véve ad eredményt. Ez lehet akár egy egyszerű szöveg lekérdezése, egy külső RSS formázott megjelentetése, de akár egy

konkrét, egész szolgáltatás használata (pl. Google Suggest, vagy Google Maps) is. Ezt az aggregációt nem feltétlenül a szervernek kell végrehajtani, a kliensböngészőben is történhet.

Mi is számít tehát Web 2.0-s alkalmazásnak?

Hogy mi is számít tehát Web 2.0-s alkalmazásnak? Mivel a definíció nemhogy nem pontos, de még csak nem is közelítő, még technológiai felsorolást sem tartalmaz, így sajnos azt mondhatjuk, hogy Web2-es az, amit annak hívnak, vagy amit annak tartanak. Inkább egy halmazról beszélhetünk, amiben beledobáljuk a technológiákat, jellemzőket, és ha ebből valaki kivesz néhányat, már Web2-esnek titulálja/titulálhatja programját. Ezen persze nincs csodálkozni való, hiszen itt tényleg inkább a web második generációról van szó, ami nem egzakt.

Tekintsük meg egyes oldalak ezen halmaz mely részeit implementálják és melyeket nem. Az alábbi szempontokat vesszük figyelembe:

- felhasználói felület
- közösségi szellem
- mennyire használja platformként a web-et
- adat és felhasználó kapcsolata

Wikipedia: Online enciklopédia lévén az adatokat a felhasználók alakítják ki, de nem birtokolják őket. (Sokan ezért nem szeretik szerkeszteni, mivel bárki írhat „jobb cikket” és lecserélheti, vagy módosíthatja a régit.) Mivel az elosztott információlétrehozáson alapul, ezért csak a Web-en van értelme használni. A felület tekintetében (bár teljesen felhasználóbarát és egyszerű) nem használja az Ajax technológiát (pontosabban csak minimális, felhasználó által észrevehetetlen szinten), ami érdekes, figyelembe véve hogy az egyik legtipikusabb Web 2.0-s példa.

GoogleMaps, GoogleEarth: A Google saját fejlesztésű AjaxSLT keretrendszerében fejleszett webalkalmazás szép, egyszerű, Ajax-os felülettel rendelkezik, melyen a földet domborzati vagy közigazgatási térképét szemlélhetjük. Az adatokat nem a felhasználók alakítják, és nem értelmezhető közösségi szellem. Web nélkül is működtethető (az ugyanilyen nevű) asztali alkalmazás segítségével. A GoogleMaps leginkább (önálló funkcionalitása mellett) egy keretnek/kiegészítőnek tekinthető, amit más alkalmazások építhetnek be.

GoogleDocs, Gmail: Online irodai alkalmazás. Felületét tekintve a Web 2.0 zászlóshajójának tekinthető. Tiszta, áttekinthető, gyors Ajax-os felület. A GoogleDocs-al már majdnem eljutunk a „webes operációs rendszerekig", tökéletes példája a web-en működő, de asztali alkalmazás jellemzőivel bíró programnak. A Gmail egyik közkedvelt szolgáltatása a Gtalk felveszi a versenyt a többi asztali „azonnali üzenetküldő alkalmazással". Közösségi szellem szintén nem értelmezhető, a felhasználó és adat kapcsolat pedig szintén speciális, mivel ezek privát, de bárhonnan elérhető dokumentumok.

Myspace: Közösségi site, ahol mindenkinek saját, önálló, szerkeszthető profilja van. A legellentmondásosabb Web 2.0-s alkalmazás, sokan nem tekintik annak. Pedig az egyik legfontosabb eszköze (még ma is) annak, hogy az embereket közelebb hozza világhálóhoz, így nem csoda hogy a *wisdump* internetes honlap web2-es toplistájának élére került. Nagy népszerűségnek örvend (főleg külföldön), mivel bárki, bárminemű HTML tudás nélkül tud saját, egyedi profilt létrehozni, blog-ot lehet rajra írni, valamint ismertségi hálózatként is működik. Beépített zenelejátszója hatalmas segítség kis-zenekaroknak (dalaik nagyközönség elé juttatásában) és nagy-zenekaroknak is (promóciós ügyekben). A szerkeszthetőségnek azonban ára van. Az oldal mellőzi az Ajax-ot, és a felhasználói tudatlanság (vagy szabadság) miatt a myspace hemzseg az áttekinthetetlen, hibás, túlzsúfolt oldalaktól. Így, az egyik legjobb Web2-es példa felületével az egyik legjobb ellenpélda is egyben.

Iwiw: Hazánk legnépszerűbb ismertség hálózatával tudatosult (nálunk) a legtöbb felhasználóban, hogy mit is jelent az új generációs világháló. Mint közösségi oldal, csak a web-en működhet, az adatokat a felhasználók hozzák létre, és azt a sajátjukként kezelik. Manapság már használja az Ajax-ot (bár a technológia nyújtotta lehetőségeket nem a legalkalmasabb helyeken használja ki), felülete nem túl letisztult, elég lassú. Sikerének kulcsa, hogy hazánkban első volt, így a „hálózat effektusnak" köszönhetően mindenki ide próbált (avagy „kényszerült") bekerülni.

Flickr: Szintén tipikusan Web 2.0-s, fénykép-megosztó alkalmazás. Az Organizr kiegészítő alkalmazás (mely a fotók rendezésére való) az Ajax segítségével asztali alkalmazás jelleget mutat. A tag alapú keresés segítségével a bonyolult összefüggések sem okoznak problémát. Mivel a képeket a felhasználók töltik fel, így ők alakítják az adathalmazt. A publikus képek elősegítik a közösségi szellemet, a privát képek pedig a felhasználó saját adatai. Bár mint fotómenedzselő alkalmazás működhetne web nélkül is, fő funkcióját így elveszítené.

Youtube: A legismertebb online videó-megosztó site, szintén húzónév az új generációban, a Myspace-hez hasonló okok miatt. Maga a megosztás összehozza az embereket, az újabban divatos „response”, azaz válasz videók pedig még jobban növeli ezt (szinte (persze nem valósidejű) videókonferencia jellegű kommunkációt folytathatunk így le, az egész közösség előtt). A videókat sajátként érezhetjük, hiszen rengeteg privát videó kerül fel. Felülete egyre fejlődik, igyekszik kihasználni az új technológiákat. A web platformként való kihasználásában megegyezik a Flickr-el.

Ebay: Felülete nélkülözi az Ajaxot, kicsit szétszórt, de viszonylag egyszerű. Közösségi szempontból viszont előremutató példa. Például pontozni lehet az emberek megbízhatóságát, így maguk a felhasználók rendezik le a kényes „megbízható-e egy ismeretlentől vásárolni” kérdést.

del.icio.us: Ugyancsak húzónév. Bár felülete nem túl letisztult, funkcióival ellensúlyozza ezt. Egyrészt (a GoogleDocs-hoz hasonlóan) on-line tárolhatjuk adataink, jelen esetben könyvjelzőinket (privát célból), valamint ezeket megfelelő tag-ekkel ellátva mások számára is könnyen fellelhetővé tesszük, így a felhasználók alakítják ki a könyvjelzők tudásbázisát.

Last.fm: Az egyik legkevésbé emlegetett, pedig (véleményem szerint) az egyik jellegzetesebb példa. Mellőzöttségét talán annak köszönheti, hogy egy konkrét dologgal, konkrétan a zenehallgatással foglalkozik, viszont akit érdekel ez a téma, azt remek funkciókkal látja el. Zenelejátszókba beépülő programjával az ember mindenféle beavatkozás nélkül feltölti a hallgatott dalok adatait, így egy hatalmas közösség tagja lesz, statisztikákat nézhet zenehallgatási szokásiról. Az előadók (manuálisan) tag-ekkel láthatók el, így könnyítve a keresést. A statisztikák kiértékelése teljesen automatikus. Legfontosabb funkciója, hogy a felhasználók statisztikáit integrálja zenekaronkénti statisztikákba. Így egy új, ismeretlen előadónál meg tudjuk nézni mely dalokat a legtöbbet hallgatottak, melyeket érdemes nekünk is meghallgatnunk. A „hasonló előadók” funkcióval pedig az általunk kedvelt előadó lapját megnézve könnyen kereshetünk ízlésünknek megfelelő hasonló zenekart (ezeknek listája szinten automatikusan alakul ki, az adott zenekart hallgató felhasználók szokásait tekintve). Saját statisztikánk alapján koncerteket, új előadókat ajánl. Így privát statisztikánkat igazán magunkénak érezhetjük, a közösség része ként pedig automatikusan mi is elősegítjük ezen funkciók működését. Felületre sem lehet panasz, Ajaxos, (a reklámoktól eltekintve) tiszta és könnyen áttekinthető a sok funkció ellenére is.

Frappr!: Az adat-, funkciókombinálás legismertebbje. Fő funkciója, hogy a GoogleMaps térképén, on-line közösségek megjelölhetik a tagok a tartózkodási helyüket.

Blogok, fórumok: A blogok írásával az emberek rögtön nagy tömegek elé tudják kitárni életüket, avagy véleményüket. Ez is nagyban elősegítette az emberek közelebb kerülését a világhálóhoz. Rengeteg weboldal üzemel blog-motorokon, a könnyű szerkeszthetőség, kezelhetőség, áttekinthetőség miatt. Fórumok segítségével pedig rendezve, rendszerezve lehet adott dolgokról vitatkozni, véleményt cserélni.

P2P: Sokan a peer-to-peer hálózatokat is hozzácsatolják a témához, nem véletlenül. Itt is az elosztottságon van a hangsúly, mint ahogy sok más Web 2.0-s témában. Hiszen mennyivel hatékonyabb és egyszerűbb, ha nem egy nagy kiszolgáló lát el mindenkit, hanem a felhasználók egymást „segítve", mindenki a maga (szerverhez viszonyítva) kis lehetőségeit felajánlva. Nem feltétlenül kell illegális cselekedetekre gondolni, rengeteg legális szoftver kerül így terjesztésre, neves szoftvercégek is felajánlanak (általában torrent-es) p2p lehetőséget letöltésre. Nyugodtan nevezhetjük a p2p-t a letöltés Web 2.0-s verziójának, aminek régi elterjedt megfelelője az FTP volt.

A Web 2.0-hoz kapcsolódó technológiák

Címkék:

A címkék további magyarázatához folytassuk tovább a fent említett Youtube-os példát. Felöltünk egy videót, ahol egy fekete macska leesik a piros háztetőről, akkor az következő címkékkel érdemes ellátnunk: „cat, black cat, falling, fall, roof, red roof". Ekkor, ha valaki olyan videókat keres ahol „piros tető" van, éppúgy ráakad, mint aki a „macska"címkére keres rá.
Sajnos sok webalkalmazás nem használja ki a címkékben rejlő igazi erőt, a *címke-láncolás*t (tag-chain). Ez azt jelenti, hogy csak egy címkére lehet keresni, nem lehet *címkékre* keresni, köztük logikai kapcsolatokkal. Így, ha valaki tényleg „fekete macska leesik a piros háztetőről" videót keres, akkor (mivel ez esetben csak egyesével tud címkékre keresni), végig kell nézni valamely egy címke összes eredményét (pl. „macsakára" keresve valószínűleg rengeteget). Jóval egyszerűbb és hatékonyabb lesz a keresés, ha implementáljuk a címke-láncolást. Így logikai „és/vagy" kapcsolatokkal összekötve a címkéket sokkal pontosabb eredményt kapunk. Zenei példával élve, ha valakinek nem jut eszébe egy (80as években rockzenét játszó orosz) zenekar neve, könnyen ráakadhat, ha egy címke-láncolást megvalósító zenei oldalon a következőhöz hasonlóan rákeres: „rock *and* 80's *and* russian". Egy társkereső oldal pedig jól példázza a „vagy" kapcsolattal való keresést: „szőke *vagy* barna *vagy* vörös" párt keresünk. Mivel általában szűkíteni szeretnénk egy bő eredményhalmazt, ezért általában az és kapcsolatot szoktuk használni.
Tipikus feature még az ún. „tag cloud" (címke felhő), mely a címkéket változó méretű szövegként sorolja fel (számosságukhoz viszonyítva), azaz minél több helyen szerepel egy tag, annál nagyobb betűmérettel szerepel a felhőben.
Mint látjuk, hatalmas előnye van a címkézésnek a keresésben és a rendszerezésben, egyetlen hátránya, hogy az entitás feltöltésekor manuálisan el kell látnunk címkével.

SEO:

Az ún. SEO (Search Engine Optimalization), azaz a kereső-motor optimalizálás manapság fontos aspektusa egy weboldal, webalkalmazás készítésekor. Akik nem napi szintű látogatói egy adott oldalnak, azok vagy más oldalak linkjei útján jutnak oda, vagy kereső-motorba beírt kifejezés eredményei alapján. Ezért fontos hogy a kereső-motorok be tudják cachelni az oldalunkon található szövegeket, de főként a kulcsszavakat.
Míg statikusak voltak a weboldalak, ez nem okozott problémát, a robot linkenként bejárta az oldalt és cachelte a szövegeket. Azonban bizonyos technikával készült oldalak bejárása nehézkes a motoroknak, régebben ilyennek tekintették a keretes szerkezetű oldalakat.
Bár adhatunk meg kulcsszavakat minden oldalhoz, ez csak félmegoldás. Egyrészt a kulcsszavak darabszámára van egy limit, valamit jórészt kézzel kell őket felvennünk (esetleg ha az oldal szerkezete olyan, akár generálhatjuk is), de ez mind kevés információt hordoz az oldal szövege nélkül.
A következő három tényezőből már egy is elég ahhoz hogy az oldalon található szöveget ne tudja a motor betölteni.

1) Az oldalra regisztrálni kell, hogy tartalmát elérjük
   Mindenképpen ügyelnünk kell rá, hogy oldalunk tartalma elérhető legyen, azonban általában mégis szükség van valami regisztrációra. Miért? Biztonsági célból.
   Bár a mai keresőrobotok be tudnak regisztrálni egy egyszerű email-címes form-on, profibb oldalakon a regisztrációkor át kell esnünk egy ún. Challenge-Respone Authentication-ön, amin a robotok megakadnak. A CRA célja hogy megbizonyosodjon róla hogy az oldal látogatója valóban ember, és nem valamilyen robot. Többféleképpen is történhet a hitelesítés, például ki kell számolni egy egyszerű matematikai feladatot, vagy egy checkbox „Ne pipáld be ha ember vagy!” felirattal, de a leggyakoribb az a kép-hitelesítéses technika. Persze ezeknek is hamar megjelent az ellenszere, a képfeldolgozó modullal ellátott robotok, azonban egy megfelelően bonyolultan megírt és egyedi képgenerálóval nem tudnak megbirkózni.
   Ha ilyen kép-hitelesítéssel szeretnénk ellátni weboldalunkat, néhány fontos dolgot figyelembe kell vennünk.
   - A képen levő szöveg ne legyen könnyen szegmentálható (a robotok ugyanis feldarabolják és szegmensenként, azaz betűnként értelmezik). Egy zajos képet hamar

megért egy gép, ezért inkább húzzuk át a szöveget egy vonallal, vagy folyassuk a betűket egymásba. Lényeg, hogy gép ne, vagy csak nehezen tudja értelmezni.

- A kép ember számára könnyen értelmezhető legyen. (Például kerüljük a kis 'l' és nagy 'I' betűket a képen)

- Hozzáférhetőség. A vakok nem fogják tudni elolvasni a képen a szöveget, számukra alternatív megoldást kell találni, például egy hanglejátszó elmond egy szót és azt kell beírni.

Általában az a célunk hogy a kereső-motorok be tudják olvasni az oldalon található szövegeket, amit el akarunk kerülni azok a spammelő robotok (melyek reklámokkal, vagy más vandalizmussal okoznak kárt). Tehát célszerű úgy megalkotni az oldalt, hogy az olvasáshoz bárkinek legyen joga, viszont „post"-oláshoz, azaz hozzászólni, cikket írni csak a regisztrált felhasználóknak. Így mindkét dolgot, vagyis az olvashatóságot, és a biztonságot is teljesítjük.

2) Az oldal (szöveg) tartalma nem HTML-ben van tárolva, hanem dinamikusan (JavaScript, Ajax, stb.) állítódik össze

   A keresőrobotok a JavaScript kódban, függvényekben levő szövegeket nem tudják értelmezni (egyenlőre), főkét ha Ajax-al egy adatbázisból dinamikusan töltjük be az oldal szövegét. Ugyancsak igaz ez egy Applet-ként futó alkalmazásra is. Ennek egyik megoldása lehet, hogy a Web 2.0-s technológiákat használó oldallal párhuzamosan van egy „puritán" verziója is az oldalnak. Persze ez is csak félmegoldás, hiszen ha a kereső-motor találtként adja az oldalunkat, akkor ezen „puritán" verzióra fog mutatni.

3) Az oldalon form-ok kitöltésével navigálunk (post)

   Képzeljünk el egy oldalt, ami egyetlen egy JSP/PHP/stb. File-ból áll, és ahol csak POST metódussal navigálunk. A főoldalon egy üres textbox, amibe adott dolgot (pl. filmcímet) beírva kapunk információt kapunk a dologról. Ezt szinten nem tudják cachelni, mivel a motor nem tudja, hogy „mit írjon be a szövegmezőbe" , így már a főoldalról sem tud továbbhaladni. Valamint mivel csak POST-al kommunikálunk, ezért linket sem tudunk készíteni egy tartalomról (hiszen minden link arra az egyetlen fájlra mutat). Bár ez eléggé sarkalatos példa, jól példázza, hogy kerülni kell az ehhez akárcsak hasonló megoldást is.

