**Szent István Egyetem**

**Gazdaság- és Társadalomtudományi Kar**

**TATA Kiválósági Központ és Informatikai Intézet**

**Auto-szűrő fejlesztése OLAP jelentések utólagos, offline tovább-feldolgozására**

**Belső konzulens neve, beosztása: Dr. Pitlik László, egyetemi docens**

**Önálló szervezeti egység vezetőjének neve, beosztása: Dr. Kovács Árpád Endre, TKI intézetigazgató**

**Készítette: Bures Tamás**

**Gödöllő**

**2010.**

Tartalomjegyzék

[1. Bevezetés 4](#_Toc273708001)

[1.1. Motiváció 4](#_Toc273708002)

[1.2. Cél 4](#_Toc273708003)

[1.3. Célcsoport 6](#_Toc273708004)

[1.4. Hasznosság 7](#_Toc273708005)

[2. Szakirodalom feldolgozása 8](#_Toc273708006)

[2.1. Ismert best practice megoldások 9](#_Toc273708007)

[2.1.1. FlavorZoom 9](#_Toc273708008)

[2.1.2. PicNet Table Filter 9](#_Toc273708009)

[2.1.3. The Idea Mill 9](#_Toc273708010)

[2.1.4. Noah Hendrix táblázat szűrője 9](#_Toc273708011)

[2.2. OLAP 10](#_Toc273708012)

[2.2.1. Az OLAP fogalma 10](#_Toc273708013)

[2.2.2. Felépítése 10](#_Toc273708014)

[2.3. Megvalósítási eszközök 12](#_Toc273708015)

[2.3.1. Web böngészők 12](#_Toc273708016)

[2.3.2. A HTML alapjai 16](#_Toc273708017)

[2.4. Platformfüggetlenség 17](#_Toc273708018)

[2.5. Megoldás tervezetek értékelése 18](#_Toc273708019)

[2.6. Online kontra offline 20](#_Toc273708020)

[3. Anyag és módszer 22](#_Toc273708021)

[3.1. Adatvagyon bemutatása 22](#_Toc273708022)

[3.2. Végleges problémamegoldás terve 24](#_Toc273708023)

[3.3. Választott technológiák részletesebben 24](#_Toc273708024)

[3.4. Megoldási folyamat 29](#_Toc273708025)

[4. Mérések 39](#_Toc273708026)

[4.1. Használat közben 39](#_Toc273708027)

[4.1.1. Első teszt 40](#_Toc273708028)

[4.1.2. Második teszt 42](#_Toc273708029)

[4.1.3. Harmadik teszt 43](#_Toc273708030)

[4.2. Tesztelési tapasztalatok 44](#_Toc273708031)

[4.3. Második fejlesztési szakasz 45](#_Toc273708032)

[4.3.1. Kisebb-nagyobb-egyenlő funkció 46](#_Toc273708033)

[4.3.2. Gombnyomásra történő keresésindítás 47](#_Toc273708034)

[4.3.3. Találati darabszám 49](#_Toc273708035)

[4.3.4. Egyedi formázhatóság 49](#_Toc273708036)

[4.3.5. Sorszámozás 50](#_Toc273708037)

[4.3.6. Mutat/elrejt funkció 53](#_Toc273708038)

[5. Következtetések 54](#_Toc273708039)

[6. Összefoglalás 55](#_Toc273708040)

[7. Irodalomjegyzék 56](#_Toc273708041)

[8. Ábrajegyzék 59](#_Toc273708042)

[Fogalomtár 61](#_Toc273708043)

[Rövidítések 63](#_Toc273708044)

[Függelékek 64](#_Toc273708045)

# Bevezetés

## Motiváció

Adathalmazokkal és adattáblákkal szinte mindenki dolgozik témakörtől és szakterülettől függetlenül. A legtöbb esetben rendelkezésünkre áll valamilyen táblázatkezelő szoftver (pl.: Microsoft Office, Open Office, stb.), de olykor (például egy weboldal által generált, OLAP jelentés esetén) tovább-feldolgozását praktikus lenne azonnal a böngészőben elvégezni. Mindezt anélkül, hogy a konverziós problémák (számok, szövegek, pénzegységek pontos felismerése, oszlop-sor elcsúszások kikerülése) sokaságát bevállalva az online jelentést át kellene emelni a szokásos (offline) táblázatkezelő környezetbe, illetve magát az OLAP szolgáltatás paraméterezését kellene túlbonyolítani.

A témaválasztás oka rendkívül egyszerű. Előbb, vagy utóbb mindenkinek a kezébe fog kerülni témakörtől függetlenül valamilyen adathalmaz, legyen az akár egy elemzés, vagy pályázat, amivel nyilvánvalóan dolgoznunk kell. A számítástechnika és az Internet fejlődésével párhuzamosan fejlődtek a böngésző programok, így ma már bármilyen számítógépen találunk legalább egy ilyen, alkalmazást operációs rendszertől függetlenül. A böngészők belső feldolgozó motorját és egy kevéske célirányos fejlesztést követően ezek a programok használhatóak lesznek ilyen adathalmazokkal való munkához anélkül, hogy drága szoftvereket vásárolva, majd azok telepítésével, beállításával törődnünk kéne.

## Cél

Az előző pontban feltárt problémákra szeretne a szerző egy olyan (OLAP-szolgáltatásokhoz output-kapszulaként integrálható, illetve önállóan is használható) alkalmazást fejleszteni, melynek a futtatására egy web-böngészőnél többre nincs szükség, hogy ezzel oszlop, illetve sor szinten szűrni, rendezni lehessen egy HTML táblázatot. Mindezt offline, vagyis immár az OLAP-szolgáltatástól elszakadva. Az így kialakítandó szolgáltatás logikáját az ismert táblázatkezelő programokból ismert autó-szűrő megoldásokhoz, illetve a kimutatás-varázslás egyes funkcióihoz hasonlítható a legjobban.

A weben talált (ingyenes) best practice (továbbiakban BP) megoldások gyengéje, hogy a felkínált manipulációs lehetőségek köre túl szűk, a szoros integráció valamilyen OLAP rendszerrel nem megoldható. A fejlesztés tehát e szolgáltatási kör bővítését és a HTML-táblán keresztüli OLAP-integráció biztosítását célozza.

A következő oldalon található 1. táblázat tartalmazza a jelenlegi BP megoldások által nyújtott szolgáltatásokat, összehasonlítva a most készülő rendszer által nyújtott szolgáltatásokkal.