Nézzünk még pár további, kerülendő technikák vagy tanácsot.
A még manapság is divatos Flash technikával óvatosan kell bánni, mivel a robotok nem (vagy csak kevéssé) tudják őket értelmezni. Gombokat, bannereket, egyéb díszítő elemeként funkcionáló Flash animációkat nyugodtan elhelyezhetünk az oldalon, de az információk ne legyenek a Flash-ben tárolva. A Flash-es gomboknál is ügyelni kell, hogy maga a Flash animáció legyen a gomb, ne pedig a Flash-ben legyen lekódolva, hogy rákattintás esetén tovább navigáljon.
Kerüljük a keretes oldalakat (frame), egyrészt nem szeretik a keresőbotok őket, másrészt ilyenkor a kereső által eredményül kapott link csak az oldal egy részére fog mutatni (általában a főablakra), ami nem szerencsés, hogyha az oldalon való navigálást a keretbe helyeztük el.
Rendesen töltsük minden oldal „title” elemét, hiszen a keresőben ez fog megjelenni.

Hasonlóan igaz ez a HTML fejrészében megadható **meta tag**-okra. Mindenképp adjuk meg a karakterkódolást, például:

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
```

Az oldal leírásának megadása, a keresők ezt is megszokták jeleníteni:

```
<meta name="description" content="Ez az oldal arról szól, hogyan…">
```

Soroljuk fel az oldal tartalmához tartozó kulcsszavakat:

```
<meta name="keywords" content="kulcsszó1, kulcsszó2, kulcsszó3">
```

Ne adjunk meg túl sok kulcsszót, és fontos hogy ne ismételjük őket. (Régebben, ha többször szerepelt egy kulcsszó, akkor jobb helyezést ért el az oldal a találati listában, ma már ezt „büntetik” a keresőmotorok. Hasonlóan negatív következménnyel jár ha „hamis” kulcsszavakat adunk meg, tehát olyanokat adjunk meg amit tényleg szerepelnek a honlapon.
Az oldal készítőjének megadása:

```
<meta name="author" content="John Doe">
```

Adhatunk tanácsot, hogy hány nap múlva jöjjön újra a robot:

```
<meta name="revisit-after" content="5 days">
```

Egyes keresőmotoroknak többféle robotja van (pl. a Google-nál, deepbot és a freshbot), melyek különböző gyakorisággal és különböző részletességben indexeli az oldalakat.
Az oldal elévülési időpontja:

```
<META name="EXPIRES" content="5 days">
```

Általában az előzővel egyenlő időtartamot adunk meg.

Robotokra vonatkozó szabályok, megszorítások:

```
<meta name="robots" content="all">
```

Bár alapértelmezés szerint engedélyezve van, és mindent indexelnek, meg szokás adni. Használjuk a robots.txt-t is. Ezt a webalkalmazás gyökérkönyvtárában helyezzük el és (ha a teljes indexelés a célunk), a következőt írjuk bele:

```
User-agent: *
Disallow:
```

Vannak még külön keresőmotor specifikus segédeszközök, ilyen például a Google's Webmaster Tools, mely segítségével a Google-nek megfelelő ún. sitemap-et készíthetünk oldalunkról, így segítve a Google keresőmotornak oldaunk indexelését.

Hírcsatornák:

A hírcsatornák olyan kisméretű, folyamatosan változó dokumentumok, melyek a fődokumentumokhoz tartozó információk rövidebb változatát és ezekre egy-egy hivatkozást tartalmaznak. Ezeket a kisméretű dokumentumokat valamilyen alkalmazás (formázva) megjeleníti, ezeket nevezzük hírolvasóknak. Maga a dokumentum egy valamilyen szabvány szerinti XML dokumentum, legismertebbek az RSS és az Atom.
Legfőbb célja, hogy nem kell minden weboldalt egyesével nézegetnünk, hogy történt-e változás, friss bejegyzés, elég a hírolvasót figyelnünk, ahol hír nevét vagy rövid változatát elolvasva mérlegelhetünk, hogy meglátogatjuk a fődokumentumot. Ez hasznos lehet gyakran frissülő híroldalaknál (így a hír kikerülését követően akár azonnal látjuk) és ritkán, akár hetente frissülő blogoknál is (így nem kell naponta nézegetni).
Gyakorlatilag bármilyen oldal rendelkezhet ilyennel, a szerveren futó motor általában automatikusan legyártja ezeket a dokumentumokat.
A csatornát magát is felhasználhatja valamilyen alkalmazás gyakorlatilag bármilyen módon, jó példa erre a Torrent-ek és az RSS kapcsolata. Ha az oldalra feltesznek egy torrentet, az megjelenik az RSS hírcsatornán is. Az igazi erő pedig abban van, hogy egyes torrent kliensek képesek az RSS csatornák értelmezésére, benne a torrent nevének szűrésére, és (mivel ezek a csatornák magára a torrent file-ra mutató linket tartalmaznak) a torrent automatikus betöltésére és a letöltés elindítására. Így, ha például megfelelő csatornához kapcsolódóan megadjuk, hogy „ *Ferdora*Live*x86 ” akkor minden 32-bites Live verziójú Fedora automatikusan letöltődik.

Példa egy RSS hírcsatorna szerkezetére:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0">
<channel>
  <title>W3Schools Home Page</title>
  <link>http://www.w3schools.com</link>
  <description>Free web building tutorials</description>
  <item>
    <title>RSS Tutorial</title>
    <link>http://www.w3schools.com/rss</link>
    <description>New RSS tutorial on W3Schools</description>
  </item>
  <item>
    <title>XML Tutorial</title>
    <link>http://www.w3schools.com/xml</link>
    <description>New XML tutorial on W3Schools</description>
  </item>
</channel>
</rss>
```

Az RSS gyökérelemben elhelyezkedő channel elem tartalmazza a csatorna alapinformációit. A title a csatorna nevét, a link általában a csatornát tartalmazó oldal linkjét tartalmazza, a description a leírását. Az hírolvasók által általában implementált további funkciókat is kihasználhatjuk, ha megadjuk a category, illetve az image elemet, az előbbit a csatorna kategorizálását segíti, az utóbbi a csatorna megjelenítendő képe. További opcionális elemekkel megadhatjuk az olvasónak azt is, hogy milyen gyakran frissítse a csatornát, mi a nyelve, stb.

A channel-ön belül pedig tetszőtleges mennyiségű item elemek helyezkednek el, melyek a konkrét híreket tartalmazzák. Itt is többféle opcionális paraméter jelenhet meg, amit meg kell adnunk, az a title, link, description, melyek rendre a hír neve, a fődokumentumra hivatkozó link, és a fődokumentum rövid leírása.

Rich Internet Application:

RIA-nak nevezzük azokat a web-alkamazásokat, melyek asztali alkalmazásokhoz hasonló funkciókkal, tulajdonságokkal rendelkeznek. Ezek az adatok fő részét a szerveren tárolják, a felhasználó pedig a böngészőben megjelenő felhasználói felületen keresztül kommunikál a szerverrel. A felület vagy (D)HTML, vagy valamilyen konténerben futó alkamazás, Flash, Java Applet. Asztali programokhoz viszonyítva nagy előnyük, hogy nem kell őket telepíteni, és automatikusan a legfrissebb verzió fog betöltődni. Mivel interneten futnak ezért előnyük, hogy a felhasználó bárhonnan elérhető őket, hátrányuk viszont, hogy mindenképpen internet hozzáférés kell hozzájuk.
A konténerben futó alkalmazások, például egy Java Applet, szinte teljesen asztali funkciókat használhat, hiszen bármely Java Swing elem éppúgy megjelenhet benne, mint egy asztaliban. Persze ezek a konténerek egy sor biztonsági korlátozás alatt, tehát teljesen asztali funkciójuk sohasem lesz. Valamint nagy hátrányuk, hogy általában relatíve nagyméretűek, (legalább egyszer) le kell töltődniük, ami sokáig tart, a felhasználó lehet, hogy elhagyja az oldalt ez alatt. További hátrány, hogy a keresőmotorok sem tudnak az ilyen konténerekben keresni. RIA-t fejlesztéshez manapság rengeteg keretrendszer áll rendelkezésre, pl. OpenLaszlo, ami DHTML, vagy Flash is tud gyártani. A két konkurens nagy cég, a Sun és a Microsoft is készíti saját, RIA fejlesztésre kihegyezett technológiáját, a Sun JavaFX, az MS SilverLight néven.

SaaS:

A „Software as a Service” jelentése, hogy a fejlesztő webes elérhetőségű szoftvert fejleszt, üzemben tartja a szervert, amin fut, a vásárló pedig nem magáért a szoftverért, hanem annak használatáért fizet. Ez lehet egy webalkalmazás, de akár egy webszolgáltatás is. A SaaS-al kapcsolatban nem a szokásos Web 2.0-s szoftverekre kell gondolnunk, amiket „mindennapi felhasználók” használnak, hanem üzleti alkalmazásokra. Üzleti alkalmazások vevőinek is megéri, például a karbantartás lekerül a vállukról, a frissítés automatikus, bárhonnan elérhető a termék. Hátrány, ami sokakat visszatart, hogy az „adatok” is máshol (tehát nem náluk) vannak tárolva, ami biztonsági kérdéseket vet fel, valamint a szolgáltató csődje esetén az egész terméket elveszíthetik.

SPA:

A „Single Page Application” azaz (egy oldalon futó alkalmazás), olyan webalkalmazás, ami bárminemű oldalfrissítés nélkül, folyamatosan csak egy oldalon fut, az oldal manipulálásával. Lehet Ajax-os vagy Flash is, a lényeg, teljesen asztali élményt nyújt.

A Web 2.0 kritikája

Rengeteg kritika éri a Web 2.0-t, és kitalálóit, támogatóit, sokan csak reklámfogásnak tartják. Az egyik nézőpont szerint már maga a *fogalom létezése* is kérdéses, hiszen a Web is, mint a világban folyamatosan változik, fejlődik és mivel nem programról van szó, nincs oka a verziószámnak. Ráadásul a Web heterogén dolog, nem lehet az egész Web fejlődéséről beszélni, nem lesz egy csapásra minden 2.0-s verziójú.
Egy másik nézőpont, még ha el is ismeri a web ekképpen történő fejlődését, *magát a fogalmat* támadja. Mint látjuk, mindenkinek megvan a maga nézőpontja, aszerint, hogy mit tekint második generációsnak, és ezek a nézőpontok nagyban eltérnek egymástól. Így egyénenként teljesen mást jelenthet a fogalom.
Támadások érik *az alkalmazandó technológiák újszerűként valótűntetéséért*. Az felhasználói felületet létrehozandó Ajax technlógia elemei (bár a fogalom szinten új), a JavaScript, a DOM, az XMLHttpRequest már mind léteztek a Web 2.0 fogalom előtt, éppúgy ahogy voltak előtte is közösségi oldalak, vagy felhasználók által létrehozott adatok (a Wikipedia 2001-ben indult).
Sokan csak felkapott dolognak (ún. *hype*-nak) látják a dolgot. Egy üres jelentés (éppúgy, mint az önéletrajzba beírt „nyitott gondolkodású, dinamikus fiatalember vagyok” sor), amit mindenki rásüt az oldalára, csak azért, hogy „divatos” legyen. Inkább reklámfogalomnak tekintik, mint új technológiának, és úgy vélik az egész jelentés éppúgy ki fog „durranni”, mint 2000-2001-ben a „Dot Com lufi” (lásd függelék).
Mások szerint maga *fejlődés is megkérdőjelezhető*. Szerintük ugyanazokról technológiákról van szó, mint eddig és semmilyen új szemléletet nem látnak az új webalkalmaztásokban. Tehát az ő szemléletük hasonló egy fentebb említetthez, de ők még a változást sem ismerik el.

Mint általában minden kritikának, ezeknek is van valóság alapjuk. Mindenbe, ami nem pontosan definiált, bele lehet kötni, bele lehet magyarázni más dolgokat, emiatt könnyen érheti sok támadás. Általánosan azonban elmondható, hogy a kritikák nagytöbbsége túlságosan komolyan veszi a Web 2.0 fogalmat. Maguk is a fogalom megalkotói és a fogalom mellett érvelők is úgy tartják, elég ha irányelvnek, követendő példának, technológiai ajánlásnak veszik a fejlesztők a Web 2.0-t körülvevő dolgokat, nem pedig szabályként.

Az Ajax technológia

Az Ajax (azaz Aszinkron JavaScript és XML) technológia fogalmát Jesse James Garrett alkotta meg. Bár köznapi szóhasználatban technológiaként emlegetjük, igazából csak egy programozási technika. Maga Garrett a következőképp fogalmazza meg:
„Az Ajax nem egy technológia. Igazából néhány technológia összessége, amely együtt új erőt alkot. A következők egyesítéséből áll:

- *XHTML* és *CSS* a megjelenítésre
- dinamikus megjelenítés és interakció a *DOM* által
- adatcsere és manipuláció *XML* és XSLT által
- aszinkron adatkinyerés *XMLHttpRequest* által
- *JavaScript* ezek összekötésére

A fogalmak áttekintése előtt röviden nézzük meg általában mit is jelent ez.
Maga a weboldal szimpla HTML és CSS-ből áll, ezek szerkezete adja a DOM-ot, azaz a röviden mondva a dokumentum fáját, benne a HTML elemekkel. Az oldal kódjába írt JavaScript kód valamilyen esemény hatására létrehozza az aszinkron XMLHttpRequest objektumot, mely aszinkron kérést indít a szerver felé, majd válasz XML vagy szöveg (szintén JavaScript-es) feldolgozása után, a módosítja a DOM-ot, ezáltal magát az oldalt.
A legfontosabb, hogy a felhasználó szemszögéből nézve, ő csak megnyom egy gombot, és az oldalon megjelenik az általa kért adat (a teljes lap frissítése nélkül, a kommunikáció a háttérben zajlik). Ráadásul mindezt aszinkron módon, tehát nem „fagy be” a böngésző a válaszig, lehet tovább olvasni, írni.

Vegyük tehát sorra az Ajax által használt technológiákat. Látni fogjuk, hogy maguk a technológiák nem újak, már jó 10 éve létezhetne az Ajax.

Az Ajax komponensei

XHTML

A HTML-hez képest az X betű az Extensible, azaz a kiterjesztőség jelölése. Az alap HTML SGML (Standard Generalizes Markup Language)-ben van megfogalmazva, az XHTML ezzel szemben XML-ben, ami jóval szigorúbb. Ennek a fő jelentőse, hogy az XHTML dokumentumoknak így jól formázottnak kell lenniük, hogy automatikusan is fel lehessen őket dolgozni könnyedén. Az XHTML 1.0 2000-ben lett W3C ajánlás, utódja, az XHTML 1.1 pedig 2001-ben, jelenleg pedig a 2.0-s verzió szabványosításán dolgoznak.

CSS

A CSS egy ún. stílusleíró nyelv, melynek célja hogy a strukturált (de akár nyers) HTML vagy XHTML vagy dokumentum megjelenítésének leírása. CSS-ben bármely HTML-ben szokásos formázás, színek, elhelyezkedések, kiemelések definiálhatóak.
A fő cél a CSS megalkotásakor, hogy így *elkülöníthetjük az adatot és a prezentációt*, azaz a megjelenítést. Így, az ajánlás szerint, az (X)HTML dokumentumban nem szabad definiálnunk semmilyen formázást, minden formázást CSS-ben kell megadnunk, és a HTML elemek „class” tulajdonságával hivatkoznunk rá (illetve egyéb módon, névvel, vagy id-vel is azonosíthatóak). Így, ha egy alkalmazásnak új kinézetet szeretnénk, nem kell az általában nagy és összetett HTML kódot átírnunk, mindössze a stíluslapot kell módosítani.
Ráadásul elég egyszer definiálnunk egy stílust, majd a kívánt elemeknél hivatkoznunk rá. CSS nélkül minden kívánt elemnél definiálnunk kellene ugyanazt az esetenként bonyolult stílust, így fölöslegesen többszörösen lenne letárolva ugyanazon definíció. Így tehát CSS-el maga a dokumentum mérete is csökken.
Mivel egymásba ágyazhatók ezek a stílusok, így egy elem megjelenítését nem egy darab stílus határozza meg, hanem egymásba ágyazott stílusok sorozata, melyek esetekben felülbírálják egy megelőző stílus valamely elemét. Ha például a HTML paragrafus elemre megadunk egy háttérszínt, az mindenre igaz lesz, ami a paragrafuson belül található, kivéve, ha felülbíráljuk.
A CSS hátránya, hogy használatával könnyen előjönnek a böngészők közötti különbségek, akár szabványos CSS használat esetén is. Ráadásul, ha csak egyes böngészők által használt,

nem szabványos definíciókat adunk, könnyen teljesen más lesz a végeredmény böngészőnként.
A stíluslapok ötlete és hasonló megoldások már az SGML megjelenésekor felvetődtek, maga a CSS1 végleges specifikációja 1996ban jelent meg, böngészőbe való implementálás pedig még három évvel később.

DOM

A DOM a HTML és XML dokumentumok objektummodellje, azaz a szerkezetük modellje. Platform és nyelv független, segítségével a böngésző és a JavaScript eléri és módosíthatja a dokumentum elemeit, struktúráját. Módosítás után a böngésző újraértelmezi a modellt és frissíti a megjelenítést.
Magát a modell-t egy faszerkezetként lehet a leginkább elképzelni, melyben HTML esetén a gyökérben a „document" található, azon belül a HTML elem, melynek gyermeke a HEAD és a BODY, és így tovább.
A DOM első specifikációja 1998-ban jelent meg, de még így is jó ideig kellett várni, míg a legtöbb böngésző megvalósította, így azelőtt a DOM-ot használó kódokat böngészőnként másként kellett megírni.

XML

Az XML megjelenést, az informatikára ritkán jellemző, egyöntetű támogatás fogadta. Tulajdonképpen egy strukturált szövegfájl, megjelenésének fő célja azaz igény, hogy egy sima, struktúra nélküli szövegfájl nem (vagy csak nehezen) értelmezhető egy program számára. Így mindenkinek volt egy-egy megoldása a strukturált adatok tárolására és továbbítására, azonban ezek nem voltak szabványosak. Az XML leírása egy megoldás erre, és mivel szabványos, minden program értelmezni tudja (természetesen, ha implementál valamilyen XML feldolgozót).
Egy XML dokumentum is, a HTML-hez hasonlóan tagok, elemek, attribútumok definiálásával jön létre, azonban jóval kötöttebb a formája, elvárás hogy a dokumentum „jól formázott" legyen, azaz eleget kell tennie az XML szintakszisnak, máskülönben nem dolgozható fel. Persze a HTML-re is vannak formai szabályok, azonban a böngészők ezt

általában engedékenyen kezelik, XML esetén viszont alapvető, hogy jól formázottnak kell lennie.