|  |  |  |  |
| --- | --- | --- | --- |
| Szolgáltatás | Saját szűrő | BP | Tervezett verziószám |
| Speciális alkalmazás mellőzése | igen | nem | verzió: 0.0 |
| Tetszőleges karaktersorra való szűrés | igen | igen | verzió: 0.0 |
| Keresett kifejezést nem tartalmazó táblázati sorok elrejtése | igen | igen | verzió: 0.0 |
| Tetszőleges karaktersort nem tartalmazó táblázati sorokon túl minden egyéb sor elrejtése | igen | igen | verzió: 0.0 |
| Több szűrő használata egyidejűleg | igen | igen | verzió: 0.0 |
| Nagyobb, mint lehetőség | igen | igen | verzió: 1.0 |
| Kisebb, mint lehetőség | igen | igen | verzió: 1.0 |
| Nagyobb, vagy egyenlő lehetőség | igen | igen | verzió: 1.0 |
| Kisebb, vagy egyenlő lehetőség | igen | igen | verzió: 1.0 |
| Tartalmaz lehetőség | igen | igen | verzió: 1.0 |
| Oszlopok elrejtése a soroknál ismert minden egyes szűrési funkcionalitást kihasználva | nem | igen | verzió: 2.0 |
| Részösszeg, sorok/oszlopok ki-bekapcsolása | nem | igen | verzió: 2.0 |
| Sor- oszlopösszegek (átlagok, szorzatok, minimum, maximum, szórás, stb.) ki-bekapcsolása | nem | igen | verzió: 3.0 |
| És/vagy kapcsoló kezelése | nem | igen | verzió: 3.0 |
| Oszlop és sor-sorrend módosítása | nem | igen | verzió: 3.0 |
| Könnyű integrálhatóság és testreszabhatóság | igen | nem | verzió 0.0 |
| Sorszámozott lista létrehozása adott oszlop értékeinek alapján | igen | nem | verzió 1.0 |

1. táblázat – a BP jelenlegi tudásának összehasonlítása a fejlesztés alatt álló rendszerrel (saját ábrázolás)

Az 1. táblázat tartalma alapján a BP megoldások olyan szempontokat helyeztek előtérbe, ahol a gyakran használt funkcionalitásokon van a hangsúly. Ezek a funkciók az én esetemben nem voltak mérvadóak és azok a funkcionalitások, amikre nekem lett volna szükségem (szélesebb manipulációs lehetőségek, sorszámozás, harmadik féltől származó program mellőzése), azokkal egyik BP sem rendelkezett (kizárólag az ingyenes megoldások halmazában maradva).

További nagy hátrányuk az általam ismert BP-nek, hogy azok működési köre kizárólag a böngészőben maradva alkalmazhatóak, ráadásul, ha valamilyen olyan funkcióra van szükségem, amit az adott rendszer nem támogat, akkor elő kell vennem valamilyen programot a hiányosságok megkerülésére. Én pontosan ezt nem szerettem volna, tehát úgy próbáltam megteremteni a saját rendszeremet, hogy a folyamat elejétől a legvégéig, ne kelljen valamilyen más programba átmozgatnom az adatokat *(pl.: valamilyen táblázatkezelő programba)* és ott alkalmaznom a változtatásokat. Szerettem volna teljesen a böngészőn belül maradni.

Később szó lesz még olyan funkcionalitásokról, amik az 1. táblázatba nem kerültek bele, mert utólag keletkeztek. Ezekre az igényekre a jelenlegi BP egyáltalán nem ismer és nem is tartalmaz megoldást. Igények:

* gomb alapú keresés
* üres sorok elrejtése
* egyedi vagy körülményes testre szabhatóság
* Excel alapú sorszámozás (sorszam függvény)
* URL-ben történő felparaméterezés – tehát már úgy inicializálódik a szűrő, hogy az adott keresőmezőkben megtalálhatóak a keresési feltételek, melyet az URL-ből érkező adatok alapján állít elő a rendszer
* könnyű integrálhatóság
* paraméterezhetőség (globálisan és egyedileg egyaránt)

## Célcsoport

Lényegében mindenki. Kicsit bővebben minden olyan ember vagy embercsoport, akiknek néha úgy kell dolgozniuk, hogy nincs korlátlanul rendelkezésre álló internetkapcsolat és táblázatkezelő szoftver a számítógépükön, viszont adattáblákkal kell dolgozniuk.

## Hasznosság

Az egyszerű kialakításnak és az offline módnak köszönhetően a felhasználók a nagy adattáblákat is könnyedén, ésszerűen tudják kezelni, szűrni, rendezni, legyen szó egy programozóról, vagy egy projektasszisztensről. A hasznosság az időmegtakarításra vezethető vissza, mely alapját az alkalmazásváltás és a konverziós kényszer elmaradása, illetve a részletes OLAP-paraméterezés helyett utólagos, felhasználóbarát manipulációk lehetősége adja. Hasonlóképpen a felhasználó-barátságot növeli, hogy az OLAP-tábla speciális szintaktikai elemei és a felhasználó környezeti változói között inkompatibilitást (pl.: tizedesjel: pont, vessző) nem kell külön figyelni, illetve a cellaformátumok esetleges zavarai (szám/szöveg konverzió) sem terhelik az adattáblák manipulációját.

# Szakirodalom feldolgozása

* Ismert best practice megoldások
* OLAP
  + **Az OLAP fogalma** – mi az OLAP, mire szolgál
  + **Felépítése** – hogyan épül fel, mi a logikája
* Megvalósítási eszközök
  + Web-böngészők
    - **Általánosságban** – mi a böngésző feladata, általános tudnivaló
    - **Belső működés** – mi alapján kerül megjelenítésre egy weboldal
    - **JavaScript támogatás a böngészőkben** – hogyan kapcsolódik a JavaScript a web-böngészőkhöz
  + **JavaScript értelmezés a böngészőkön belül** – mi alapján fut le a JavaScript kód a böngésző berkein belül
  + **HTML alapjai** – a nyelv leírása, működése, fontosabb tudnivalók
* **Platformfüggetlenség** – miért fontos napjainkban és mit is jelent valójában
* **Megoldás tervezetek értékelése** – milyen megoldás variációk jöttek szóba
* **Online kontra offline** – a téma legfontosabb kérdése, előnyök-hátrányok, általános jellemzése a két módszernek

## Ismert best practice megoldások

Ebben a fejezetben feldolgozásra kerülnek a dolgozat témájával versenyző, jelenleg is „piacon” lévő nagyobb megoldások. Azért nagyobbak, mert ezzel a témával nagyon sokan, sokféleképpen foglalkoztak és a dolgozat terjedelme nem engedi meg, hogy az összeset sorra vegyem.

Általánosságban mindegyikről elmondható, hogy a sorszámozási funkció és a gombra induló keresés egyikben sem található, a további funkciók is csak keveredve oszlanak el a megoldások között.

### FlavorZoom

A FlavorZoom (FlavorZoom, 2010) egyetlen előnye, hogy az adott táblázatban képes keresni szöveges és számszerűsített adatokra, azonban nem adja meg a lehetőségét az oszlop specifikus szűrésnek, illetve nem rendelkezik speciális szűrési lehetőségekkel.

### PicNet Table Filter

A PicNet (PicNet Table Filter Demo, 2010) táblázatszűrője már valamivel fejlettebb a FlavorZoom-nál, de itt is akadnak hiányosságok. Először is a kisebb-nagyobb-egyenlő funkció itt működik, azonban itt szabad szavas keresésre csak olyan szempontból van lehetőség, ha nem töredékekre próbálunk keresni.

### The Idea Mill

A The Idea Mill (The Idea Mill, 2009) saját fejlesztésű szűrője áll talán a legközelebb az enyémhez, már ami az alap funkciókat illeti. Ismeri a kisebb-nagyobb-egyenlő szűréseket a számoszlopokon, lehetőségünk vagy szabad szavas/szótöredékes keresésre, azonban a tagadott keresést nincs (pl.: minden, ami nem Párizs – „!Paris”). Automatikusan számolja a számoszlopok alatt mennyi az adott oszlop összege, lehetőség van egyedileg formázni az adott oszlop megjelenését (az én esetemben erre nem volt szükség).