Az XML helyesség modelljére ráhúzható még egy modell, az érvényesség modellje. Ez azt jelenti, hogy nem elég hogy a dokumentum jól formázott is legyen, de helyesnek is kell lennie, azaz meg kell felelnie bizonyos szemantikai szabályoknak is. Ezeket a szemantikákat valamilyen sémával adjuk, valamilyen sémaleíró nyelven (DTD, W3C XML Schema). A sémában nem csak az szerepelhet, hogy például egy mezőben csak szám állhat, hanem olyan kötöttségek is, hogy egy elemen belül milyen elemek szerepelhetnek, milyen számosságban. A következő dokumentum például jól formázott, de olyan értelmezésben, hogy a number elemen belül csak szám állhat, nem érvényes.

```
<?xml version="1.0" ?>
<numbers>
      <number>
      szöveg
      </number>
</numbers>
```

Ezekre a megkötésekre azért van szükség, mert az XML dokumentumok nem emberi olvasásra készültek, hanem programok közti információcserére, így kritikus fontosságú, hogy értelmezhető legyen (jól formázott), és a nekünk megfelelő érvényes szerkeszthető és tartalmú legyen (helyes). A feldolgozás általában kétféleképpen történik, két teljesen más filozófiát valló eszközzel, az egyik a SAX (szeriális feldolgozás), a másik a DOM (fában történő feldolgozás).

Mint ebből az egyszerű példából is látjuk, rengeteg redundáns adat szerepel benne, ez azonban elengedhetetlen, hogy egyértelmű legyen.

Előnyei közé tartozik, hogy szöveg alapú, tehát nem bináris, szinten minden igényt kielégít struktúra szinten, bármi ábrázolható benne, általánosan elfogadott, standardizált, platform-független.

Hátránya az előbb említett redundancia, nagy mérete (bináris adatábrázoláshoz képest).

Maga a technológia alapköveit Tim Bray fejlesztette ki, miután egy munkája során találkozott egy, az XML elődjének, vagy inspirációjának számító struktúra-szerkezettel. Egy tizenegy tagú munkacsapat fejlesztésének eredményeképp 1998 vált W3C ajánlássá. Szétnézve a mai informatika világában hatalmas sikert ért el, hiszen ma már nem is technológiáról beszélhetünk, hanem technológia családról.

XMLHttpRequest

Az XMLHttpRequest egy olyan objektum, mely a kliens oldali script (JavaScript) által használható XML vagy egyéb szöveg küldésére és fogadására http protokollon keresztül, egy web szerver felé. Fontos, hogy a két oldal között aszinkron csatorna jön létre, így a küldés/fogadás idejére nem áll le az alkalmazás.A koncepciót a Microsoft fejlesztette ki 2000-ben, később a Mozilla is implementálta, szabványosítása még mindig folyamatban van.
Az Ajax technológia legfontosabb része, mivel ő valósítja meg az aszinkron hívást.

JavaScript

A JavaScript egy script nyelv (azaz minden futáskor lefordítódik, avagy jelen esetben interpretálódik), az ECMAScript egy implementációja. kliens oldali webfejlesztéshez. Objektum-orientált, de nincsen osztály, gyengén típusos. Közhasználatban a JavaScript alatt igazából a böngészők ECMAScipt implementációját értjük. Történeti szempontból ugyanis a JavaScript egy Netscape fejlesztés, melyet az 1996-os Netscape Navigator implementált először. A sikerre való tekintettel a Microsoft kifejlesztette az ezzel ekvivalens kliens-oldali script nyelvét, a JScript-et. Ezután a JavaScript előterjesztésre kerül az ECMA International-nak, ők standardizálták, ezért mondják azt, hogy mind a JavaScript, mind a JScript az ECMAScript egy implementációja. Mint látjuk, nevén kívül nem sok köze van a Java programozási nyelvhez.
A nyelv elsődleges célja, hogy a HTML-be ágyazott kód segítségével, manipuláljuk a DOM-ot, valamint Ajax esetén ő hozza létre az XMLHttpRequest objektumot, majd dolgozza fel a kapott eredményt. JavaScript programozáskor erősen támaszkodunk a böngészőben végrehajló eseményekre, például akkor futtatunk le egy függvényt, amikor egy gombot megnyomnak, vagy ha a kurzor egy bizonyos hely kerül, vagy ha leütnek egy billentyűt. Fontos alkalmazása még az ún. kliens oldali validálás. Ez azt jelenti, hogy mondjuk egy születési év mezőbe beírt évszámot már kliens oldalon leellenőrizünk, hogy 1900 és 2007 közé eső egész szám legyen. Mivel a kódunkat a böngészők más-más módon futtathatják, valamint mivel a kód általában önmagában (a HTML kód nélkül) nem értelmes, nem futtatható, ezért a debuggolás (nyomkövetés) különösen nehéz lehet. Célszerű a kész kódot minden támogatni kívánt böngészőben letesztelni, és a böngésző saját JavaScript nyomkövetőjét használni (ha van).

Az Ajax és a webfejlesztés sajátosságai

Dinamikus megjelenítés, felhasználói felület

Már az Ajax előtt is létezett két technika mely hasonlított eredményében.
Az egyik, a dinamikusan változó oldal, JavaScript segítségével. Itt persze csak annyiról van szó, hogy az oldal letöltésekor már elérhető minden információ, a JavaScript a már letöltött meglevő adatokból dolgozik. Például egy képsorozat minden képe letöltődik, de elrejtjük őket, és egy gomb megnyomására váltakoznak. Bár teljesen *úgy tűnik, hogy gombnyomásra töltődnek be*, az *összes adat rendelkezésre kell, hogy álljon*, már akkor, mire betöltődik az oldal.
A másik hasonló technika a keretes oldalak. Itt pont az előző ellentéte figyelhető meg, *látjuk, hogy az oldal éppen akkor töltődik be* (amikor például megnyomunk egy gombot), viszont az adatok gombnyomásra kérődnek le a szervertől, tehát *nem kell, hogy rendelkezésre álljanak* az oldal betöltésekor. A felhasználó tehát látja, hogy ekkora az oldal egy része (a frame) újratöltődik.
Az Ajax-al e kettő technika ötvözete jelenik meg. Az adatok nem kell, hogy rendelkezésre álljanak az oldal betöltésekor, és nem kell egy egész oldalnak (vagy frame-nek) újratöltődnie. A JavaScript mód a DOM bármely részét megváltoztathatja, legyen az egy új elem beszúrása, egy elem szövegének megváltoztatása, egy elem stílusának megváltoztatása, valamint igaz ez az elemek tényezőinek elérése is.
Kerülő megoldás volt még az Ajax megjelenése előtt, hogy rejtett keretekben (iframe-ekben) folytatták le a háttér-kommunikációt.

Válaszidő

Az a Ajax módosítja a szokásos „optimális válaszidő” modellt is. Azt szokták mondani, hogy a válaszidő három tényezővel jellemezhető.

- 0,1 másodperc az ideális válaszidő
- 1 másodperc a még elfogadhat válaszidő
- 10 másodperc olyan, mintha nem is lenne válasz, a felhasználó elhagyja az oldalt

Ajaxnál ez picit módosul, az aszinkronság miatt a felhasználók elnézőbbek, különösen, ha nem „kritikus” információt töltünk be, hanem valami extra dolgot, ami nélkül az oldal fő funkciója megmarad.

Szerverterhelés csökkentése

Mivel a kommunikáció XML-ben is történhet, lehetőség van rá, hogy a HTML kódot ne a szerver, hanem a kliensböngésző állítsa össze JavaScript-el. Így elosztódik a terhelés, nem egy szerveren dolgozik minden igényelt lap legyártásán, hanem mindenkinek a saját böngészője, így a kliens terhelésével csökken a szerver terhelése. (Itt is megmutatkozik az egész Web 2.0-t jellemző elosztottság.) Azonban ennek ára, egyrészt JavaScriptben bonyolultabb módosítani a DOM-ot(például a semmiből létrehozni egy táblázatot), mint egy szerveroldali nyelven (pl. jsp, php) megírni a statikus HTML-at és a belefűzni a dinamikus adatokat. Ráadásul, egy szerver oldali nyelvhez (mint pl. a JSP) jóval bővebb eszközkészlet áll rendelkezésünkre, mely akár egy teljes táblázatot automatikusan gyárt le. Ilyenkor nem küldhetünk XML-t, de az XMLHttpRequest objektum szöveget is képes átvenni, így nem ütközünk akadályba. Ráadásul sávszélességben sincs különbség, hiszen az XML rengeteg járulékos szerkezeti információt tartalmaz a „nyers adat” mellett.
Ha tehát nem nagy szerverterhelést létrehozó programot fejlesztünk, akkor küldhetjük az beszúrandó adatot rögtön formázott HTML-ként, így kihasználhatjuk a szerveroldal HTML formázási funkcióit is.

Cachelés:

A cachelés egyébként is probléma a webfejlesztés terén, Ajaxnál ez különösen elő jön.
Ha az XMLHttpRequest-en keresztül is hívjuk meg például a ’pavarotti/albums.xml’ file-t, a böngésző cachel-i azt, majd hiába módosítjuk, újra meghívva is a régi file-t fogjuk kapni.
Erre szabványos módszerként az a következő meta-tagok megadását ajánlják:

```
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Expires" CONTENT="-1">
```

vagy például ha Java-ban állítjuk elő a választ, akkor

```
response.setHeader("Cache-Control", "no-cache");
response.setHeader("Pragma", "no-cache");
response.setHeader("Expires", "-1");
```

Ezekkel, mindössze egyetlen gond van, a böngésző vagy be tartja őket, vagy nem.
Célszerű tehát valami más módszerhez folyamodni, mint például az alábbi, nem túl szép, de tökéletesen hatásos.
A módszer tulajdonképpen annyiból áll, hogy egy véletlen értékű paramétert illesztünk az URL végéhez. Nem a ’pavarotti/albums.xml’-t fogjuk ezután meghívni, hanem a ’pavarotti/albums.xml?nocache=véletlenszám’. Ez a véletlen szám lehet tényleges véletlen szám, de akár az aktuális dátum és idő is. Az URL-hez hozzáillesztett plusz paraméter semmilyen módon nem érinti a meghívott file-t. Ha nem egy file, hanem egy szerver oldali program (pl. jsp), akkor sincs gond, mivel azokat a paramétereket, amiket nem olvastatunk be a kódban, azokat figyelmen kívül hagyja.
Minden URL-nek meg kell felelnie a szabványnak, tehát egy paraméter esetén így

```
’valami.html?paraméter=576’
```

több paraméter esetén így kell kinéznie.

```
’valami.html?első_paraméter=576&másodikparaméter=567&harmadik_paraméter…’
```

Tehát az első paraméternek ’?’ a többinek ’&’ jellel kell összekötve lennie.
Nézzünk hát egy konkrét megvalósítást:

```
function noCache(url){
if ((/\?/.test(url))
return url.concat(”&nocache=”+new Date);
else
return url.concat(”?nocache=”+new Date);
}
```

A `/\?/` rész egy reguláris kifejezés, maga feltétel azt teszteli található-e ’?’ az URLben. Ha igen, akkor &-el fűzi hozzá a cache elkerülésére bevezetett véletlenszámot, egyébként ’?’-el. Innentől annyi a dolgunk, hogy az Ajax hívásokban nem az URL-t hívjuk meg, hanem a nocache(URL)-t.

Cross-Platform

A webfejlesztés különlegessége, hogy különböző platformokra (böngészőkre) fejlesztünk, viszont mindegyiken (közel) tökéletesen kell működnie a programnak. (Ráadásul e nélkül is két platformon történik a fejlesztés, a kliensoldalon kívül a szerveroldalon, egy teljesen másik nyelven.) Már egy kisebb program megírásakor is szembesülünk azzal, hogy mennyire más lesz az eredmény, ha nem elég körültekintően járunk el.

Érdemes megnéznünk az alábbi, 2007 harmadik negyedévi kimutatást a böngészők használatáról.

```
Internet Explorer:      81.63%
Mozilla / Firefox:      13.49%
Safari:                 03.00%
Opera:                  00.66%
Netscape Navigator:     00.06%
```

Itt azért megmutatkozik az operációs rendszerek eloszlása is, hiszen Internet Explorer a Windows beépített böngészője, a Safari a Mac OS X-é, a Mozilla pedig Linuxé. Az átlagfelhasználók többsége pedig nincs tudatában, hogy a beépített mellett másik böngészőt is használhat.
Mint látjuk, ha a programunk legalább Internet Explorer és Gecko alapú böngészők (pl. Firefox) alatt működik, már lefedtük a böngészők több mint 90%-át. Azonban a két böngésző meglehetősen különböző. Az Microsoft a Vista fejlesztése közben igencsak elfeledkezett a böngészőjéről és mire a 7-es verziójú Explorer megjelent, már jópár felhasználó áttért a sokkal barátságosabb Firefox-ra, vagy Operára. Az Explorer felhasználók nagytöbbsége még ma is az XP-be beépített 6-os verziót használja.
Megjelenítésben nagy különbségek lehetnek Explorer és Firefox között, ezért ajánlott nem egy böngészőn fejleszteni az alkalmazást, hanem új feature-ök esetén rögtön tesztelni a másikon. Különösen igaz ez a dinamikus részekre és a stíluslap szerkesztésekre. Következő pontunkban látni fogjuk, hogy általában melyik böngészőre esik a választás, mint a fejlesztés irányadójára.
Az egyik legtipikusabb példa a két böngésző közti különbségre az XML file-ok kezelése.

Vegyük az alábbi példát:

```
<?xml version="1.0" ?>
<root>
      <child>valami</child>
      <child>valami2</child>
</root>
```

A Firefox kissé másképp kezeli ezt, mint az Explorer, ugyanis minden (tagon kívüli) whitespace-t szövegelemként értelmez, az alábbi módon:

```
<?xml version="1.0" ?>
<root>
[szöveg]<child>valami</child>
[szöveg]<child>valami2</child>
</root>
```

Így, az XML bejárásakor gondok adódhatnak, ugyanis a root elem első gyermeke nem az első child elem lesz, hanem a [szöveg].
Célszerű ezeket a whitespace-eszek eltávolítani, így az XML-t azonos módon tudjuk kezelni mindkét böngésző alatt.

```
function removeWhitespace(xml)
{
var loopIndex;
for (loopIndex = 0; loopIndex < xml.childNodes.length; loopIndex++) {
      var currentNode = xml.childNodes[loopIndex];
      if (currentNode.nodeType == 1) {
      removeWhitespace(currentNode);
      }
      if (((/^\s+$/.test(currentNode.nodeValue))) &&
      (currentNode.nodeType == 3)) {
      xml.removeChild(xml.childNodes[loopIndex--]);
      }
   }
}
```

Így, első használat előtt eresszük rá ezt a függvényt az XML-re, ezután már biztonsággal használhatjuk.

Nyomkövetés

A webfejlesztés nyomkövetésének legfőbb része a JavaScript és a CSS nyomkövetése. Firefox alatt beépített DOM Inspector-t találunk, mellyek folyamatosan nyomon követhetjük a DOM aktuális állapotát, faszerkezetben ábrázolva, benne az elemek aktuális tulajdonságaival és értékeivel. Ugyancsak beépített a Hibakonzol mely a JavaScript hibákat és CSS figyelmeztetéseket jeleníti meg, eléggé informatív módon. Különös a JavaScript hibák nyomon követése kényelmes, mivel szemléletesen jelzi, hogy a script mely sorában található a hiba. Jól elkülönül, hogy a script interpretálásával volt baj (szintaktikai hiba), ez már betöltéskor jelentkezik általában, vagy futás közben történt (például kiírja, hogy nem létezik az „xy" nevű elem, amire a script hivatkozik.
A beépített modulokon kívül rengeteg kiegészítő található a Firefox-hoz webfejlesztés témában, melyek közül kettőt emelnék ki, az egyik a FireBug, a másik a Web Developer. Nagyjából hasonló funkciókkal rendelkeznek, leginkább megszokás kérdése, ki melyiket használja főleg. Legfőbb funkciójuk, hogy egy-egy elemről külön információkkal tudnak szolgálni, letilthatjuk tesztelés céljából a böngésző egyes funkcióit, könnyen elérhetjük az alkalmazás CSS és JavaScript kódját (melyet akár valós időben is módosíthatunk). Különösen hasznos az egymásba ágyazott CSS stílusok rengetegében, hogy meg tudják mondani egy adott elemre mely stílus definíciók vannak érvényben, melyek felülbírálva, és ezeket melyik stílusból örökölték.
Internet Explorer alatt már kevésbé rózsás a helyzet. Már az elég problémás, hogy sok esetben nem a szabvány szerint zajlanak a dolgok, de a beépített, nyomkövetést segítő eszközök száma gyakorlatilag nulla (még az oldal forrásának vizsgálata esetén is mindössze a jegyzettömb jön). Az Explorer is figyelmezet JavaScript hiba esetén, azonban ezek jórészt semmi információtartalommal nem bíró üzenetek („a várt elem objektum"). Van azonban egy beépülő modul, mely ezek hiányát részben enyhíti, melynek neve „Internet Explorer Developer Toolbar". Ebben már található DOM figyelő, és egyéb hasznos funkciók, azonban még mindig el marad Firefox mögött, különösen a JavaScript nyomkövetésben.
Számomra tehát nyilvánvaló, hogy a fejlesztés fő irányvonala a Firefox-ban érdemes hogy zajoljon, mivel jóval gazdagabb funkciókban, azonban emellett mindenképpen szánjunk elég idő a tesztelésre Explorer alatt is, hiszen a legtöbb felhasználó ezt használva fogja böngészni oldalunkat.