### Noah Hendrix táblázat szűrője

Hendrix (Hendrix, 2009) szűrője is hasonló a fentebb leírtakkal, bár a The Idea Mill-éhez viszonyítva sokkal butább. Először is, ez a szűrő is csak táblázat szinten képes szűrni, oszlop specifikusan nem. Képes az úgy nevezett zebracsíkozásra (ez azt jelenti, hogy a táblázatban váltakozó színnel jelennek meg a páros és páratlan sorok).

## OLAP

### Az OLAP fogalma

„Online Analytical processing, or OLAP […], is an approach to swiftly answer multi-dimensional analytical queries. OLAP is a part of the broader business intelligence, whicl aslo encompasses relational reporting and data mining. The typical applications of OLAP are in business reporting for sales, marketing, management reporting, […] budgeting, forecasting […] and similar areas […].” (Online analytical processing - Wikipedia, the free encyclopedia, 2003)

Az OLAP alkalmazások olyan speciális alkalmazások, amelyek segítségével feltárhatunk adott adatok közti kapcsolatokat és viszonyokat. Gyakran szerepelnek relációs jelentések (relational-reporting) és adatbányászati technológiák (Data mining - Wikipedia, the free encyclopedia, 2002) alappilléreként. Gyakori felhasználási területei az OLAP technológiának: eladások üzleti jelentései, marketing, vezetőségi jelentések, előrejelzések, pénzügyi jelentések stb.

Ez a dolgozat szempontjából azért fontos, mert a jelenlegi hallgatói dolgozatok azért nem tudnak konkrét témában szakpolitizálni, adott témát demonstrálni, mert az Interneten fellelhető adatok sokasága, az esetek túlnyomó részében nem OLAP technológiára épül, így az adatgyűjtés irracionálisan sok időt venne igénybe.

### Felépítése

„At the core of any OLAP system is the concept of an OLAP cube (also called a ’multidimensional cube’ or a ’hypercube’). It consists of numeric facts called measures which are categorized by dimensions. The cube metadata is typically created from a star schema or snowflake schema of tables in a relational database.” (Online analytical processing - Wikipedia, the free encyclopedia, 2003)

Bármilyen OLAP rendszerről is legyen szó, minden ilyen elemző rendszernek a lelkét az úgynevezett OLAP-kocka (OLAP-cube) biztosítja (további nevek lehetnek még a hiperkocka vagy multi-dimenzionális kocka). Egy ilyen „kocka” általában számszerűsített adatokat tartalmaz (nevezzük egységeknek), amiket úgynevezett dimenziók kategorizálnak. Ez a struktúra legtöbbször egy relációs adatbázis tábláiból építkezik csillag- vagy hópehely topológiában (tartalmazva a szükséges meta-adatokat). Egyszerűbben fogalmazva: feltárja az adatok közti kapcsolatot.

Fontos tudnivaló, hogy szemben a pivot táblával, az OLAP mögött megbújó SQL-parancsok közül – például a crosstab – lehetőséget biztosítanak arra, hogy a cellatartalmak ne csak számszerűsített adatokat tartalmazzanak, hanem szöveget is, ezzel is elősegítve a vizualizálás lehetőségét.

A dolgozatom témája teljes egészében, a MIAU (MIAU: 1998-2010: Magyar Internetes Alkalmazott/Agrár Informatikai Újság, 1988) szerverén megtalálható OLAP modulra épül. Sajnos az OLAP definíciók nem határolják le a pontos határait annak, hogy akkor pontosan mit is nevezünk OLAP rendszernek, hiszen az adatbázis-lekérdezések, hasonlóságelemzések, majd azokból keletkezett jelentések egyaránt részei a modulnak.

Vannak, akik minden adatbázis-lekérdésre és az azokból származó adatokból történő elemzésekre is úgy tekintenek, hogy az már része egy ilyen OLAP-nak. Akadnak olyanok is, akik szerint az adatok begyűjtését és azok szervezése nem tartozik az OLAP-ba, mert számukra az OLAP mint definíció csak a mesterséges intelligencián alapuló adatfeldolgozást és hasonlóságelemzést jelenti.

Én nem kívánok részt venni ebben a definíció-háborúban, én a MIAU szerverén működő OLAP-ot tekintem OLAP rendszernek és a dolgozatom témája is ezen a tényen alapul, a fejlesztések is ennek a kijelentésnek a tükrében valósultak meg.

## Megvalósítási eszközök

### Web böngészők

#### Web böngészőkről általánosságban

„A web browser or Internet browser is a software application for retrieving, presenting, and traversing information resources on the World Wide Web.” (Web browser - Wikipedia, the free encyclopedia, 2001)

A web böngészők tehát olyan alkalmazások gyűjtőneve, amelyek különféle adatok lekérdezésére, azok megjelenítésére illetve adatok továbbítására használunk napjainkban. Minden adat alapvetően a World Wide Webről (Világháló - Wikipédia, 2004), más néven az Internet felől érkezik. Az már egy másik kérdés, hogy a böngészők lokális adatok és fájlok megnyitására, megjelenítésére is alkalmasak.

Minden, az Interneten megtalálható információforrásnak van egy egyedi, saját azonosítója, amit röviden URI[[1]](#footnote-2)-nak nevezünk.

A legismertebb böngészők napjainkban: Microsoft Internet Explorer, Mozilla Firefox, Opera, Google Chrome, Apple Safari.

A bemutatott eszközöknek és technológiáknak további előnyei közé tartozik, hogy ma már elég sok eszköz képes HTML (lásd 2.2.2. A HTML alapjai) tartalmak megjelenítésére és kezelésére *(notebook, mobiltelefon)*, az Internet adta lehetőségek pedig sokfelé elérhetőek a vezetékes-, vezeték nélküli- és mobilhálózatok kialakítása révén.

Az is tény, hogy napjainkban a legtöbb internetes adatbázisokhoz elsődlegesen böngészők segítségével fér hozzá a felhasználó, így minden fejlesztés, ami ezt a folyamatot támogatja, jelentős érdeklődésre tarthat számot. A dolgozat célja egy gyorsan hasznosuló innovációs folyamat katalizálása, mely mögött az üzleti modell elsődlegesen referencia-gazdálkodás reformját felvállaló, NKTH-támogatással (Nemzeti Kutatási és Technológiai Hivatal, 2010) kialakított, MY-X FREE online elemző csomag használata kísérleti szolgáltatásként.

#### Belső működése

Az elsődleges célja a böngészőnek, hogy az információt az Internetről megszerezze, és megjelenítse a felhasználónak. A folyamat mindig azzal kezdődik, hogy a felhasználó megadja az URI-t (például: http://www.szie.hu). Az URI prefixuma elsődlegesen meghatározza a folyamat feldolgozását. A mi esetünkben http-vel kezdődik, így a teljes folyamat a HTTP protokollon (HTTP - Hypertext Transfer Protocol Overview, 2010) fog végighaladni.

A korábban felsorolt böngészők egytől-egyig támogatnak további prefixumokat, például: HTTPS (https - Wikipédia, 2007), FTP (File Transfer Protocol - Wikipédia, 2005) vagy file[[2]](#footnote-3).

Vannak olyan prefixumok is, amiket közvetlenül a böngésző nem tud alkalmazni. Ilyen például a mailto: is. A mailto: prefixummal a böngésző automatikusan átadja a vezérlést az adott operációs rendszeren beállított alapértelmezett levelező alkalmazásnak.