Vissza gomb

A böngésző a „vissza” gomb megnyomásakor az „előzmények”-ből olvassa be az előző újonnan betöltött oldalt. Ajaxnál ez próbléma, mivel nincs „új, betöltött oldal”, hiszen csak a DOM-ot manipuláljuk, nem keletkezik a böngészőben új előzmény objektum.
Maga a gomb nem működése nem biztos, hogy problémát okoz, hiszen ha olyan az alkalmazásunk, hogy bármikor elérjük az alapfunkciókat, nem kerülünk zsákutcába, stb., akkor nincs is szükség rá.
Ha mindenképpen kell a „vissza” funkció, akkor két lehetőségünk van. Az első, hogy saját JavaScript-ben implementált vissza gombot használunk, ami nem a legegyszerűbb, hiszen folyamatosan tárolnunk kell, hogy mikor és mi változott az oldalon, hogy a gomb megnyomásakor az oldal állapotát valamely megelőzőre visszagörgessük. A másik lehetőség, hogy valahogyan vegyük rá a böngészőt, hogy mindig tárolja le az oldal aktuális állapotát. A GoogleMaps ezt IFrame-ek használatával oldja meg.
Ami biztos, valamilyen Ajax keretrendszer használata nélkül, nekünk kell gondoskodnunk a vissza gomb kezeléséről, avagy olyanra írjuk az alkalmazásunk, hogy e nélkül is könnyen lehessen navigálni rajta.

Vizuális jelek

Ha az alkalmazásunk nagymennyiségű adatot tölt be, vagy a felhasználónak lassú az internet-hozzáférése, akkor jó időbe telhet mire egy Ajax által beszúrt elem megjelenik az oldalon. Ha nem hagyunk semmilyen vizuális jelet arról, hogy itt bizony még adatátvitel van folyamatban, akkor felhasználó azt hiheti, hogy a funkció nem is működik (hiszen nem látja a kért elemet). Ez különösen fontos akkor, ha az oldalainkat, akár a főoldalt is, fokozatosan építjük fel Ajax segítségével, ekkor tudatosítanunk kell a felhasználóval, hogy amit lát, az még nem a kész oldal.
Ez a vizuális jel lehet bármi, akár egy kis homokóra is, ám Ajaxnál általában „Ajaxos” betöltő jelet használunk, ami lehet egy kör alakban járkáló, árnyékot húzó pötty, vagy vonal. Ugyancsak fontos nemcsak a betöltődés előtt jelezni, hogy adatforgalom van folyamatban, hanem azt is, hogy a betöltődés elkészült. Ha egy nagy videót szúrunk be, az nyilván észrevehető lesz, de ha egy folyamatosan frissülő táblázat egy eleméről van szó, már nem. Lehet, hogy a felhasználó azt sem tudja, hogy változott valami adat, így valószínűleg nem

fogja észrevenni. Ezért fontos, hogy ilyen esetekben valami feltűnő módon, például piros színnel jelezzük. Ez már megszokott dolog az asztali alkalmazásoknál, azonban web-fejlesztésnél eddig nem volt ilyen probléma, mivel az oldal csak akkor változott, ha azt a felhasználó is akarta, manapság viszont akár az ő tudta nélkül is folyamatosan frissülhet az oldal.

Az irányítás meghagyása

Könnyen eshetünk abba a hiába, hogy az újdonságként számító technológiát lépten-nyomon használjuk. Nem biztos, hogy a legjobb megoldás, ha az alkalmazásunk folyamatosan elveszi az irányítást a felhasználótól az állandó frissítgetéssel, és automatikus beszúrásokkal. Ha a felhasználó azt érzi, hogy a program nem az ő általa elvártan működik, sohasem tudja, hogy éppen mi (és miért) történik a háttérben, hamar elveszíti érdeklődését.
Különösen igaz ez, ha felhasználó akarta ellenére küldözgetjük az adatokat.
Nézzünk egy példát. Roppant idegesítő, ha például egy regisztrációs form-on írnánk be a felhasználó nevet, és az oldal karakterenként ellenőrzi, hogy a bevitt név nem-e foglalt. Bőven elég ezt megtennünk a végén, ha a felhasználó elhagyta a mezőt, vagy már 1-2 másodperce nem gépel.
Ennél drasztikusabb példa, ha valamilyen adatbázist kezelünk web-en keresztül és úgy kerülnek be adatok az adatbázisba, hogy nem is akartuk. Sohase küldjük hasonló alkalmazásokban „érzékeny” adatokat a felhasználó jóváhagyása nélkül. Szép dolog az automatikus adatküldés, de ilyenkor inkább használjuk a jó öreg gombokat. Várjuk meg amíg a felhasználó tényleg el akarja küldeni az adatot.
Tartsuk tehát szem előtt, ne az alkalmazás irányítsa a felhasználót, hanem a felhasználó az alkalmazást.

B-terv

Nem minden régi böngésző támogatja a JavaScript-et, de az is lehet, hogy a felhasználó kikapcsolta azt. Ha alkalmazásunkat számukra is elérhetővé akarjuk tenni, készítenünk egy B-tervet, azaz az alkalmazásnak egy olyan verzióját, ami JavaScript nélkül is működik, és ez párhuzamosan fut az Ajaxos oldallal.

Nem csak a felhasználók, hanem a kereső-motorok miatt is igaz ez. Alkalmazzuk hát a SEO módszereit, így az oldalul kereshetővé válik számukra. Célszerű minden, JavaScript nélküli (azaz az egyszerűsített) oldalra elhelyezni egy linket a fő alkalmazásra, így aki ezt a linket kapná kereső-találtként, könnyen átnavigálhat a „valódi" oldalra.

Keretrendszerek

Mostanra rengeteg kisebb nagyobb Ajax keretrendszer került elkészítésre, mind kliensoldaliak, mind szerveroldaliak. Vannak, melyek csak az Ajax hívás egy általános formáját valósítják meg, és akadnak egész komolyak is. Jó dolog használni valamilyen keretrendszert, különösen, ha nem szeretnénk elmélyedni az Ajax rejtelmeiben, mivel általában úgy vannak implementálva bennük a funkciók, hogy nekünk nem kell törődnünk a böngészők közti különbségekkel és egyéb problémákkal.
Az egyik legérdekesebb közülük a Google Web Toolkit, melyet a Google saját eszközeinek fejlesztéséhez fejlesztett ki. Érdekessége a Java-JavaScript fordító, így alkalmazásunkat megírhatjuk Java nyelven, megszokott IDE-nkben.

Same Origin Policy

A kliensoldali script-ek egyik korlátozása, mely szabályozza, hogy egy adott helyről betöltődött script ne tölthessen be más helyről származó adatokat. Ez azért fontos, mert egy adott weblap látogatásakor így biztosak lehetünk benne, hogy az nem fog direkt kapcsolatot létesíteni köztünk és egy másik forrás között. Ahhoz hogy ne sértsük meg a korlátozást, mind a protokollnak (http,https, stb.), mind a „host"-nak (azaz a címnek) meg kell egyeznie. Nem engedélyezett például a http://www.oldalam.hu/teszt1/x.html forrásból https protokoll betöltése, vagy http://oldalam.hu/teszt1/x.html betöltése (az hostcím különbözősége miatt).
A korlátozás megkerülésének egyik módja, hogy a szerveren létrehozunk egy proxy-szerű dolgot, és ezen keresztül fogjuk betölteni a másik forrást.
Így a http://www.oldalam.hu/teszt1/proxy -ra hivatkozva már nem sértjük meg a korlátozást.

JavaScript tömörítés

Nagyobb méretű webalkalmazásoknál szokás, hogy igyekeznek a hivatkozott JavaScript file-t bizonyos technikákkal betömörítik, melynek nyilvánvaló célja, a sávszélesség csökkentése. Már egy szimpla tömörítővel, ami a fölösleges whitespace-eket távolítja el a kódból, valamint rövidre cseréli a változóneveket, körülbelül a felére csökkenthetjük a méretet. Ennek semmilyen hátránya nincs, hiszen a kód maga ugyanazt jelenti, amit eltávolítunk az mindössze az kód-olvashatóságot segítő részek. Vannak még elvetemültebb módszerek, melyek szinte bináris file-ként tömörítik, ezek azonban már kliens oldalon több processzoridőt igényelnek, mire „kitömörítődnek" futás előtt.

Egy rövid kód, tömörítés előtt:

```
xmlHttp.onreadystatechange=function()
  {
  if(xmlHttp.readyState==4)
    {
    var xml =xmlHttp.responseXML;

      similars = xml.getElementsByTagName("name");

      var i;
       var similar="<div class=\"bbg\"><b>Similar Artists:
       </b></div><br>";

      for (i=0;i<5;i++) {
      similar+=(similars[i].firstChild.nodeValue);
      similar+="<br>"
      }

      document.getElementById(div_id).innerHTML=similar;

    }
  }
```

A kód, tömörítés után:

```
xmlHttp.onreadystatechange=function(){if(xmlHttp.readyState==4){var
_1=xmlHttp.responseXML;similars=_1.getElementsByTagName("name");var i;var _3="<div
class=\"bbg\"><b>Similar Artists:
</b></div><br>";for(i=0;i<5;i++){_3+=(similars[i].firstChild.nodeValue);_3+="<br>";}document.g
etElementById(div_id).innerHTML=_3;}};
```

Ajax minták

Form-validálás

Eddig is gyakran alkalmazott technika volt, hogy a form-okon a felhasználó által bevitt adatokat JavaScript-el ellenőrizték, például hogy a születési dátumnak tényleg szám lett-e megadva. Ajax-al azonban lehetőségünk van arra, hogy az olyan (csak szerver által ismert) információkat is validáljunk, mint hogy nem-e foglalt a felhasználó név. Így egy regisztrációs form-on, mire a felhasználó jelszavát gépeli be, már tudathatjuk is vele, hogy a név foglalt

Automatikus kiegészítés

Nem vitás, hogy az Ajax egyik úttörő alkalmazása, Jesse James Garrett egyik példája a cikkében, a Google Suggest is alkalmazza.
Az ötlet maga az, hogy például egy hosszú városnévnél fölösleges végiggépeltetnünk az egész szót a felhasználóval, hiszen egy idő utána egyértelmű lesz, hogy mire gondol (vagy ha nem is egyértelmű, megmutatjuk neki a több lehetséges variációt).
Ennek több előnyös oldala is van, egyrészt a felhasználó időt takaríthat meg. Másrészt csökkenthetjük az elgépelések számát is, hiszen ha egy fél szó begépelése után a felhasználó az ajánlást választja, nem lesz lehetősége sem elrontani a szót. További előny lehet, ha olyasmit keres (avagy akar beírni) a felhasználó, aminek csak egy része jut eszébe. Például egy dalcím esetén, ha csak a cím közepe ugrik be neki, azt begépelve és az ajánlatokat látva valószínűleg megtalálja, amit keres.
Ennél a mintánál különösen fontos, hogy körültekintően implementáljuk, hogy a segítségből ne hátrány legyen. Nem kell minden billentyű leütésekor lekérni a szervertől az ajánlatokat, hiszen egy gyorsan gépelő embernél (és hosszabb szónál) hirtelen nagy sávszélességet fog lefoglalni. Bár ez mindig alkalmazás függő, általában érdemes 1-2 másodpercenként lekérni a mező értékét.

Automatikus betöltés

A „load on demand”-ra jó példa lehet egy képnézegető webalkalmazás. Az asztali programok mind használják azt a feature-t, hogy míg gyönyörködünk az n-edik képben, már előre betöltik az (n+1)-ediket. Ez a „load on demand” lényege, előre betöltjük azokat a részeket, amiknek még nem kell látszaniuk, de a felhasználó *valószínűleg* meg fog tekinteni. Ezeket általában valamilyen rejtett elemben tároljuk addig.
Itt sem szabad túlzásokba esnünk, egy cikk esetén ne töltsünk be előre 10 oldalt, elég a következőt.
Az automatikus betöltéshez hasonló az a minta (inkább fejlesztési szempont) is, melynek lényege, hogy az oldal (Ajax-al felturbózott) betöltésekor, a fontos dolgokat töltsük be először, a kevésbé fontosakat pedig később.

Automatikus frissítés

A Gmail példáján át a legkönnyebben elképzelhető. Ha a Gmail-t nyitva hagyjuk a böngészőnkben, akkor nem marad úgy statikusan abban az állapotában, ahogy hagytuk, hanem ha új email érkezett, azt jelzi. Bizonyos időközönként érdemes lekérdezni a szervertől, hogy az adott dologban (chat, fórum, időjárás előrejelzés, stb.) történt-e változás.

Felugró ablakok

Nem a klasszikus felugró ablakokról van szó, hanem ugyanazon az oldalon jelenlevő felugró dologról, mely lehet ablak, de akár egy buborék is, mely a dokumentum fölött jelenik meg. Kényelmi funkciókat tehetünk így programunkba, például egy web-könyvesbolt esetén a borító kis kép föle véve a kurzort megjelentethetjük a teljes borítót.

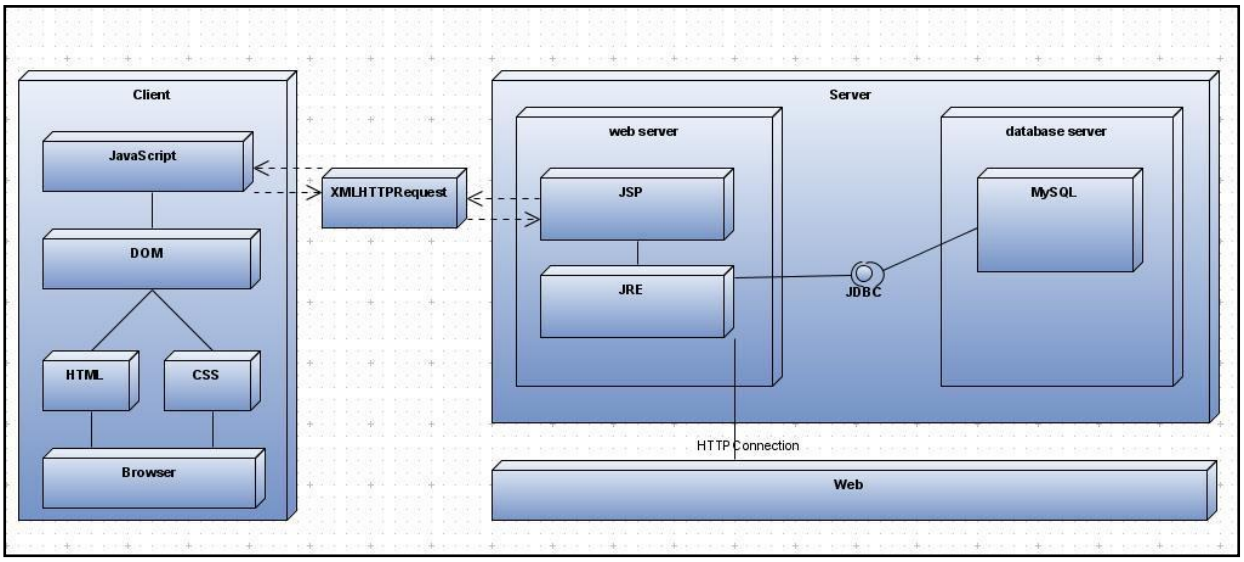
Web 2.0 példaalkalmazás

Az alkalmazás témája:

A web oldal egy nyílt közösségi oldal, a téma zenekarok, ezek albumai, dalai, valamit a zenekarok kategorizálása. Zenekar bevitelekor fontos, hogy megfelelő címkéket adjunk hozzájuk, így segítve a kategorizálást és a későbbi keresést. Hogy a zenekari oldalak minél tartalmasabbak legyenek az alkalmazás más oldalakról nyert információkat is beillesz, valamint minden zenekarnak van saját fórumrésze.