A dolgozat szempontjából ennek nincs releváns értelmezése, de szükségszerű volt a megemlítése, mert a böngészők alapvető működéséhez hozzátartozik.

#### JavaScript támogatottság

„A JavaScript programozási nyelv egy objektumalapú szkript nyelv, amelyet weblapokon elterjedten használnak. Eredetileg Brendan Eich, a Netscape Communications mérnöke fejlesztette ki; […] később „JavaScript” nevet kapott, és szintaxisa közelebb került a Sun Microsystems Java programozási nyelvéhez.” (JavaScript - Wikipédia, 2004)

Először a Netscape cég implementálta a technológiát böngészőjében. Bár a böngészőt már nem fejlesztik tovább, mégis az ő nevükhöz köthető a böngészés és vele együtt a JavaScript megjelenésének fénykora.

A JavaScript-et webes környezetben más néven kliensoldali programozási nyelvnek is hívják, mivel a programkód nem szerveroldalon a szerver által kerül lefuttatásra, hanem a kliens oldalon, tehát a felhasználó által használt böngészőn belül.

Pontosan ez teszi lehetővé más alkalmazásokkal való együttműködést; szebb, látványosabb, funkciódúsabb felhasználói felületek kialakítását.

Természetesen a problémák is erről a tőről fakadnak, mert sajnos a böngészők egyes esetekben, ugyanazt a kódot nem teljesen ugyanúgy futtatják le. Ennek jelenleg az oka az, hogy még mindig nem elérhető olyan szabvány, ami pontosan definiálná a JavaScript értelmezésére vonatkozó szabályokat.

Éppen ezért kezdtek megjelenni a különféle keretrendszerek (framework), melyek fejlesztésekor pontosan az volt a cél, hogy a manapság használt böngészők mindegyikében ugyanúgy jelenjenek meg a JavaScript-ben írt programok, kiegészítők. Tehát már nem a fejlesztő dolga biztosítani a „rugalmasságot” hanem a keretrendszeré, így az ilyen jellegű funkciók a felhasználótól, de még a tervező, programozó előtt is rejtve maradnak.

Összességében, a JavaScript olyan forráskód, melyet a felhasználó szaktudás nélkül is könnyen meg tud tekinteni, módosítani is képes azt. Ebből kiindulva tehát, a tudás elsődlegesen nem védhető, a dolgozat mögötti üzleti modell nem is tesz kísérletet szerzői jogok védelmére, „csak” a PR aspektusokra, vagyis példamutatásra koncentrál.

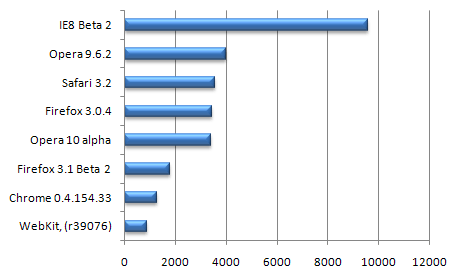
#### JavaScript értelmezés

Körülbelül két évvel ezelőtt (2008. kora ősz) az Apple, Safari névre keresztelt böngészőjét hirdette a világ leggyorsabb böngészőjének, már ami a megjelenítéshez kapcsolódó teljesítményt illeti. Ez a kijelentés az elmúlt háromnegyed-évben, alapjaiban rúgta fel a böngésző piacot.

A cél innentől kezdve nem az volt a fejlesztők számára, hogy biztonságos, használható böngészőket fejlesszenek (bár többnyire azért sikerült ezt az irányvonalat is megtartani), hanem a lényeg, hogy kinek volt képes a böngészője ugyanazt a JavaScript kódot gyorsabban – és természetesen eredményesen vagy eredményesebben – ugyanúgy lefuttatni. Ekkor kezdődött el az úgy nevezett JavaScript benchmarking és egyben háború.

Ekkor született meg a SunSpider (SunSpider JavaScript Benchmark, 2009) (ami a dolgozat írásának idejében a 0.9.1-es verziónál tart (Stachowiak, 2010)) névre hallgató JavaScript benchmark, amely komplexitásával átfogó képet ad a különféle böngészők teljesítményéről. Nem sokkal később más benchmarking programkezdeményezések is megjelentek ezen a „piacon” azonban a SunSpider komplexitásával és úgy mond táborával soha nem tudták felvenni a versenyt.

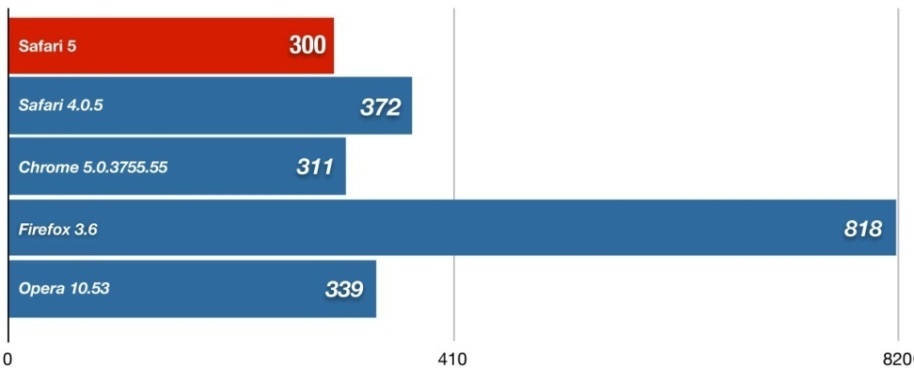
A tavaly decemberi helyzetet mutatja be az 1. ábra.



1. ábra - a böngészők JavaScript feldolgozásának sebessége. Az Y-tengelyen az adott böngésző neve és verziószáma, míg az X-tengelyen a SunSpider benchmark program által mért összpontszám olvasható. A kisebb érték a jobb. (Ifj. Zettner, 2008)

Látható, hogy a versenyt a csupasz WebKit nyerte (ez a HTML motor dolgozik a Google Chrome-ban és az Apple Safari-jában egyaránt), a nagy szoftveróriás böngészője pedig többszörösen is a sor végén kullog (Ifj. Zettner, 2008).

Természetesen a böngészők teljesítményjavulása a mai napig nem fejeződött be. 2010 tavaszán volt a második nagy roham, amikor gyakorlatilag az összes elterjedtebb vagy ismertebb böngésző komolyabb verzióváltást hajtott végre. Szerencsére a verzióváltás a biztonsági és apróbb hibák javítása mellett, a JavaScript feldolgozási sebességre is kiterjedt. Az alábbi ábrán (2. ábra) egy összehasonlító teszt végeredménye látható (Loyola, 2010). A tesztet egy 2,4 GHz-es Intel Core i5 processzorral, 4 GB memóriával felszerelt Macintosh gépen hajtották végre. A teszt program itt is a SunSpider JavaScript benchmark program volt. (A kisebb érték a jobb).



2. ábra - a böngészők JavaScript feldolgozási sebessége 2010 tavaszán. A tesztprogram által adott pontszám látható a képen (a kisebb érték a jobb). (Loyola, 2010)

Pontosan itt jönnek szóba a már korábban említett JavaScript keretrendszerek. Ezekből a rendszerekből is többfélét találhatunk, a legelterjedtebbek listája: jQuery (jQuery: The Write Less, Do More, JavaScript Library, 2009), Prototype, Mootools, Dojo, YUI, Google Web Toolkit, stb.