Fejlesztéshez használt eszközök:

Kliens oldalon támogatott böngésző az Internet Explorer és a Firefox. Szerver oldali technológia a Java Server Pages , a konténer Apache Tomcat, az adatbáziskezelő MySQL



A weboldal felépítése és funkciói

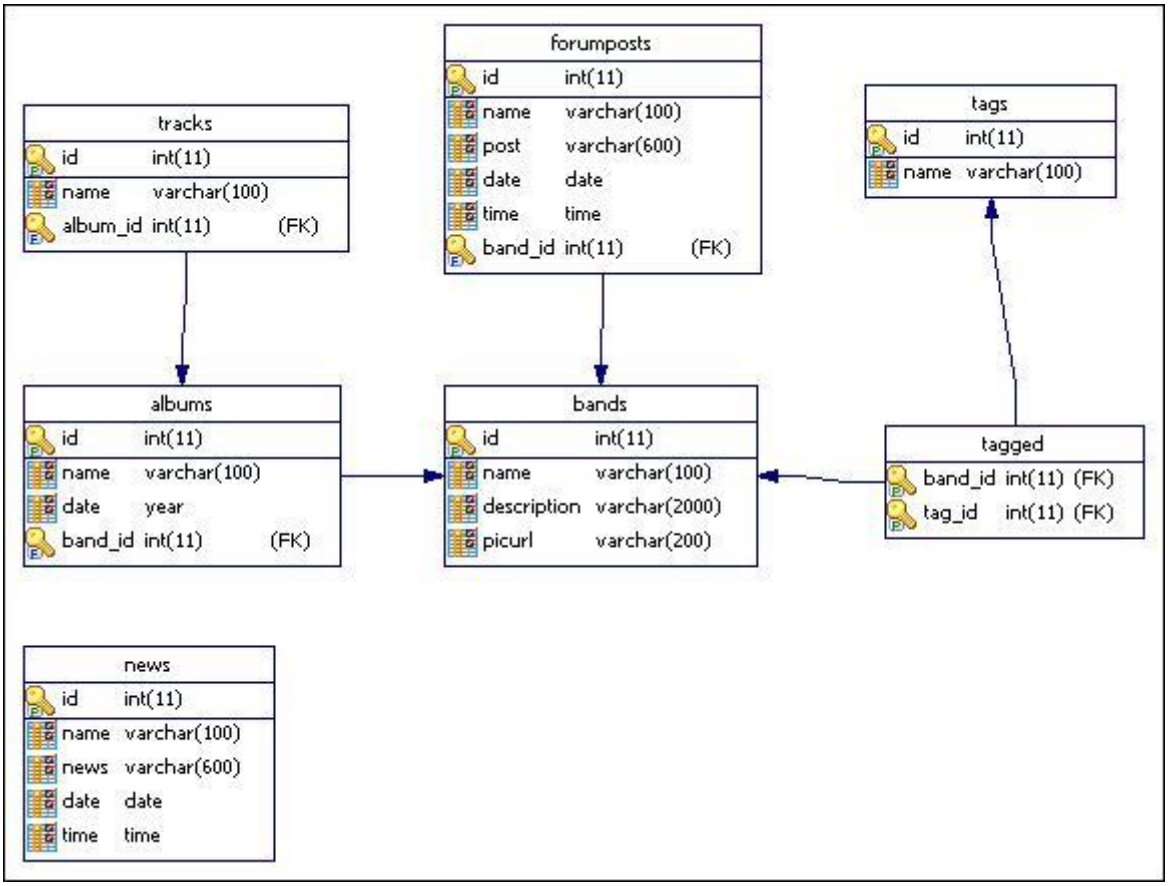
Ebben a részben felhasználói szemszögből ismertetném a webalkalmazást.
Az oldalon baloldalt található egy oldalsáv, ahonnan a főfunkciókat bármikor elérhetjük. Az oldal nagyobbik részét pedig a főablak képezi, melyben alapállapotban az oldal hírei jelennek meg. Az oldalsávon a keresőmezőkön kívül érhetjük el a „címkefelhőt", valamint a „zenekar hozzáadás" funkciót is.
A keresőmezők általános működése a következő. A beírt szöveget tartalmazó nevű entitást (címkét, dalt, albumot, zenekart) keres az adatbázisban, majd az eredményül kapott neveket megjeleníti. A beviteli mező alatt egy menüben megjelennek az ajánlások, melyek egyikére kattintva az megjelenik a főablakban.
A keresőmezők a következők: címke keresése, album keresése, zenekar keresése, dal keresése. Címke keresésekor szimplán a címke neve jelenik meg, zenekarnál szintén. Albumnál zárójelben szerepel a zenekar a neve és a kiadás éve is, dalnál pedig a zenekar neve és az album, amelyen található. A keresőmezők alatt található egy speciális keresőmező, mely nem ajánl eredményeket. Ide egy (adatbázisban nem szereplő) zenekar nevét beírva, majd a „Get" gombot megnyomva, a főablakba beilleszti a zenekarhoz (később ismertetett módon) megtalált lejátszókat.
A címke felhőt behozva megjelennek az adatbázisban található címkék nevei változó betűméretettel. A betűméret attól függ, hogy mennyi zenekar van címkézve az adatbázisban azzal a címkével. A címkére kattintva megjelenik a címke lap.
A címke lapra az címkeresőn keresztül is eljuthatunk. Itt láthatjuk azon zenekarok felsorolását, melyek ezzel a címkével el vannak látva (a zenekar nevére kattintva bejön a zenekar lap), és innen érhetjük el a címke láncolást is.
Címke láncoláskor előbb meg kell adnunk a legördülő menüben hogy „és" avagy „vagy" kapcsolatban szeretnénk láncolni őket. Ezután a már ismert ajánlós módszerrel keressük meg a láncolni kívánt címkét, majd kattintsunk rá. Ekkor láncolja a két címkét, az oldalon megjelenik a logikai összefüggés (pl. american and black) és azok az előadók, akikre ez igaz. Bármeddig láncolhatjuk a címkéket.
Bár album-ajánlásnál az album neve, dal-ajánlásnál a dal neve, zenekar ajánlásnál a zenekar neve jelenik meg a keresőmezők alatt, mindhárom esetben a zenekar-lapra jutunk a linkre kattintva.
A zenekar-lap két fő részből áll. A főablakban a zenekar információk mellett megjelentik egy fórum, mely az adott zenekarhoz tartozó hozzászólásokat tartalmazza. Hozzászóláshoz be kell

gépelnünk a nevünket és a hozzászólni kívánt szöveget. Ekkor, valamint félpercenként automatikusan, a fórum frissül, azaz frissítgetés nélkül is automatikusan látjuk, ha új hozzászólás érkezik. Ha a zenekar-lap tartózkodunk, akkor a baloldali menüben megjelenik egy gomb, amivel elrejthetjük, avagy éppen láthatóvá tehetjük a fórumot.
Az zenekar információk sok mindenből tevődnek össze. A kép az adatbázisban tárolt URL-ről töltődik be, a rövid leírás szintén. Ezután három lejátszó következik, amiket legtöbb esetben nem azonnal látunk, helyettük csak a beöltést jelző animáció látszik, amíg egyenként be nem töltődnek. Az első lejátszó a zenekar Myspace-es oldaláról ágyazzuk be, a másodikat a Last.fm zeneadatbázisból. A harmadik egy videó-lejátszó a Youtube-ról, mely a Last.fm-en leghallgatottabb dalhoz keres videóklippet, koncertet. Ezután az adatbázisban levő albumok felsorolása következik, dalcímekkel együtt, majd a öt „hasonló zenekar" felsorolása. Ezután pedig a zenekarhoz tartozó címkék, melyből egyikre kattintva megjelenik a címke-lap.
Szintén a zenekar lapon tudunk a zenekarhoz albumot vagy címkét adni. Címke hozzáadása értelemszerű, albumnál meg kell adnunk a kiadás dátumát, valamint a számcímeket sortöréssel elválasztva egymástól.

A webalkamazás megtervezése, előkészítése

MySQL, adatbázisséma

Az adatbázis az alábbi táblákból áll.



Mint látjuk középpontban a bands tábla áll. Egy zenekarhoz tartozhat több album, egy albumhoz pedig több dal. A címkézettséget egy külön táblában tároljuk, mivel egy zenekarhoz több címke is tartozhat és fordítva. Az összes fórumhozzászólást egy táblában tároljuk (tehát nem zenekaronként), a híreket pedig teljesen külön.
Mivel JSP technológiát használunk így le kell töltenünk a MySQL Connectior/J-t a http://dev.mysql.com/downloads/connector/j/ weboldalról. A letöltött *Source and Binaries* archívumból a nekünk a *mysql-connector-java-x.x.x-bin.jar*-ra van szükségünk, melyet majd el kell helyezni a webalkalmazás megfelelő library könyvtárába (lásd később). Ezzel a szokásos JDBC kapcsolat fog rendelkezésünkre állni.

Érdemes adatbázisunkat időnként lementeni egy *.sql* fileba, mely legyen többször lefuttatható. Ezt megírhatjuk magunk is, de érdemes *a mysqldump.exe*-t használni a következő módon: *mysqldump adatbázis_név -u root > filenév.sql.* Így egész adatbázisunk „scriptelődik" egy fileba, melyet bármikor újra betöltethetünk az alábbi módon: *mysql.exe –u root,* majd a konzolba gépeljük be: *use adatbázisnév*, ezután *source filenév.sql*.

Apache Tomcat telepítése és beállítása:

A programfejlesztéshez az Apache Tomcat 5.5.25-ös web konténert használtam.
A http://tomcat.apache.org/ -ról töltsük le ezen verzió CORE disztribúcióját, mely mind Windows, mind Linux alatt futtatható. Továbbiakban a Windowson való beállítást/futtatást ismertetem.

A Tomcat a *bin* könyvtárban levő *catalina.bat* –al indítható el. Ha már be van állítva gépünkön a „*java_home*" környezeti változó, és egy megfelelő JDK-ra mutat, akkor a konténer máris futtatható, azonban mégis ajánlatos ezt kézzel beírnunk. Legegyszerűbb mód erre ezen *.bat* fájl szerkesztése.

Az „@echo off" sor után vigyük be a következőt:

```
set JAVA_HOME=c:\Program Files\Java\jdk1.6.0_02\
```

Természetesen, a megfelelő elérési útra átírva. Fontos, hogy JDK-ra kell hogy mutasson és nem JRE-re. Ezután a program indítása „*catalina start*", leállítása a *„calatina stop"* paranccsal már rendben le fog zajlani. A konténer globális beállításait a *conf* könyvtár *web.xml* és *server.xml* fájlja tartalmazza, ennek alapbeállításai általában megfelelőek.

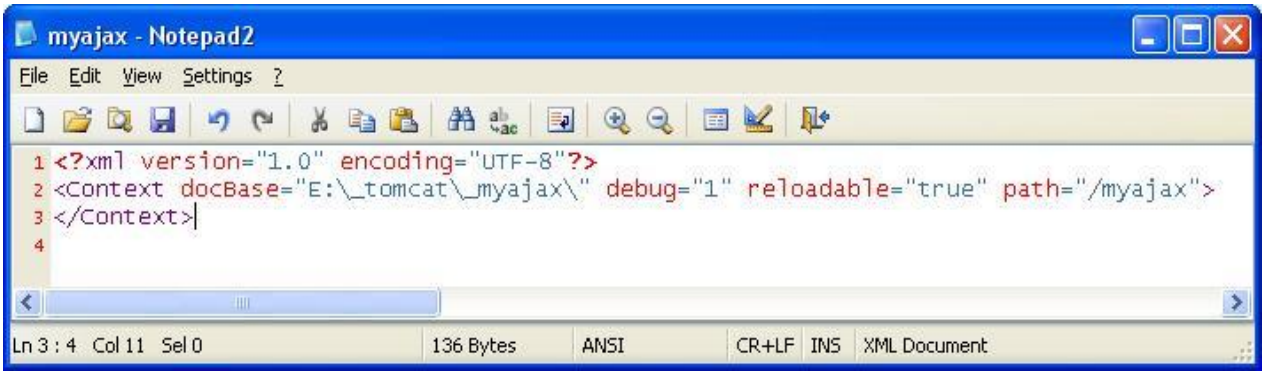
A web alkalmazás szerkezete:

Kontextus megadása:

Maga az alkalmazás a fájlrendszerben bárhol elhelyezkedhet, kétféleképpen. Vagy egy könyvtárban helyezzük el, vagy egy ún. webarchívumban, azaz *.war* (**W**eb **Ar**chive) fájlban (azaz a -későbbiekben leírt – teljes projekt-könyvtárstruktúrát *.zip* formátumban betömörítjük *.war* kiterjesztéssel).

Alapértelmezés szerint a Tomcat a *webapps* könyvtárban keresi az alkalmazásokat, ez módosítható a *conf/server.xml* file-ban. Érdemes azonban minden projekt számára saját kontextust megadni, ekkor sem a Tomcat globális beállításait nem kell módosítanunk, viszont az alkalmazásunknak sem kell a Tomcat könyvtáron belül elhelyezkednie. Esetünkben a

conf\Catalina\localhost könyvtárba kell tennünk egy tetszőleges nevű, *.xml* kiterjesztésű fájlt, melyben alkalmazásonként külön-külön megadhatjuk beállításainkat. A fájl tartalma a következőképp kell kinézzen:



A *context* tag paraméterei a következők:

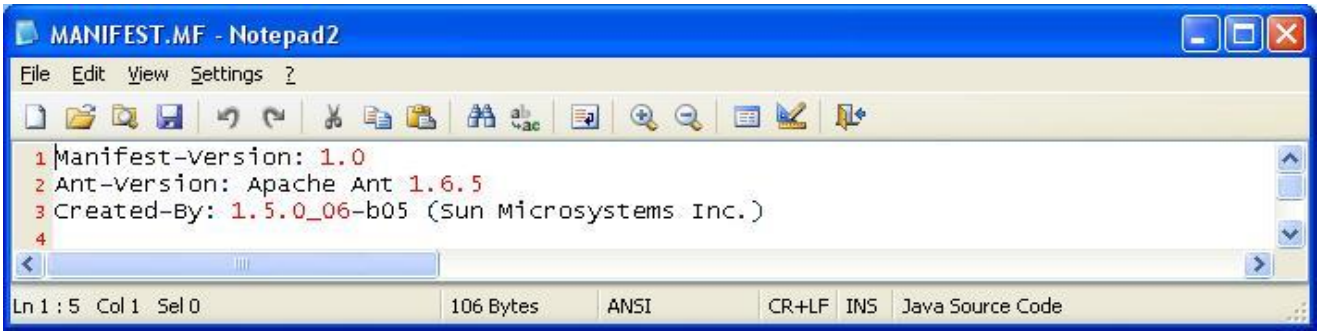
-docBase: megadja az alkalmazás gyökerének tényleges fájlrendszerbeli elérési útját (vagy a *.war* fájl útját)

-path: kontextus útvonal, a böngészőben megadott URL elejét illeszti a megadott útvonalakra, és a megfelelő alkalmazás felé irányítja a kérést (tehát http://localhost/myajax/teszt.jsp URL megadásakor ezen alkalmazás felé irányítja)

-debug: ha állítunk be loggolást végző alkalmazást (pl. log4j), akkor ennek szintjét adhatjuk meg

-reloadable: csak fejlesztéskor használandó „true” állapotban, ekkor, ha az alkalmazás osztályaiban változás történik, a Tomcat újratölti az egész alkalmazást.

Nézzük ezután az alkalmazás könyvtárszerkezetét. A gyökérkönyvtárban és azon belül bármely könyvtárban helyezhetők el fájlok, azonban van néhány speciális könyvtár. Az egyik ilyen a META-INF, melyre akkor van szükség, ha az előzőekben említett webarchívumként tároljuk (az akkor már általában végleges) alkalmazásunk. A benne található manifest.mf fájl meta információkat tárol az archívumról.



A másik, az igazán fontos könyvtár a WEB-INF. Három legfontosabb eleme a következők:

1) az ún. „deployment descriptor” , azaz a *web.xml* file (később bővebben)
2) a *classes* könyvtár, mely a lefordított java osztályainkat, bean-einket, stb. tárolja
3) a *lib* könyvtár, mely a library-ket, azaz a *.jar* file-okat tartalmazza

Ez a szerkezet azért fontos, mert egyrészt egységes struktúrát ad az alkalmazásnak (és az összes ezen úton megírt alkalmazásnak), másrészt ehhez a könyvtárhoz a kliens (direkt http kérés vagy egyéb úton) semmiképpen nem fér hozzá, tehát e könyvtár tartalmát a web-konténer nem adja át direkt úton, így védve alkalmazásunk.

A web.xml felépítése:

Servlet-ek beállításai

```
<servlet>
  <servlet-name>example</servlet-name>
  <servlet-class>servlets.Example</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name> example </servlet-name>
  <url-pattern>/peldaprogram/*</url-pattern>
</servlet-mapping>
```

A <servlet-name> a servlet hivatkozási nevét adja meg, a <servlet-class> az osztályt, melyben a servletet definiáltuk.
A <servlet-mapping> elemen belül (a servlet előbb megadott nevével hivatkozva) megadhatunk egy (vagy több) URL mintát, ami alapján a web-konténer eldönti hogy egy hivatkozott URL meghívása esetén melyik serverhez forduljon. Például ha a (http://localhost:8080/myajax/example/test.tst esetén az Example servlet-hez kerül az irányítás)
A lefordított servleteket a WEB-INF/classes mappában kell elhelyeznünk.

JSP fájl esetén a <servlet-class> elem helyett a <jsp-file> elemet kell megadnunk, benne a jsp fájl (gyökérhez viszonyítva relatív) elérési útját megadni. A JSP fájlokat a servlet-ekkel ellentétben bárhol elhelyezhetjünk.

További paraméterek megadható a servlet elemen belül.
Servlet inicializációs paraméterek megadása (melyet a szerver osztályon belül lekérdezhetünk):

```
<init-param>
   <param-name>color</param-name>
   <param-value>blue</param-value>
</init-param>
```

Szinten servlet elemen belül megadható a <load-on-startup> elem.

```
<load-on-startup>1</load-on-startup>
```

Pozitív (egész) szám megadása esetén a servlet automatikusan elindul a Tomcat indításakor/inicializálásakor (ha több servletben is adtunk meg ilyen tagot, akkor növekvő sorrendben indítja őket).

Biztonsági szempontból ajánlatos letiltani az ún. *directory listing*-et, azaz hogyha az URLben csak egy könyvtár van megadva, akkor ne listázza ki a tartalmát.

```
<init-param>
        <param-name>listings</param-name>
        <param-value>false</param-value>
    </init-param>
```

Egyéb elemek a web.xml-ben:

A *web.xml*-ben megadhatunk ún. welcome-file-t, vagy welcome-file listát. Ez a fájl fog betöltődni, ha nem adunk meg teljes URL-t, azaz általában a átirányít a főoldalra.

```
<welcome-file>index.jsp</welcome-file>
```

Ekkora http://localhost:8080/myajax esetén automatikusan az *index.jsp* fájlt tölti be.
Lista megadásának módja: <welcome-file-list> elemen belül soroljunk fel <welcome-file> elemeket.

Alapértelmezett hibajelentő oldalak megadása:

```
<error-page>
    <error-code>404</error-code>
    <location>/errors/filenotfound.html</location>
</error-page>
```

A web-konténer a http kérés által kiváltott hibát („http response code”)egyezteti felsorolt <error code>-okkal és az a megadott oldalt tölti be.

Taglib-ek megadása:

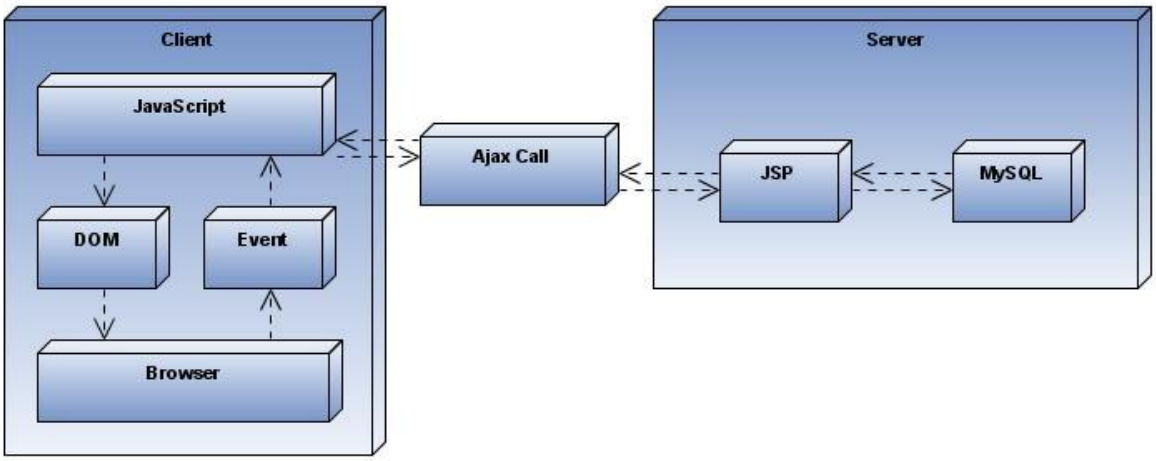
JSLT, azaz JSP Tag Library-k megadásához egy <taglib> elemen belül,mind a <taglib-uri>, mind a <taglib-location> elembe írjuk be a taglib relatív elérési útját. Például:

```
<taglib>
  <taglib-uri>/WEB-INF/struts-bean-el.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-bean-el.tld</taglib-location>
</taglib>
```

Megadhatjuk az alkalmazás nevét és leírását a <display-name> és <description> elemekkel. Ebben a fájlban is megadhatjuk a (fentebb már említett) kontextust, azonban ha már a *conf/catalina/localhost*-ban beállítottuk, akkor itt felesleges. Érdemes úgy eljárni hogyha az alkalmazásunk a *tomcat/webapps*-ban található akkor a *web.xml*-ben adjuk meg a kontextust (így minden, az alkalmazásra vonatkozó beállítás egy helyen lesz található), ha viszont máshol tároljuk akkor a *conf/catalina/localhost*-ban érdemes megadni.

SPA és egyéb sajátosságok

Mivel a weboldalt „Single Page Application”-ként szeretnénk megírni, ezért minden úgy kell megoldanunk, hogy sohase töltse újra az oldalt, valamint mivel nem keretes az oldal, ezért minden új dolgot Ajax-al kell beillesztenünk.
Az eseményeket az oldalsávból és a főablakból is kiválthatjuk, hatásukra általában a főablakban jelenik meg egy új oldal. Érdemes úgy megírnunk a főablakban megjelenő oldalakat, hogy (megfelelő paraméterekkel meghívva őket) önmagukban is működjenek, ez megkönnyíti a tesztelést, másrészt komponensszerűen tudjuk fejleszteni az alkalmazást.
Tehát minden oldal működőképes magában, és ezt a már megírt oldalt illesztjük be a főablakba, bizonyos események hatására. Mivel mindent Ajax-al illesztünk be, így minden eseménynek meg kell hívnia egy JavaScript függvényt, amely meghív egy **általános** Ajax hívást hivatkozva egy szerveroldali JSP-re, és ezt újfent JavaScriptet használva kell beillesztenünk.



Könyvtár és fájlstruktúrára vonatkozóan érdemes következő módszert alkalmaznunk.
Az alkalmazás gyökerében (ahol a WEB-INF könyvtár is található) legyenek a .jsp file-aink, valamint a következő könyvtárak:

- css (a CSS file-okat tartalmazza)
- img (a képeket tartalmazza)
- js (a JavaScript fileokat tartalmazza)
- sql (az SQL scripteket, adatbázis backup-okat tartalmazza)
- src (a Java forrásfileokat tartalmazza)

Bár a HTML-be beágyazva is szerepelhetnek CSS és JavaScript definíciók, utasítások, érdemes ezeket külön file-okban tárolni, így áttekinthetőbb, és így létrehozhatunk külön verziókat is. Valamilyen szinten persze mindenképp szerepelni fognak JavaScript utasítások a HTML elemek eseménykezelőiben, de érdeme is csak függvényhívásokat tenni.

Az így megírt külön file-okra elég a webalkalmazásban egyszer hivatkoznunk.
CSS-re az alábbi módon:

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

A JavaScript-re pedig:

```
<script type="text/javascript" src="js/ajax.js"></script>
```

A webalkalmazás implementálása

A keret JavaScript elkészítése

Először is írjuk meg az Ajax hívást végrehajtó JavaScript kódot.
Ezt általánosan úgy szokták megírni, hogy az a dinamikusan kapott eredményt egy ún. callback függvénynek adják át, így egy ilyen Ajax hívás specifikációja az alábbi:

```
function getAjax(link,callback)
```

Mivel nekünk majdnem mindig annyira van szükségünk, hogy a kapott eredményt beillesszük egy *div* elembe, így az általános függvényünk specifikációja a következőképp néz ki:

```
function getAjax(link,div_id)
```

Két féle Ajax hívás lehetséges, éppúgy ahogy HTTP hívás is kettő van, GET és POST.
A két metódus nagyban különbözik, és másra valók.

A GET féle paraméter átadási metódus így néz ki például:

```
http://mysite.com/page.jsp?paramater1=value1&paramter2=value2
```

A paramétereket az URLben adjuk át. Előnye hogy így „linkelhető" lesz az oldal ezen állapota, hiszen az állapotot meghatározó paraméterek az URL-ben vannak. Hátránya ugyanez, bizonyos esetekben (pl. jelszó) nem engedhetjük, meg hogy az átadott paraméter látszódjon az URLben. A GET főleg adat lekérdezésre való, nem pedig változtatásra (pl. fórumhozzászólás)
A POST metódussal küldött adat nem jelenik meg az URL-ben, így biztonságosabb. Mint látjuk, itt nem is lenne értelmezhető a GET-nél írt linkelhetőség, hiszen itt nem egy URL-hívással van dolgunk. A POST így főleg (bár lekérdezhetünk is vele) adattovábbításra való. Mindenképpen ezt használjuk, ha az interakció valamilyen változást idéz elő a szerveren (pl. új sor egy adatbázis táblában), mivel a böngészők egy esetleges frissítéskor POST esetében figyelmeztetik a felhasználót, hogy az adat újra el lesz küldve, GET-nél nem.

Mivel, mint látjuk mindkettőre szükségünk lesz, így mindkét általános metódust implementálnunk kell. Nézzük először a GET implementációját.

A GET metódus:

```
1  function getAjax(link,div_id)
2    {
3   var xmlHttp;
4      try
5      { // Firefox, Opera 8.0+, Safari
6      xmlHttp=new XMLHttpRequest();
7      }
8    catch (e)
9      { // Internet Explorer
10     try
11       {
12       xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
13       }
14     catch (e)
15       {
16         try
17         {
18         xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
19         }
20       catch (e)
21         {
22         alert("Your browser does not support AJAX!");
23         return false;
24         }
25       }
26     }
27     xmlHttp.onreadystatechange=function()
28       {
29       if(xmlHttp.readyState==4)
30         {
31         var str =xmlHttp.responseText;
32           if (document.getElementById(div_id)==null) ;
33           else document.getElementById(div_id).innerHTML=str;
34         }
35       }
36     xmlHttp.open("GET", link ,true);
37
38     xmlHttp.send(null);
39   }
```

A 3-tól 26-odik sorig mindössze létrehozzuk az XMLHttpRequest objektumot, a böngészőnek megfelelően, ha egyiket se tudjuk, akkor hibaüzenetet dob fel a böngésző.
A következő sorok annak az eseménynek szólnak, ha megkapjuk a válaszszöveget. A 27-es sorban található *onreadystatechange* eseménykezelő mindig kiváltódik, ha az kérés állapota megváltozik. Ezen állapotkódok közül a 4-es a „betöltődött”, ezért ennek meglétét vizsgáljuk a 29-es sorban. Ezután a 33-as sorban beillesztjük a 31-ik sorban válaszul kapott szöveget. Ha nem szöveget, hanem XML-t várunk, akkor nem *responeText*-et, hanem *responseXML*-t kell írnunk.

A 36 és 38-ik sor még mindig a hívás része, a 36-ik sorban adjuk meg a hívás metódusát (itt GET), az URL-t, amit meg hívunk, a *true*-val pedig jelezzük, hogy a hívást aszinkron végezze el a böngésző, majd a 38-ik sorban a *send* metódussal indítjuk a hívást.

A POST metódus:

```
1  function postAjax(link,data,div_id)
2    {
3   var xmlHttp;
4      try
5      {  // Firefox, Opera 8.0+, Safari
6      xmlHttp=new XMLHttpRequest();
7      }
8    catch (e)
9      {  // Internet Explorer
10     try
11       {
12       xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
13       }
14     catch (e)
15       {
16       try
17         {
18         xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
19         }
20       catch (e)
21         {
22         alert("Your browser does not support AJAX!");
23         return false;
24         }
25       }
26     }
27
28     xmlHttp.onreadystatechange=function()
29       {
30       if(xmlHttp.readyState==4)
31         {
32         var str =xmlHttp.responseText;
33         document.getElementById(div_id).innerHTML=str;
34         }
35       }
36
37     xmlHttp.open("POST", link ,true);
38     xmlHttp.setRequestHeader("Content-type",
39         "application/x-www-form-urlencoded; charset=UTF-8");
40     xmlHttp.setRequestHeader("Content-length", data.length);
41     xmlHttp.send("data="+data);
42   }
```

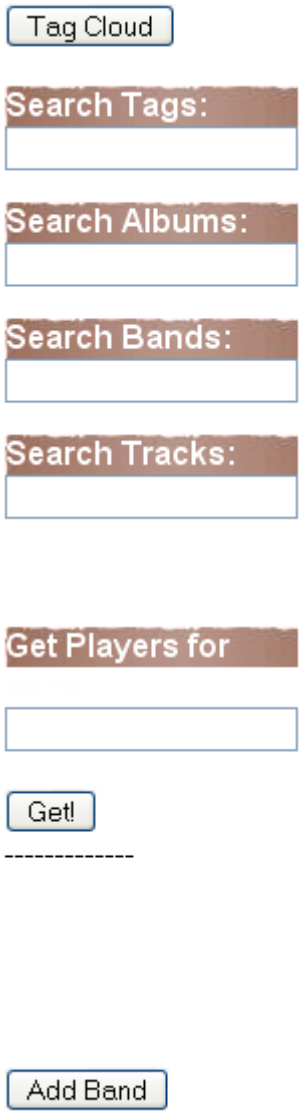
A POST metódus nagyban hasonlít az előzőre, lényegi különbség csak a 36-38 sorokban van, valamint megjelenik egy plusz paraméter, mégpedig az elküldendő adat (hiszen itt nem az URL-ben fog szerepelni az adat). Az *open* metódust nem GET, hanem POST paraméterrel kell meghívnunk, valamint be kell állítani tulajdonságot a kérés fejrészében. Az egyik a kérésben levő tartalom típusa és karakterkészlete, valamint ajánlott a tartalom hosszát is beállítatnunk. Ami még eltérés, a *send* metódust nem *null* paraméterrel hívjuk meg, hanem itt adjuk át az elküldendő adatot. Fontos, ha több adatot szeretnénk egy POST hívással elküldeni, akkor ezt érdemes valamilyen elválasztó-karakterrel megtenni, majd a szerver oldalon feldarabolni.

Az oldalsáv elkészítése

Az ablakot két fő részre osztjuk, egy oldalsávra, és egy főablakra, valamint számba vesszük a főablak egy részét időnként eltakaró fórumrészt. Ezt megtehetjük a következő struktúrával:

```
<div class="leftContent">…</div>
<div id="main_parent" class="rightContent">
     <div id="main_window" class="mainleft_finv">…</div>
     <div id="forum_window" class="mainright_finv">… </div>
</div>
```

Az oldalsáv megjelenítését befolyásoló CSS definíciók a „*leftContent*” és a „*rightContent*”.




```
.leftcontent {
     position:          absolute;
     padding:           6px;
     top:               0;
     bottom:            0;
     left:              0;
     width:             170px;
     height:            100%;
     overflow:          hidden;   }
.rightcontent {
     border-left:       1px solid #cccccc;
     padding:           10px;
     position:          absolute;
     top:               0;
     left:              170px;
     right:             0;
     bottom:            0;
     height:            100%;
     overflow:          scroll;
     width:             85%  }
```

A *top* és *left* attribútumokkal fixáljuk az oldalsáv valamint a főablak elhelyezkedését, az oldalsáv mindig 170 pixel széles, a főablak azonban kitölti az egész további teret. A két *overflow* attribútummal azt adjuk meg, hogy a főablak scrollozható legyen, a menü azonban ne, valamint definiálunk egy bal oldali szegélycsíkot. Mivel a főablak CSS szerkezete attól függ, hogy a fórum éppen látszik-e, avagy nem, ezt a fórumnál fogjuk kitárgyalni.
Az oldalsávon levő elemek közül először nézzük meg a keresőmezőket.

A keresőmezők:

Négyféle keresőmezőnk van, a címke, album, zenekar és dalkereső. Működésük gyakorlatilag ugyanaz, mindössze abban különböznek, hogy melyik táblán dolgoznak, valamint az ajánlatokat más-más *onclick* tulajdonsággal adják vissza. Egyéb eltérés, hogy dal és album keresésnél nem csak az entitás nevét adja vissza, hanem ahhoz tartozó egyéb információkat is.
HTML-beli felépítésük a következő:

```
<form id="tags_form">
<div class="bbg">Search Tags: </div> <input type="text"
onkeydown="timer=setTimeout('getSugg(\'tags\')',1000)"
name="what" />
<div id="tags" class="invis"></div>
</form>
```

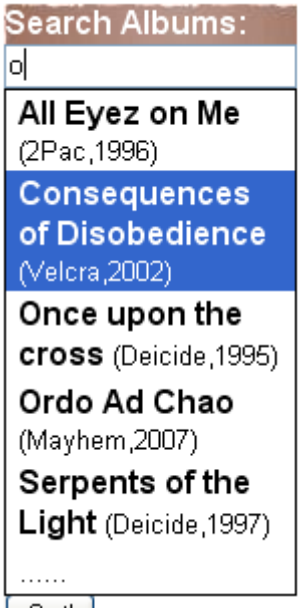
Ez a címke-kereső mező, a többinél a kiemelt helyeken *albums, bands*, és *tracks* szerepel.
A *getSugg* függvény a *getAjax* függvény egy módosítása, mellyel általánosan kérdezhetünk le keresőmező függően eredményeket (azt hogy mit kell visszaadni, azt az egyetlen paraméteréből tudja).
Az *XMLHttpRequest* példányosítása előtt előállítja a meghívandó linket. A keresendő ajánlatokhoz a begépelt mintát a következőképen olvassa ki.

```
function getSugg(table) {
...
var form = (table+"_form");
var value = (document.getElementById(form).what.value);
...
}
```

A `document.getElementById` metódusal direkt módon elérjük az adott id-jű elemet (azt hogy melyik form, azt a paraméterből tudjuk).
Ebből összeállítunk egy ilyen linket:
sugg.jsp?like=*érték*&search=*tábla*&nocache=*véletlenszám*
Mielőtt megnéznénk a visszatérő érték JavaScript beli feldolgozását (mely szinten ebben a függvényben van definiálva), nézzük meg mit csinál a szerveroldali *sugg.jsp*.

A kapott *search* és *like* paramétertől függően lekérdez az adott táblából legfeljebb öt elemet:
`select * from table where name like '%like%' order by name limit 5;`
A kapott ResultSet-et a megadott táblának megfelelően dolgozza fel, azonban minden létrehozott *div* elem, melyek sortöréssel vannak elválasztva egymástól, ugyanolyan CSS osztályú (amit dinamikusan változtatunk az *onmouseover, onmouseout* eseménykezelők segítségével, ez idézi elő az elem kijelölését, ha rávisszük a kurzort). Amiben különböznek, az a *div* elemben található szöveg, valamint a *div* elem *onclick* eseménykezelője, mely akkor aktiválódik, ha rákattintunk az elemre. Minden esetben elrejtjük az ajánlás hatására megjelent *div* elemet, és a címke esetet kivéve láthatóvá tesszük a fórumot (lásd később), címke esetén elrejtjük. A fórum letiltását, avagy engedélyezését fontos mindig lekezelnünk, hiszen az oldalsáv funkciói mindig elérhetők, függetlenül attól, hogy a főablakban mi látható. Igaz még minden esetre, hogy (mint a lekérdezésben is látjuk) maximum öt elemet íratunk ki, ha ötnél több lenne a kiíratandó, akkor egy „…” felirat jelzi a felhasználónak, hogy pontosítsa a kifejezést. További eltérések:

Címke esetén a címke neve szerepel a szövegben, az *onclick* hatására pedig meghívjuk a *getBandsFromTag* JavaScript függvényt, amellyel a címke-lapra jutunk.



Album esetében az album neve, kiadási éve és a hozzá tartozó zenekar neve jelenik meg a szövegben, az *onclick* hatására meghívjuk a *getBand* JavaScript függvényt, azaz a zenekar lapot, az adott zenekarral.

Zenekar esetén a zenekar neve szerepel a szövegben, kattintásra szintén a zenekar-lap töltődik be.

Dal esetén a dalcím, előadó, és az album (amelyen szerepel) kerül a szövegre, kattintáskor szintén a zenekar-lap töltődik be.