Az előbb felsoroltak közül mindegyik remekül használható, alapjaiban többé-kevésbé ugyanúgy működnek. A különbségek inkább a részletekben rejlenek, amibe most nem kívánok részletesen belemenni, mert ez a téma egy önálló szakdolgozatot is megérne.

A megjelenítés sebessége az OLAP-bővítmények esetén alapvető kérdés - lévén nem ritkán (sőt, ha a szerveroldali manipulációkat minimálisra szorítjuk) - akkor alapvetően nagy adatmennyiségek kezelését kell biztosítani a felhasználó számára racionális időkeretek között. Ennek azért nagy a jelentősége, mert nem szeretnénk, ha a felhasználó az OLAP kimeneteként létrejövő HTML táblát mégis csak táblázatkalkulációs felületen akarja inkább továbbalakítani. Tehát fontos, hogy mindenképpen a böngészőhöz „láncoljuk” a felhasználót.

### A HTML alapjai

„A HTML (angolul: HyperText Markup Language=hiperszöveges jelölőnyelv) egy leíró nyelv, melyet weboldalak készítéséhez fejlesztettek ki, és mára már internetes szabvánnyá vált a W3C (World Wide Web Consortium) támogatásával.” (HTML - Wikipédia, 2004)

Nagyon sokan elkövetik azt a téves kijelentést, miszerint a HTML is egy programozási nyelv. Nem az. A HTML egy leíró nyelv, melyben adva van a logika és az építőkockák, szimplán csak jól kell használni a megadott logika mentén azokat a bizonyos kockákat.

Számos eszközt kínál strukturált dokumentumok létrehozására. Ilyen eszközök lehetnek például: bekezdések, fejlécek, számozott és számozatlan listák, de nem lehetetlen képek, videók, interaktív anyagok elhelyezése sem.

Minden elemnek, amivel az adott objektumot létre tudjuk hozni, megvan a saját, speciális címkéje (tag), amit a kiválasztott szabványnak megfelelően kell alkalmazni. Minden ilyen objektumnak számos egyéb attribútuma van, melyek alapvetően meghatározzák az adott elem megjelenését, viselkedését.

A JavaScript kódok elhelyezése is itt történhet meg. A tartalmak készítőjének pedig nagyon körültekintőnek kell lennie, mert mára több szabvány (Standards Only (sorted by date) - W3C, 2009) (pl.: HTML 4.01, XHTML 1.0, XHTML 1.1, stb.) is létezik, és itt is igaz, ami a JavaScript-nél, nem minden böngészőben néz ki úgy az adott tartalom. Nem lehetetlen, de nem is könnyű.

A pontos szabvány megválasztása azért fontos, mert elsősorban gondolni kell a jövőre, ami a számítástechnikában kicsit nehézkes téma, de azért nem lehetetlen. Olyat érdemes választani, ami esetleg még 5 év múlva is támogatva lesz.

A szabvány korrekt megválasztásának másik velejárója, hogy minél közelebb mozgunk a szabvány előírásaihoz, akkor annál kevesebb problémánk lesz a böngészőkön belüli megjelenítésbeli problémákkal.

## Platformfüggetlenség

Mi is az a platformfüggetlenség? Rendkívül egyszerű dolog, ám napjainkban egyre fontosabb tényezővé válik az ipari, és/vagy célterületeken. Én, mint fejlesztő, semmilyen jogon nem várhatom el a felhasználótól, hogy csak olyan platformot, eszközt, operációs rendszert használjon, ami az én termékemmel működőképes. Erre a jelenlegi helyzetben a fejlesztőknek három megoldásuk van.

Az egyik ilyen megoldás, hogy ugyan azt a terméket több platformra is elkészítik, így a termék mindenhol ugyan úgy néz ki, ugyan úgy viselkedik és legfőképp a funkcionalitása is az eredeti marad. Körülményes, költséges, viszont minden helyzetben a legjobb megoldás lehet, ha az erőforrásokból nincs hiány.

A második megoldás nem más, mint amikor a fejlesztők csak olyan technológiákat választanak céljuk eléréséhez, ami mindenhol, minden környezetben ugyanazt produkálja, így elég pár egyszerűbb szabályt betartva úgy megalkotni az alkalmazást.

A harmadik és egyben a legelterjedtebb megoldás, hogy az alkalmazást valamilyen webes környezetbe ültetik, tehát tényleg semmire sincsen szüksége a felhasználónak az alkalmazás használatához, csak egy web böngészőre.

## Megoldás tervezetek értékelése

Az előnyök és a hátrányok forrása is ugyanarra vezethető vissza. Pont azért előnyös a HTML és JavaScript párosra építeni az alkalmazást, mert széles körben elterjedt technológián alapul, nincsenek különös igényei (pl.: önálló futtatási környezet telepítése, nagy hardverigény, stb.).

A hátránya pedig az, hogy átgondolatlan tervezés esetén nem lesz teljesen kompatibilis a különféle böngészőverziókkal, valamint a JavaScript sajátosságai miatt az adatkezelés is körülményessé válhat. Ami pedig a legfontosabb, egy böngészőben futó alkalmazástól soha nem várhatunk olyan rugalmasságot és gyorsaságot, mint a kimondottan erre a célra írt alkalmazástól.

Ha nem is tucatnyi, de azért egy-két megoldástervezet (online, vagy offline jellegű feldolgozás, szerver- vagy kliensoldali megvalósítás) született. A sok-sok kritérium és lehetőség közül csupán egy dolog volt csak fix, mégpedig az, hogy az alkalmazásnak el kell futnia a böngészőn belül, mert így megmaradhat a platformfüggetlenség.

Egyértelmű volt számomra, hogy webes környezetbe ültetem az alkalmazásom, viszont itt további sarkalatos kérdések vetődtek fel.

Az egyik ilyen eldöntendő kérdés az volt, hogy az alkalmazás valamilyen online verzióban működjön-e vagy sem. Ez azt jelentené, hogy a kényes transzformációkat, számításokat, rendezéseket nem a felhasználó böngészője végzi, hanem központilag egy erre a célra kialakított weboldal, amit valamilyen szerveroldali programozási nyelv segítségével építettek fel (pl.: Java (Java (programozási nyelv) - Wikipédia, 2004), PHP (PHP - Wikipédia, 2004)).

Az online működés sematikus folyamatábrái:

. ábra – első lépésként a böngésző megkapja az adatokat a web-kiszolgálótól (saját ábrázolás)

. ábra – második lépésként a felhasználó elvégzi, a szükséges műveleteket majd visszaküldi az adatokat a web-kiszolgálónak (saját ábrázolás)

. ábra – a web-kiszolgáló alkalmazza a kért változtatásokat, újra generálja a HTML kódot és visszaküldi a böngészőnek a kész tartalommal (saját ábrázolás)

## Online kontra offline

„The terms „online” and „offline” (also stylized as „on-line” and „off-line”) have specific meanings in regard to computer technology and telecommunications. In general, „online” indicates a state of connectivity, while „offline” indicates a disconnected state. In common usage, „online” often refers to the Internet or the World Wide Web.” (Online and offline - Wikipedia, the free encyclopedia, 2002)

Az online megvalósítás további kérdést vet fel: mi történik akkor, ha esetleg egyszerre több százan, vagy akár több ezren is használni kívánják az alkalmazást? Lassulások, elérhetetlenségek, adatvesztések, hálózati időtúllépések léphetnek fel még akkor is, ha a terhelés a rendszertervben meg lett jelölve, ki lett dolgozva.