Térjünk vissza a *getSugg* JavaScript metódushoz. Az ő dolga innentől annyi, hogy a kapott HTML formázott szöveget beillessze a szövegmező alá, az eddig láthatatlan `<div id="tags" class="invis"></div>` -be, a következő módon.

```
document.getElementById(table).innerHTML=xmlHttp.responseText;
```

Ugyanitt kezeljük le, hogy mi történik, ha a felhasználó nem gépel be semmit (azaz üres a szövegmező), ekkor elrejtjük a szövegmezőt (ismét az *invis* CSS osztályt kapja), egyébként (azaz ha van begépelt adat) a *search_suggest* osztályt, a következő módon: `document.getElementById(table).className = 'search_suggest'.`

A keresőmezőben levő eredmények valamelyikére rákattintva meghívódik az előzőekben ismertetett JavaScript függvények valamelyike, melyek mindössze a *getAjax* függvény segítségével beillesztik a főablakba a keresőmezőhöz tartozó lapot, a kiválasztott eredménnyel paraméterezve.

A keresőmezőkön kívül található további elemek az oldalsávon mind a *getAjax* függvényt használják a beszúrásra.

Címke felhő

A címkefelhőt a *getCloud.jsp* Ajax-os beillesztésével kapjuk. Lényege, hogy a címkék akkora méretben legyenek, amilyen hányadban szerepelnek az adatbázisban.
Kell tehát egy olyan lekérdezés, ami visszaad két oszlopot, a címke nevét, és az előfordulási darabszámát.

```
SELECT name, COUNT(tagged.band_id) AS num FROM tagged,tags
WHERE tagged.tag_id = tags.id GROUP BY tagged.tag_id"
```

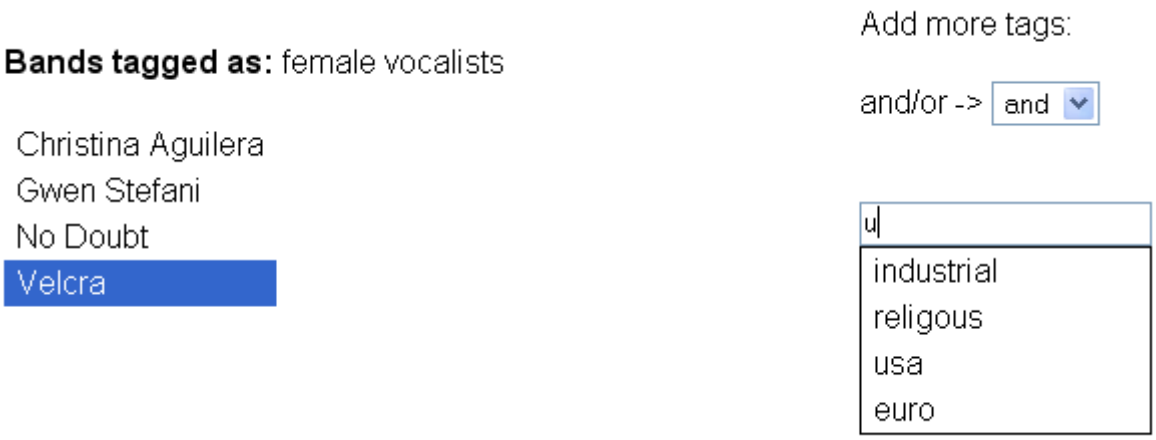
Az eredményt tároljuk le mindenképpen, mivel a számsorozat maximumára és minimumára szüksége van a következő algoritmusnak, valamint egy határra, hogy mekkora legyen a legnagyobb és a legkisebb betűméret.

```
for (int i=0; i<v.size();i++) {
int size =  min_font_size +
((v.elementAt(i).getId() - min))*(max_font_size - min_font_size)/(max-min);
out.println(printTag(v.elementAt(i).getTagName(), size));  }
```

A printTag függvény feladat annyi, hogy formázza, és megfelelő eseménykezelőkkel látja el a címke nevét.

Címke lap

Mint az előzőekben kifejtettem, rendkívül fontos, hogy sok és megfelelő címkét adjunk a zenekarokhoz, hiszen ezek képezik azon lekérdezések gerincét, mely alapján összetett összefüggésekkel kereshetünk a zenekarok közt.
A címke-keresőmezőből vagy zenekarlapon egy címkére kattintva jutunk el erre a lapra, melyen elsőként csak erre a címkére szűrt zenekarok listája található. Ehhez mindössze le kell kérdezni azon zenekarok nevét, amelyek el vannak látva vele.



Emellett található az a funkció, amivel láncolhatjuk a címkéket, felül egy combobox, amivel kiválaszthatjuk, hogy „és” vagy „vagy” kapcsolatot szeretnénk, alatta pedig egy kereső mező (mely szintén a címkék között keres, az előzőekben megszokott módon). A keresőmező nem a szokásos *getSugg* JavaScript metódust hívja meg, hanem a „*moreTags*” metódust (mely a *moreTags.jsp*-t kérdezi le és illeszti be Ajax-al). Az itt található hívás azért különbözik, mert a *getSugg* általánosan lett megírva, valamint az oldalsávon manipulál, ezért itt nem használhatjuk.
Mivel a láncolás logikai összekötőjét (és/vagy) a láncolás elején adhatjuk meg, ezért különböző esetként kell kezelnünk azt, amikor még csak egy címkénk van, és amikor már több, láncolva (ekkor már nem változtathatjuk meg az összekötőt). Ezért a címke lap egy címke esetén a „*getBandsFromTag*” több esetén pedig a „*getBandsFromTags*”. Bár nagyon hasonlóak, lényegi különbségek vannak köztük. Míg a *getBandsFromTag* egy paramétert vár, az egyetlen címkét (amire kattintottunk), addig a *getBandsFromTags* bonyolultabb. Egyrészt nem egy címkét vár, hanem többet, valamint még egy plusz paramétert, azt összekötő logikai jelet. Így, a szervoldal számára egyértelmű, milyen összefüggést kell lekérdezni.

Ha például „amerikai vagy ázsiai” összefüggést eredményét szeretnénk megtudni az adatbázisból, a lekérdezésnek is hasonlónak kell lennie. „olyan zenekar, ami amerikaival van címkéve VAGY olyan zenekar ami ázsiaival van címkézve.”
Több címkére nézve az algoritmus a következő (ahol *v*, egy (a címkék neveit tartalmazó) vektor, az *andor*, pedig egy sztring, a paraméterül átvett logikai összekötő):

```
String query="select name from bands where id in";
query+=" (select band_id from tagged where tag_id in (select
id from tags where name = '"+v.elementAt(0)+"')) ";
     for (int i=1;i<v.size();i++) {
          query += (" "+andor+" id in ");
          query += " (select band_id from tagged where tag_id
          in (select id from tags where name =
          '"+v.elementAt(i)+"')) ";
               }
query+="ORDER BY name";
```

Tehát összeállítunk egy azonosító halmazt, mely azon zenekarok azonosítóját tartalmazza, amire igaz a feltétel, majd ezekhez zenekarnevet rendelünk. Az első címkére vonatkozó lekérdezés nincs a ciklusban, mivel N címke közé N-1 összekötő jel kell.

Zenekar lap

A zenekar kar betöltését a *getBand* JavaScript függvény végzi, mely egyetlen paramétert vár, a zenekar nevét. Ugyanekkor megtörténik a fórum betöltése is, melyet szintén ez a függvény hajt végre, de a fórum láthatóságát azaz esemény váltja ki, amellyel eljutottunk ide. (például a címke lapon kattintunk egy zenekar nevére, ekkor az ottani *onclick* eseménykezelő elindítja a *getBand* függvényt, és az *enableForum* függvényt)

Maga függvény:

```
1 function getBand(bname)  {
2   bname = bname.replace(/ /g,"+");
3   getAjax('getBand.jsp?band='+bname,'main_window');
4   getAjax('getPosts.jsp?band='+bname,'forum_window');
5   getAjaxXMLSimilar(bname,'simil');
6   getAjax('myspaceplayer.jsp?band='+bname,'player_place');
7   getAjax('lastfmplayer.jsp?band='+bname,'player_place2');
8   getAjax('hitvideo.jsp?band='+bname,'player_place3');
9   timer=setTimeout('refreshForum();',30000);
10 }
```

Mint látjuk, elég gyakran használjuk a megírt *getAjax* függvényt, melynek csak a betöltendő linket és a célelemet kell megadnunk.
Az első sorra azért van szükség, mert az URL-ekben nem szerepelhet szóköz. Viszont a függvényt általában úgy hívjuk meg, hogy egy HTML elemből beolvassuk a zenekar nevet, melyek nyilván szóközzel szerepelnek. Ezeket át kell alakítanunk, szóköz helyére pluszjelnek kell kerülnie, melyet a JavaScript beépített *replace* metódusával végre is hajthatunk, reguláris kifejezések segítségével (*/ /g* jelentése a szóköz minden előfordulása).
Ezután betöltjük a zenekar információkat a főablakba, majd fórumot, a hasonló előadók listáját, és a lejátszókat, valamint elindítjuk a fórumot folyamatos frissítő függvényt. (Ezek ismertetését lásd lejjebb.) A zenekar információk megjelenését követően már olvashatjuk is az információkat azalatt, míg a többi elem (fórum, lejátszók) betöltődnek.

Zenekar információk betöltése:

Ez a legösszetettebb lap a webalkalamzásban, a *getBand.jsp* szolgáltatja. Mint a keresőmezőknél említettük, erre a lapra jutunk el album, vagy dal kiválasztásánál is, az ott levő eseménykezelők is a *getBand* függvényt hívják meg, zenekarnévvel.

Lekérdezünk mindent a zenekar táblából, majd megfelelően formázva megjelenítjük a zenekar nevét, a képet, a rövid leírást. Ezután kitesszük a lejátszóknak szánt *div* elemeket, és beléjük teszünk egy betöltést jelző képet.

Következik az albumok és dalcímek megjelenítése. Ehhez albumonként kell bejárnunk a dalcímek tábláját, így egymáshoz rendelődnek. Létrehozunk minden albumhoz egy *Vector*-t, benne a számcímekkel.
Kiíratásra az Apache fejlesztésű DisplayTag könyvtárat használjuk, mely jól példázza, mennyivel nagyobb erő van (és ezzel együtt sajnos erőforrás igény) abban, ha szerveroldalon rakjuk össze a HTML oldalt és nem kliens oldalon.
Az adott JSP oldal elején meg kell adnunk a tag-könyvtárat:

```
<%@ taglib uri="http://displaytag.sf.net" prefix="display" %>
```

Használatához még a JSP kódban át kell adnunk a vektort a globális tárolóba:
`request.setAttribute( ("album") , songs );` Azaz album névvel eltároltuk a songs vektort.

Ezután a JSP kódon kívül, JSP tagként hívjuk meg a DisplayTag-et:

```
<display:table name="album" class="table" >
 <display:column property="track" />
 <display:setProperty name="basic.show.header" value="false"/>
</display:table>
```

Már csak annyi a dolgunk, hogy a CSS definícióink közt legyen *tr.odd* és a *tr.even*, ugyanis ezeket fogják hivatkozni a létrejövő *table* elem sorai.

Legvégül a zenekarhoz tartozó címkék jelennek meg a szokásos módon.
Fontos, hogy ezek csak azok az elemek, amiket a *getBand.jsp* szolgáltat, ebbe még beillesztődik a fórum, és a mashup-ok, azaz a lejátszók.

Fórum

A fórum a főablakban, a zenekar információk mellett található. Mindenképpen ki van kapcsolva, ha nem a zenekar lapon tartózkodunk. Ha ott vagyunk, akkor pedig elrejthető egy gomb megnyomásával. 

A ki és bekapcsolást az *disableForum*, és az *enableForum* függvényekkel hajthatjuk végre:

```
function enableForum(){
document.getElementById("isForum").className="vis";
document.getElementById("forum_window").className="mainright_fv";
document.getElementById("main_window").className="mainleft_fv";
}

function disableForum(){
document.getElementById("isForum").className="invis";
document.getElementById("main_window").className='mainleft_finv';
document.getElementById("forum_window").className="mainright_finv";
}
```

A bekapcsolt fórum elrejtését/megjelenítését a *resize* függvénnyel tesszük meg:

```
function resize() {

if(document.getElementById("forum_window").className == 'mainright_fv') {
document.getElementById("isForum").innerHTML="Show Forum";
document.getElementById("main_window").className='mainleft_finv';
document.getElementById("forum_window").className='mainright_finv';
}
else {
document.getElementById("isForum").innerHTML="Hide Forum";
document.getElementById("main_window").className='mainleft_fv';
document.getElementById("forum_window").className='mainright_fv';
}
}
```

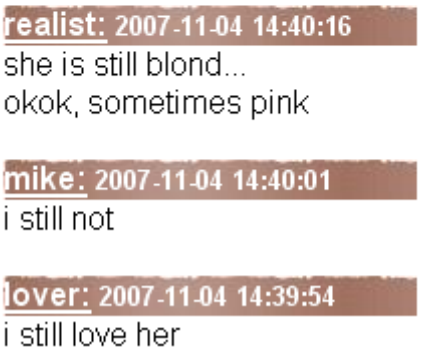
A láthatóságot az elemek CSS osztályainak változtatásával állítjuk és a két eset eléggé hasonló. Az *isForum* elem egy gomb, ennek vagy a láthatóságát, vagy a feliratát állítjuk.

Ha a fórum látható, akkor a zenekar infók és a fórumrész CSS beállításai:

```
.mainleft_fv{
     float: left;
     width: 68%;}
.mainright_fv{
     float: right;
     width: 28%;}
```

Ha a fórum nem látható:

```
.mainleft_finv{
     float: left;
     width: 100%;}
.mainright_finv{
     visibility:hidden;}
```



Térjünk rá magára a fórum elem tartalmára, melyet a *getPosts.jsp*-ből szerzünk be.
A zenekarhoz tartozó hozzászólásokat időrendi sorrendben kérdezzük le, majd formázva a kiírjuk. A hozzászólások fölött lesz található egy név és egy hozzászólás szövegmező, valamint az elküldő gomb.
Ami fontosabb az a fórum működése. Hozzászólás beküldésekor újratöltődik a fórum, valamint félpercenként is, így frissítgetés nélkül is látjuk, ha valaki hozzászól.

A hozzászólás az *newPost* függvényünkkel adódik át a szervernek a *postAjax* metódust használva:

```
function newPost(){
var name = document.getElementById("nameField").value;
var band = document.getElementById("band_id").innerHTML;
var post = document.getElementById("postfield").value;
postAjax('makePost.jsp',name+"||"+band+"|||"+post,'isPostForum');}
```

A *makePost.jsp* átveszi ezen összefűzött paramétereket, majd az aktuális dátummal és idővel beteszi az adatbázisba a hozzászólást. A fórumfrissítést, a *refreshForum* függvényt az egyszerűség kedvéért a *postAjax* metódusban futtatjuk le (ellenőrizzük, hogy ha a fórum hívta meg a függvényt, csak akkor). A *getBand* függvény végén láthattuk az alábbi sort:

```
timer=setTimeout('refreshForum();',30000);
```

Ez az a JavaScript mechanizmus, amely addig idő után hajtja végre a megfelelő utasítást.
A *refreshForum* működése mindössze annyi, hogy *getAjax*-al ismét lekérdezi a zenekar fórumát, majd ugyanúgy végrehajtja a fenti konstrukciót, így folyamatosan frissülni fog a fórum.

Lejátszók beillesztése:

Itt jelentkezik először a Same Origin Policy által okozott probléma. Eddig ugyanis mindig saját adatbázisunkból dolgoztunk, most viszont más oldalak szolgáltatásait vennénk igénybe. A Myspace-es lejátszót nem tudjuk lekérdezni a *getAjax* metódusunkkal, ugyanis a kliens böngésző hibaüzenetet adna és nem működne. Mivel csak a saját szerverünk felé tudunk kommunikálni, ezért a szerverünket *átjáró*nak kell használni a myspace.com felé való kommunikáció során.

Másik probléma, hogy nem tudjuk a zenekar Myspace -es oldalának a címét, és egyáltalán nem biztos, hogy www.myspace.com/zenekarnév lesz a cím. Ezért célszerű a Google-re hagyatkozni, azaz lekérdezzük tőle a „zenekarnév Myspace" kombinációt és az első linket tekintenünk helyes címnek.

Még mindig akadhat problémánk, ugyanis a megtalált oldal alakja kétféle lehet:

- www.myspace.com/valaminév
- **profile.myspace**.com/index.cfm?fuseaction=user.view**profile**&friendID=123087153

Erre ügyelnünk kell.

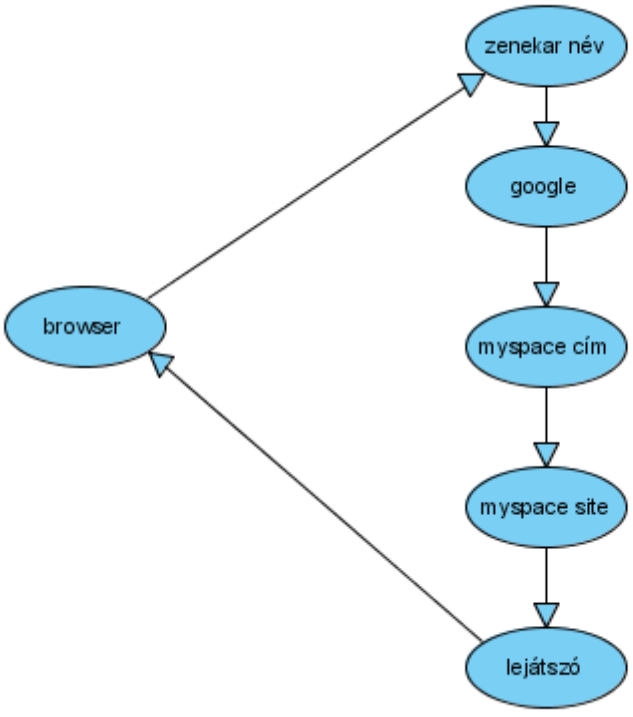
Írnunk kell tehát egy olyan függvényt, amelynek inputja mindössze egy zenekarnév, kimenete pedig a zenekar Myspace-es lejátszója. Mivel a probléma összetettebb, mint az eddigiek, érdemes egy külön Java osztályt létrehoznunk, amit majd hivatkozunk a JSP-ből.

Ahhoz, hogy Java-ból kifelé kommunikáljunk HTTP protokollon keresztül a következő konstrukcióra van szükségünk, URL-ben itt épp a Google-től kérdezzük le a Myspace oldal címét.

```
HttpURLConnection conn = null;
URL url = new URL ("http://www.google.com/search?q=" +band+"+myspace");
conn = (HttpURLConnection)url.openConnection();
conn.setRequestProperty("User-agent","Mozilla/4.0");
InputStream page = conn.getInputStream()
```

Így meg kapjuk a Google eredményt, ebből vágjuk ki az első címet, ügyelve a fent említett kétféle alakra. Ezután használjuk a fenti módszert ismét a kapott linkre, ekkor már kezünkben lesz az egész Myspace-es oldal HTML tartalma. Ebből vágjuk ki a lejátszót, amit az `<OBJECT id="mp3player"` és a `</OBJECT>` sztringek határolnak.
Az eredményünkben, ami már a lejátszó teljes HTML kódja (hivatkozása), hajtsuk még végre a következő utasítást, hogy letiltsuk az automatikus lejátszást:

```
lejátszó = lejátszó.replace("&a=0","&a=1");
```



Hasonló módon kérhetjük le a Last.FM-es lejátszót. Különbség hogy nem kell a Google-t segítségül hívni a zenekar oldalának megtalálásához, mivel az mindig a http://www.last.fm/music/zenekarnév oldalon található. Ezt beolvassuk az előzőekben leírt módszerrel, majd kivágjuk a * * és * * által határolt részt.

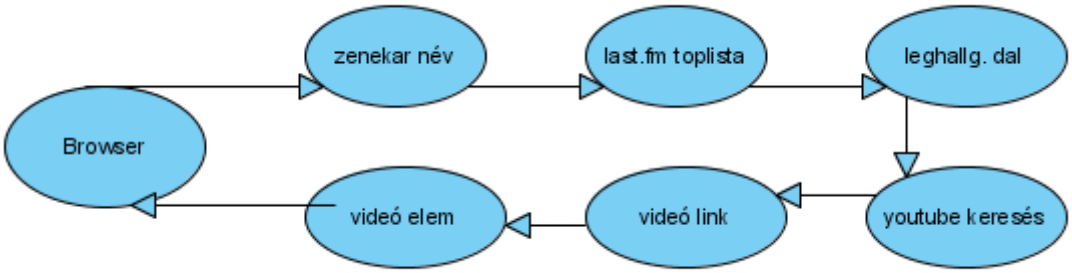
Mivel ezek a módszerek elvárják, hogy legyen lejátszó az igényelt oldalon (ami persze nem biztos, hogy így van) ezért ügyelnünk kell arra, hogyha nincs akkor üres sztringet adjunk vissza, így az alkalmazásban szimplán annyi történik, hogy nem jelenik meg a lejátszó.

A létrehozott osztályokat következőképpen hívjuk meg JSP-ből (melyet a *getAjax* JavaScript függvénnyel máris használhatunk):

```
<%
ajax.getMyspace player = new ajax.getMyspace();
out.println(player.getMyspacePlayer(request.getParameter("band")));
%>
```

Webszolgáltatások használata

A két zenelejátszó alatt található egy videó lejátszó is, mely első ránézésre szimplán a Youtube.com-ről tölt be egy videót azonban egy kicsit komplikáltabban működik.
A Last.fm webszolgáltatásából lekérdezzük a *leghallgatottabb dal* címét, majd erre keresünk rá a Youtube webszolgálgatásán.



Ugyanúgy a szerveren keresztül vagyunk kénytelenek kommunikálni, valamint mivel XML-ben kapjuk a választ, érdemes valamilyen XML feldolgozót használnunk, pl. JDOM.

A leghallgatottabb dal megkeresése:

Nyittassuk meg a http://ws.audioscrobbler.com/1.0/artist/*zenekarnév*/toptracks.xml linket az átjárónkkal. A kapott (sok elemből álló) XML-ből a következő érdekel minket:

```
- <mostknowntracks artist="Velcra">
  - <track>
      <name>My Law</name>
      <mbid/>
      <reach>854</reach>
      <url>http://www.last.fm/music/Velcra/_/My+Law</url>
    </track>
  - <track>
```

Ezt az elemet JDOM-ot használva a következőképpen érhetjük el:

```
Document d = new SAXBuilder().build(toptrackslist);
String hitSong = d.getRootElement().getChild("track")
                  .getChild("name").getText();
```

Így már a zenekar nevén kívül birtokunkban van a leghallgatottabb dal címe is. Ehhez keresünk videót YouTube-on, amit a következő linken tehetünk meg:

http://gdata.youtube.com/feeds/videos?alt=rss&vq=zenekarnév+dalcím

A kapott vaskos válasz XML-ből a következő elememkre van szükségünk:

```
String hitVideoName = (d.getRootElement().getChild("channel")
                      .getChild("item").getChild("title").getText());
String hitVideoLink = d.getRootElement().getChild("channel")
                      .getChild("item").getChild("link").getText());
```

Hasonló előadók

Lássunk példát a kliens oldali XML feldolgozásra is, a hasonló zenekarokat, melyet szintén webszolgáltatásból érünk el. A *getBand* függvényben meghívott *getAjaxXMLSimilar* függvény a *getAjax* egy módosítása, mely tartalmazza a kliens oldali XML feldolgozást.
Az elkért link:
http://ws.audioscrobbler.com/1.0/artist/zenekarnév/similar.xml
A szükséges elemek pedig:

```
– <similarartists artist="Velcra" streamable="1" picture="http://userserve-ak.last.fm/serve/1
  – <artist>
      <name>The Cyan Velvet Project</name>
      <mbid/>
      <match>6.56</match>
      <url>http://www.last.fm/music/The+Cyan+Velvet+Project</url>
      <image_small>http://userserve-ak.last.fm/serve/50/584642.jpg</image_small>
      <image>http://userserve-ak.last.fm/serve/160/584642.jpg</image>
      <streamable>1</streamable>
    </artist>
  – <artist>
      <name>45 Degree Woman</name>
      <mbid/>
      <match>5.54</match>
      <url>http://www.last.fm/music/45+Degree+Woman</url>
      <image_small>http://userserve-ak.last.fm/serve/50/361257.jpg</image_small>
      <image>http://userserve-ak.last.fm/serve/160/361257.jpg</image>
      <streamable>1</streamable>
    </artist>
  – <artist>
      <name>Suburban Tribe</name>
      <mbid>ef454227-c310-4a03-80d0-79caec4492a5</mbid>
      <match>4.69</match>
      <url>http://www.last.fm/music/Suburban+Tribe</url>
      <image_small>http://userserve-ak.last.fm/serve/50/215946.jpg</image_small>
```

A válasz XML feldolgozása:

A lekérdezett XML rengeteg adatot tartalmaz, nekünk csak a nevekre van szükségünk, abból is csak ötre. Az XML dokumentumokra alkalmazhatjuk a *getElementsByTagName* beépített függvényt, mely egy tömbbe helyezi az összes „*name*” nevű nevű tagok szövegelemeit. Ezekből a nevekből felépítjük a nekünk megfelelő HTML elemeket.

```
xmlHttp.onreadystatechange=function() {
  if(xmlHttp.readyState==4) {
    var xml =xmlHttp.responseXML;
      similars = xml.getElementsByTagName("name");
      var i;
      var similar="<div class=\"bbg\"><b>Similar Artists:
        </b></div><br>";
      for (i=0;i<5;i++) {
      similar+=(similars[i].firstChild.nodeValue);
      similar+="<br>"
      }
      document.getElementById(div_id).innerHTML=similar;
    }
  }
```

Az eredmény:

Similar Artists:

Darkthrone
Immortal
Gorgoroth
Burzum
Emperor

Adatok bevitele

Zenekar beviteléhez az „*add band*” gombra kell kattintani az oldalsávon, ekkora a főablakba betöltődik az *addBand.jsp*. Album vagy címke felvételére a zenekar lapon van lehetőség (*addAlbum.jsp* és *addTag.jsp*), a dalcímek felvétele pedig az albumfeltöltés része. Mindhárom esetre igaz, hogy az adott JSP paraméter nélkül betöltve megjelenít egy form-ot, amit megfelelően ki kell töltve és POST metódussal ugyanennek (!) a JSP-nek elküldve dolgozódik fel. Ehhez mindössze a JSP legelején vizsgálni kell, hogy van egy adott típusú paraméter megadva, vagy nincs.

Az zenekar hozzáadás teljesen világos, az megadott dolgok bekerülnek az adatbázisba, ezután megtaláljuk ott az „üres” zenekart (akinek még nincs albuma és nincsenek címkéi).

Albumnál előbb a megadott adatokkal és a zenekarhoz tartozóan felvesszük az albumot, majd a dalcímek mezőből, a sortöréseknél elválasztva beolvassuk a dalok listáját és azokat is felvesszük az albumhoz tartozóan.

Címke felvitele esetén előbb meg kell győződnünk róla, hogy olyan címke szerepel-e már az adatbázisban, ez egy egyszerű lekérdezés:

```
select 1 from tags where name = címkenév
```

Ha a ResultSet nem üres, akkor van ilyen címke, csak hozzá kell rendelni a zenekarhoz. Ha azonban üres, akkor előbb fel kell vennünk egy ilyen nevű címkét, csak ezután mehet a hozzárendelés.

Hírcsatorna elkészítése:

Az előzőekben ismertetett szerkezetű RSS hírcsatornát készítünk a legutolsó öt hírhez.

Az alap keret, amibe kell illesztenünk a híreket.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0">
<channel>
  <title>W3Schools Home Page</title>
  <link>http://www.w3schools.com</link>
  <description>Free web building tutorials</description>
...
</channel>
</rss>
```

Ide kell beillesztenünk az item elemeket. Az utolsó öt hírből álló ResultSet-et az alábbi módon dolgozzuk fel:

```
while (rs.next ())      {
      idVal = rs.getInt ("id");
      name = rs.getString ("name");
      news = rs.getString ("news");
      time = rs.getTime("time");
      date= rs.getDate("date");

      out.println("<item>");
      out.println("<title>"+name+"</title>");
      out.println("<link>http://scorn.ath.cx:8080</link>");
      out.println("<description>"+news+"</description>");
      out.println("</item>");
            }
```

Így megkapjuk a szabványos RSS formátumot, viszont hogy ne egy JSP legyen a hivatkozott dokumentum, ezért érdekem a következő mapping-et létrehozunk Tomcat-ben, a web.xml-ben.

```
<servlet>
    <servlet-name>rssfeed</servlet-name>
    <jsp-file>/getRSSFeed.jsp</jsp-file>
</servlet>
<servlet-mapping>
    <servlet-name>rssfeed</servlet-name>
    <url-pattern>/rss/*</url-pattern>
</servlet-mapping>
```

Most már bármely webalkamazás/rss/* útvonalra hivatkozva az hírcsatornát kapjuk, érdemes tehát a linkünkben a következőképpen hivatkozni: /rss/rss.xml .

Összefoglalás

A dolgozat elméleti részének fő célja a Web 2.0 és az Ajax technológia ismertetése, melyet úgy érzem sikerült érthetően, de mégis részletesen leírnom, ismertetnem, odafigyelve, hogy a sajátosságok kiemelésére.
A program implementációjának ismertetése a Web 2.0 sajátosságok mentén haladt, azaz csak azon programozási technikákat ismertettem, melyek e szempontból fontosak. A program nem tekinthető késznek, rengeteg egyéb funkciót lehetne belepakolni, és sok felhasználó esetén más implementációs technikák is szükségesek (adatbázis kapcsolat pool, szerveroldali adat cache-elés, regisztráció alapján létrehozott felhozott felhasználók). Ezek implementálástól eltekintettem, azon oknál fogva, hogy nem szükségesek a cél, azaz a Web 2.0 és Ajax ismertetésének eléréséhez.
Megállapíthatjuk, hogy bár ezen alkalmazások célja, hogy asztali alkalmazáshoz hasonló funkciókkal rendelkezzen, a fejlesztése eltérő, mivel a kommunikáció a weben a zajlik, így a nagyobb válaszidő miatt jelentősen módosulnak a tervezési szempontok. Fejlesztős során ráadásul sok olyan dologgal kell megbirkóznunk, mely nem jellemző az asztali fejlesztés során, ilyenek a különböző kliens platformok, a nehéz nyomkövetés. a sokféle technológia egyszerre való használata.
Legvégül vizsgáljuk meg a programot azon szempontok alapján, melyeket az ismert Web 2.0 alkalmazásoknál is felsoroltunk.
A felhasználói felület kihasználja az Ajax nyújtotta lehetőségeket, az alkalmazás sohasem frissíti az egész oldalt, használunk olyan mintákat, mint az automatikus kiegészítés, automatikus frissítés, automatikus betöltés. Mivel a felületet gazdagítjuk más oldalakról származó funkciókkal így az alkalmazást teljes funkcionalitásában csak a weben működik, valamint a web platformként való használatához az hozzájárul, hogy az adatokat a felhasználóknak kell létrehozni, bár azt nem tekinthetik sajátjuknak. A közösségi szellem megvalósulását segíti a fórum, (ez az a pont, amin sokat lendítene a felhasználói profilok implementálás), valamint az elosztott adatlétrehozás.

- közösségi szellem
- mennyire használja platformként a web-et
- adat és felhasználó kapcsolata

Függelék:

Ajax és Flash oldalak összehasonlítása:

Ajax:

-html elemek, css, és scriptek sokasága
-hozzáfér a DOM minden eleméhez
-javascript engedélyezése szükséges hozzá
-kicsi az erőforrás igénye
-interpretált
-egyszerűbb vizuális élmény
-ajax alapú oldal kiegészíthető flash tartalommal
-alkalmas Single Page Application létrehozására
-szerverrel való kommunkáció (XML vagy szöveg)

Flash:

-egy html-be ágyazott objektum
-nem fér hozzá a DOM-hoz
-plugin kell hozzá
-nagy az erőforrás igénye
-fordított
-nagyobb vizuális élmény
-flash alapú oldal nem egészíthető ki ajax tartalommal
-alkalmas Single Page Application létrehozására
-szerverrel való kommunkáció (XML vagy szöveg)

Dot com bubble:

A fogalom jelentése, hogy az internet üzleti lehetőségeinek felismerésekor rengetegen fektettek pénzt az internetes alkalmazásokba, mint üzletágba, néhány nagyon sikeres példán felbuzdulva. Túl sokan remélték azonban, hogy kevés befektetéssel nagy hozamra tesznek szert, és a lufi „kipukkanását” jó néhány webalkalmazás nem érte túl. A megmaradt ismert alkalmazások elemezése, közös ismertetőjegyeinek keresése segítette létrejönni a Web 2.0 foglamat.

Az XML HTTP Request objektum specifikációja:

Metódusok:

abort()	Megszakítja az aktuális kérést.
getAllResponseHeaders()	Sztringként visszaadja az összes http header-t.
getResponseHeader("headernnév")	Vissza headernnév nevű http header-t.
open("method","URL",async,"uname","pswd")	Beállítja a kérés paramétereit. A method lehet „GET”, „POST” vagy „PUT”. Az URL a relatív vagy abszolút elérési cím. Az async egy logikai paraméter, ha igaz, akkor a kérést aszinkron módon kezeljük.
send(tartalmom)	Elküldi a kérést.
setRequestHeader("név","érték")	Http header-t adhatunk meg a kéréshez.

Mezők:

onreadystatechange	Eseménykezelő, mely mindig kiváltódik ha a kérés állapota változik.
readyState	Az objektum állapota 0 = inicializálatlan 1 = betöltés alatt 2 = betöltött 3 = interaaktív 4 = kész
responseText	A válasz szövegként.
responseXML	A válasz XML objektumként.
status	Http státusz szám. (pl. „403” a letiltottra)
statusText	Http státusz szöveg. (pl. „letiltott” a 403-ra)

Köszönetnyilvánítás

Köszönöm témavezetőmnek, Adamkó Attilának, hogy tanácsaival segítette munkámat, valamint Jeszenszky Péternek, hogy felhívta figyelmemet a témára.

Irodalomjegyzék

[1] John Musser, Tim O'Reilly: Web 2.0 Principles and Best Practic**es** **(**O'Reilly Media, Inc., 2007.)

[2] Steve Holzner, John Wiley: Ajax for Dummies (Wiley Puiblishing Inc., 2006.)

[3] Nicholas C. Zakas, Jeremy McPeak, Joe Fawcett: Professional Ajax (Wiley Puiblishing Inc., 2006.)

[4] Justin Gehtland, Ben Galbraith, Dion Almaer: Pragmatic Ajax (The Pragmatic Programmers LLC., 2005.)

[5] Dave Crane, Eric Pascarello, Darren James: Ajax in Action *(*Manning Publications Co., 2006.*)*

[6] Bruce Perry, Ajax Hacks **(**O'Reilly Media, Inc., 2006.)

[7] Danny Goodman, Michael Morrison: JavaScript Bible (Wiley Puiblishing Inc., 2004.)

[8] Tim O'Reilly: What is Web 2.0 (http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html 2005.09.30.)

[9] Web 2.0 Conference (http://www.web2con.com/pub/w/32/speakers.html , 2007.04.22.)

[10] Apache Tomcat 5.5 Servlet/JSP Container, Documentation (http://tomcat.apache.org/tomcat-5.5-doc/index.html ,2007.08.17.)

[11] JavaServer Pages v2.0 Syntax Reference (http://java.sun.com/products/jsp/syntax/2.0/syntaxref20.html , 2007.02.06.)

[12] The XMLHttpRequest Object, W3C (http://www.w3schools.com/xml/xml_http.asp , 2007.05.02.)

[13] MySQL 5.1 Reference Manual (http://dev.mysql.com/doc/refman/5.1/en/index.html , 2007.10.12)

[14] Scriptplayground, Tag Cloud Tutorial (http://www.scriptplayground.com/tutorials/php/Tag-Cloud/ , 2007.11.01)