A másik szóba jövő megoldás pedig az online ellentettje volt, az offline, tehát kapcsolat nélküli alkalmazás. Ennek a lényege, szemben a másikkal, hogy a transzformációkat, számításokat és rendezéseket nem központilag, hanem egyénileg, tehát a kliens számítógépén végezzük el. További előnyös szempont lehet, hogy az adatfeldolgozáshoz nincs szükség folyamatos internetkapcsolatra, tehát akkor is tudunk vele dolgozni, ha éppen nem rendelkezünk aktív kapcsolattal.

Itt azonban fontos megjegyeznem, hogy valójában nem egy 100 százalékosan kapcsolat nélküli megoldásról van szó, hiszen az adathalmazt továbbra is az internet felől kapja meg a böngésző. A lényeges hangsúly itt inkább azon van, hogy az adathalmazzal való műveletek sokasága kerül át a felhasználó számítógépére, így megkímélve a felesleges terheléstől a webkiszolgálót (6. ábra).

Releváns értelmezés tehát az, hogy kíméljük a központi web kiszolgáló terhelését (főleg nagy felhasználó számnál) és csökkentsük az adatmozgatás mennyiségét, valamint miért ne helyezzük ki a számításokat a felhasználó gépére, ha ez manapság már nem okoz teljesítménybeli problémákat.

Ezeket mérlegelve úgy döntöttem, hogy a kapcsolatnélküli megoldást választom.

6. ábra – a kapcsolat nélküli megoldás. Magára a kapcsolatra csak addig van szükség, amíg az adatok meg nem érkeznek a böngészőbe (saját ábrázolás)

# Anyag és módszer

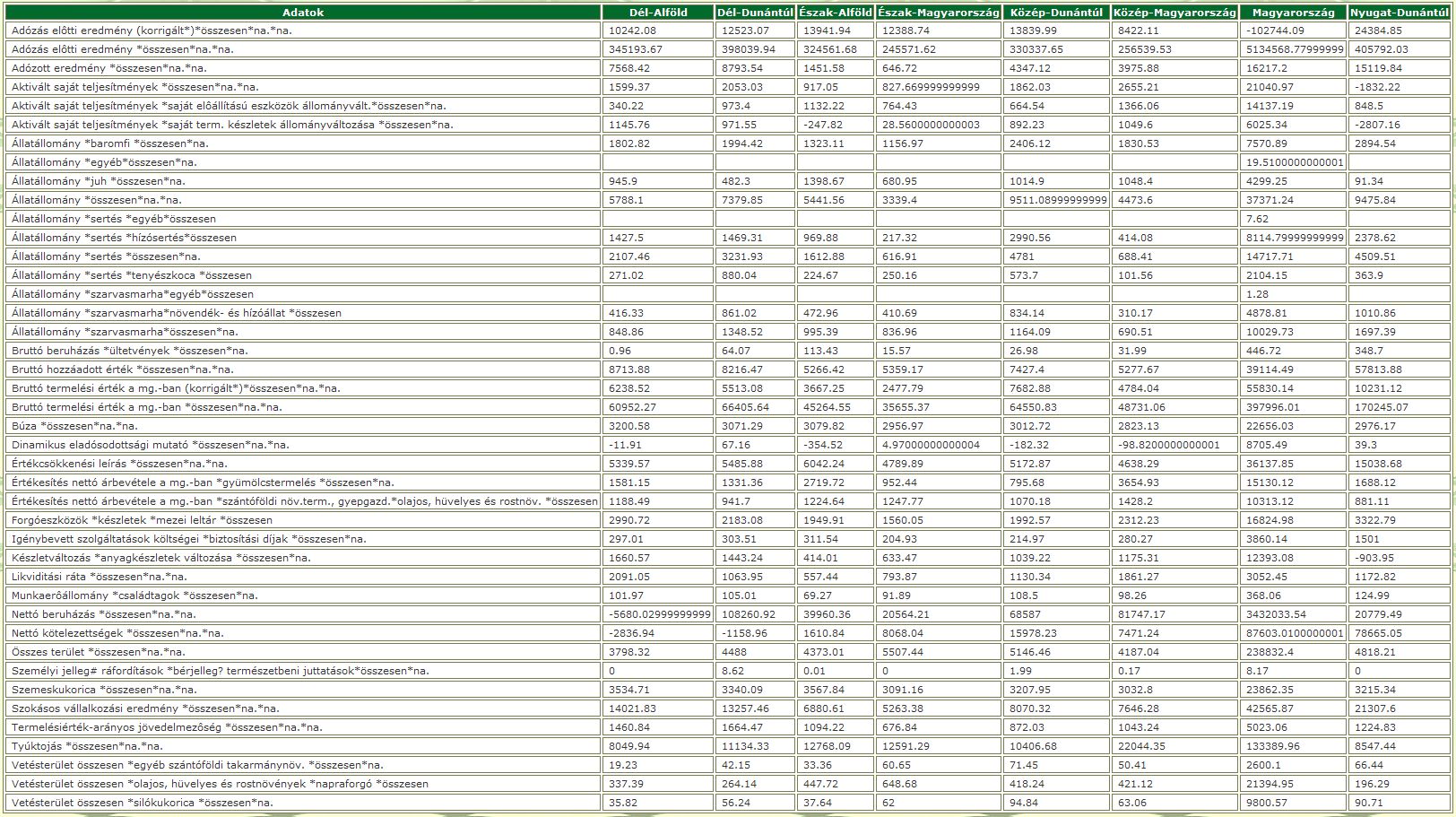
* **Adatvagyon bemutatása** – bemutatásra kerül az a fajta adathalmaz, amivel az alkalmazás segítségével kívánok dolgozni
* **A végleges problémamegoldás terve** – miért ezen az úton haladva valósítottuk meg az alkalmazást
* **Választott technológiák részletesebben** – részletesebb ismertetés a korábban tárgyalt technológiákról
* **Megoldási folyamat** – az alkalmazás megvalósításának menete
* **Végeredmény** – mit kaptunk eredményként és ez hogyan illeszkedik az eredeti elképzeléshez.

## Adatvagyon bemutatása

Napjainkban az információ mennyisége és az adatáramlás sebessége exponenciálisan nő. Ezzel párhuzamosan nő az adatok közti kapcsolatok száma, így akármilyen témakört is nézünk, mindenhol előfordulhat olyan adatmennyiség, amit szűrések és rendezések nélkül, a számítógép segítségével is nehezen vagyunk képesek átlátni.

Ha alapul vesszük az OLAP táblákat, ott is megfigyelhetjük, hogy ezek is már egy szűrt eredményhalmazt adnak az adott adatmennyiségből. Viszont könnyen megeshet, hogy a szűrt eredményt tovább kell szűrni (akár többször is). Például olyan esetben, amikor egy pénzügyi intézet jelentéseket készít mért adatokból (az adatok közti összefüggéseket pedig OLAP rendszerrel tárták fel) készült táblázatok adatai megjelenhetnek havi-, heti- és esetleg napi bontásban. Példa OLAP tábla a 7. ábrán látható.

Mit tehetünk ilyenkor? Például a szűrt eredményhalmazt egy speciális alkalmazással és bonyolult, időigényes transzformációkkal valahogy tovább szűrjük, vagy az egész táblázatot egy viszonylag könnyen kezelhető – böngészőben is futni képes – alkalmazásnak odaadjuk, és az elvégzi a szerver-oldali manipulációk helyett a megfelelő transzformációkat, típusmegfeleltetéseket. A felhasználónak pedig nem marad más dolga, mint pár szűrési szabályt definiálni és a legvégén, tovább dolgozni a szűrés szűrésének az eredményével.



. ábra – részlet egy OLAP táblából a http://miau.gau.hu/myx-free/olap/olap2/2\_olap\_m.php3 oldalról

## Végleges problémamegoldás terve

A végleges terv tehát az, hogy ha a felhasználónak van valamilyen táblázata, amivel a továbbiakban szeretne dolgozni, akkor egy olyan felületet kell biztosítani számára, ami egyértelmű, átlátható és egyben használható is.

A cél az, hogy a teljes alkalmazás a böngészőn belül fusson, minden más igény a háttérbe szoruljon, mivel nincs mód más technológiák és erőforrások bevonására.

Az alkalmazás ezért lett úgy felépítve, hogy az adott OLAP rendszerrel könnyen integrálva, máris használhatóvá válik a felülete, ahol a felhasználó azonnal el kezdheti manipulálni, az OLAP által „kiköpött” eredménytáblázatokat, halmazokat.

## Választott technológiák részletesebben

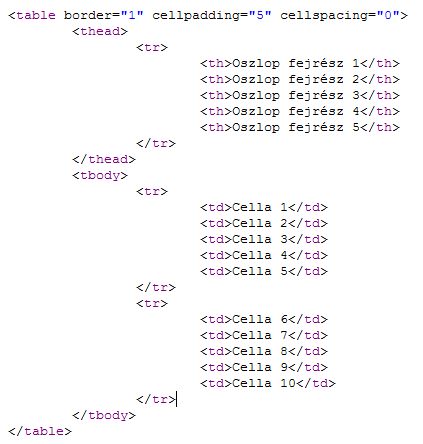
A korábban említett HTML és JavaScript párosra építettem az alkalmazást.

A HTML tehát egy leíró nyelv, melynek logikája és struktúrája kötött, nem ad teret a fejlesztőnek olyan értelemben, hogy adott problémának több megoldása lenne. Nincsenek változók, függvények, ciklusok. Minden HTML fájl egy leírást ad arról, hogy a benne létrehozott tartalomnak hogyan kell kinéznie a böngészőben. Többnyire mindenre van kész megoldása, ami ma az Interneten megtalálható: képek, táblázatok, videók, paragrafusok, címsorok, számozott és számozatlan listák, külső objektumok beágyazása (Adobe Flash, QuickTime, Windows Media stb.).

Az alkalmazásnál maradva, a tervezés fázisában alaposan szemügyre vettem a HTML nyelv által nyújtott lehetőségeket és rájöttem arra, hogy miért kellene bonyolult megoldások kialakításával foglalkozni, amikor úgymond, félkészen kaphatok mindent, amire szükségem van.

Arra is rájöttem, hogy mivel csak táblázatokkal fogok dolgozni, ezért megkötöttem az alkalmazás szempontjából azt, hogy hogyan kell felépülnie egy ilyen táblázatnak.

A következő példán keresztül szeretném szemléltetni, hogy az alkalmazásom szempontjából hogyan kell a HTML nyelv elemeit felhasználva egy táblázatot felépíteni forráskód szinten (8. ábra).



8. ábra – egy táblázat HTML forráskódja (saját ábrázolás)

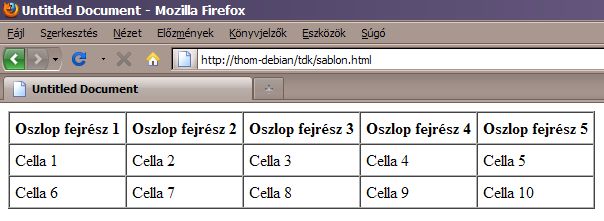
A 8. ábrán nagyon jól látható, hogy miért is leíró nyelv a HTML. A forráskódban az első és az utolsósorban található <table></table> címkepáros mondja meg a böngészőnek, hogy az általuk közrezárt tartalomtípus táblázat lesz. A <table> címkében található egyéb attribútumok nem kötelezőek, pusztán a böngészőnek mondják meg, hogy alapvetően hogyan jelenítse meg a táblázatot. Legyen-e a táblázatnak szegélye, a cellák között mekkora távolságot tartson, a cellaszegély és a tartalom között mekkora távolságot tartson stb.

A következő szinten azt tudjuk megmondani a böngésző értelmező motorjának, hogy melyek azok a cellák, amik a táblázat fejlécéhez tartoznak és melyek azok a cellák, amik a táblázat törzséhez. A fejléc rész a <thead></thead> címkék között, míg a törzsi rész a <tbody></tbody> rész között helyezkedik el.

Akár a fejlécről, akár a törzsi részről beszélünk, mindkét esetben először létre kell hozni egy sort, mely tetszőleges számú cellát tartalmazhat. Sort a <tr></tr> címkével tudunk létrehozni. A sorokban pedig cellát a <th></th> vagy <td></td> címkével tudunk létrehozni, azonban a <th> és a <td> címke között van egy árnyalatnyi különbség, mely csak az értelmezés szempontjából releváns. A <th> címke csak a fejléc részben szerepelhet, a <td> pedig csak a törzsi részben.

Bár a HTML lehetőséget nyújt összevont cellák létrehozására (ilyenkor az összevonandó cella esetében a <td> címkén belül meg kell adni egy attribútumot, ami a colspan névre hallgat), ám az alkalmazás szempontjából ez nem megengedett, mert gyakorlatilag lehetetlenné tenné a szűrő szempontjából a táblázatkezelést.

A 9. ábrán látható forráskód egy böngészőben a következőképpen jelenik meg:



9. ábra – Megjelenített táblázat a web böngészőben (saját ábrázolás)

A képen jól látható, hogy már itt más módon kezeli a böngésző az táblázat fejlécét annak törzsétől, ugyanis vastagon szedi a fejlécben szereplő tartalmakat, míg a törzsben nem, pedig a forráskódban nem található erre utaló címke (pl.: <strong></strong> vagy <b></b>).

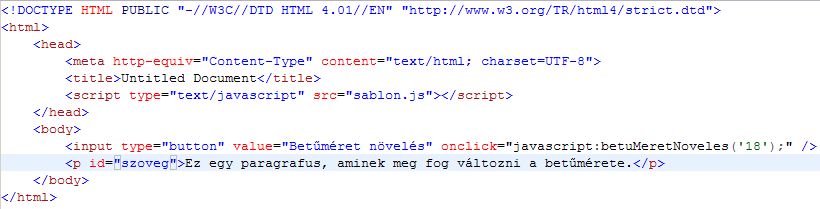
A HTML – ahogy azt korábban is megjegyeztem – lehetőséget ad, számos más típusú objektum, vagy objektum halmazok implementálására is az adott oldalon belül, ehhez csupán ismerni kell a megfelelő szintaktikát.

Ezen lehetőségek közé tartoznak a képek, videók, űrlapok, vagy akár önálló programocskák, amik vagy kiegészítik, vagy módosítják, esetleg teljesen más értelmezést adnak az oldalon található objektumoknak.

Pontosan erre való többek között a JavaScript is. Önálló program, vagy programcsoportként egyaránt felfogható, a keretrendszerek támogatásával pedig a lehetőségek és elérhető szolgáltatások csoportja tovább nőtt, megtartva azt az egyetlen tulajdonságot, hogy minden böngészőben működni fog a program.

A JavaScript teljes egészében képes a teljes tartalom fölé pozícionálnia magát, így – bár a későbbiekben is lesz róla szó – képesek vagyunk közvetlenül elérni a HTML forráskódban található objektumokat, ezzel egy időben pedig vezérelhetjük is azokat.

Működési példa. Van egy, a 10. ábrán látható forráskódunk:



10. ábra - HTML forráskód, ami tartalmazza a JavaScript fájl hivatkozását és egy paragrafust, valamint egy gombot (saját ábrázolás)

A fenti kódban egy teljes HTML dokumentum leírása látható. Csak úgy, mint a korábbi táblázat-példánál, itt is több részre különül el maga a dokumentum.

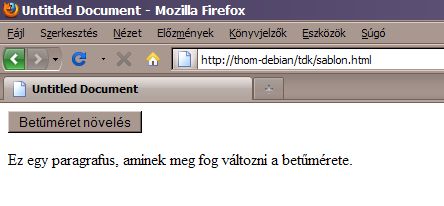
A <html></html> címkék közti rész tartalmazza az érvényes forráskódot, ezt veszi figyelembe a böngésző. A <head></head> címkék közti tartalom szintén értelmezésre kerül, azonban az itt szereplő információk magára az oldalra, annak megjelenítésére, a kereső robotoknak és a böngészőnek szólnak, látható tartalma nincs (ez alól kivétel a <title></title> címkepáros, mely a böngészőablak fejlécében jelenik meg). A valódi tartalom csak a <body></body> címkék között található, ez kerül valójában megjelenítésre.

Nézzük részleteiben is a dolgot. Látható, hogy a <head></head> részben van egy olyan sor, ami <script></script> objektumot definiál. Ez a címkepáros mondja meg a böngészőnek, hogy a definiált objektum JavaScript típusú (type=”text/javascript”) és magát az objektumot pedig a **sablon.js** fájlban (src=”sablon.js”) találja meg.

A <body></body> részben két objektumot találunk, az egyik egy gomb, a másik pedig egy paragrafus (<p></p>), aminek az azonosítója (id) „szoveg”.

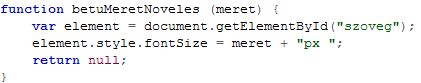
A példa lényege az, hogy amikor a felhasználó majd a gombra kattint, akkor a JavaScript segítségével a paragrafus szövegének betűméretét megnövelem 18 képpontosra. Ehhez meg kell mondanom a böngésző JavaScript értelmező motorjának, hogy melyik függvényhez forduljon, amikor kattintás történik a gombon. Erre a célra szolgál a gomb esetében az onclick attribútum.

A teljes HTML tartalom megjelenítve a 11. ábrán látható:



11. ábra – a 10. ábrán látható forráskód megjelenítve a böngészőben (saját ábrázolás)

Amint megtörtént a kattintás a gombra, a böngésző, a korábban definiált fájlban megkeresi a függvényt, a mi esetünkben a betuMeretNoveles névre kereszteltet. A JavaScript függvény forráskódja a 12. ábrán látható.



12. ábra – a JavaScript függvény forráskódja (saját ábrázolás)

A függvénynek van egy bemenő paramétere, ez fogja meghatározni a paragrafus szövegének betűméretét a szkript lefutása után.

Az első sor azonosítja az objektumot, a mi esetünkben a szoveg azonosítóval rendelkező paragrafust. A második sor beállítja annak új méretét, míg a harmadik sor azt mondja a böngészőnek, hogy a függvénynek nincs olyan visszatérési értéke, amit mindenképpen vissza kell adnia.

A függvény sikeres lefutása után, a megjelenített tartalomban a 13. ábrán látható módosítást fogja eredményezni.



13. ábra – módosított betűméret a script lefutása után (saját ábrázolás)

Természetesen ennél sokkal bonyolultabb és összetettebb feladatokra is használható a JavaScript nyújtotta lehetőségek. Mivel önálló programozási nyelv, ezért a lehetőségek tárháza végtelen, csak a fejlesztő képzelete szab határt.

## Megoldási folyamat

Rendelkezésemre áll minden –a technológia által nyújtott – szolgáltatás, ami a probléma megoldásához elegendő. Az egyetlen feladatom most már csak az, hogy ezeket helyesen alkalmazva elérjem a kitűzött célokat.

Azt már korábban említettem, hogy a HTML oldalról elegendő annyi, hogy a táblázatok kialakításakor egy adott struktúrát követelek. Ez a követelés több szempontból is előnyös a számomra.

Az első, hogy így mindig ugyanazt fogom kapni és nem kell törődnöm az egyedi kialakítású táblázatokkal, hiszen ha nem jó a formátum, akkor nem is fog működni.

A második ilyen előny, hogy így felesleges fejlesztésektől kímélem meg magam, mert nem kell kvázi, végtelen számú sémára felkészítenem az alkalmazást.

Tehát, a HTML struktúra adott, csak illesztenem kell hozzá a JavaScript-et. Itt jön képbe a keretrendszer. Az alkalmazás fejlesztése során én a jQuery keretrendszert (jQuery: The Write Less, Do More, JavaScript Library, 2009) választottam, mert én személy szerint ezt a keretrendszert részesítem előnyben. Számomra logikus, érthető és végtelenül egyszerű.

A jQuery-ben már korábban is az fogott meg, hogy rendkívül moduláris, viszont önmagában is rengeteg olyan szolgáltatást tartalmaz, ami az esetek 90 százalékában is több mint elegendő. Olyan alapvető szolgáltatásai vannak, aminek kézzel történő implementálása akár napokat, sőt heteket venne igénybe, ha nekem kellett volna megírnom. Végtére is, miért találnám fel megint a kereket, mikor azt előttem mások már kitalálták. Nekem „csak” annyi a dolgom, hogy a kereket a helyére illesztve, finomítok a működésén, majd használom.

Az első – és egyben legfontosabb – szolgáltatása a keretrendszernek az, hogy alapvetően úgy működik, hogy minden hozzá kapcsolódó script csak akkor fut le, amikor már a teljes dokumentum be lett töltve, sikeresen meg lett jelenítve.

Ez azért fontos számomra, mert könnyen elképzelhető akkora táblázat, hogy a böngésző már rég elkezdte feldolgozni és vele együtt megjeleníteni a HTML tartalmat, amikor az még egészében nem érkezett meg a felhasználóhoz.

Lássuk, hogyan is épül össze az alkalmazás. A részletekre az adott szinten kívánok kitérni és azokról magyarázni, elsőre túl összetett lenne mindenről beszélni.

Van négy darab fájlom, melyek ugyanazon a szinten vannak egymáshoz viszonyítva, viszont mindegyiknek más a célja. A fájlok leírását az 2. táblázatban foglaltam össze.

|  |  |  |
| --- | --- | --- |
| Fájlnév | Típus | Leírás |
| tablazat.html | HTML | Az a HTML fájl, ami magába foglalja a tárgyalt táblázatot. A táblázat legfőképpen valamilyen szerveroldali nyelv által generálódott, az én esetemben PHP ez a nyelv. |
| jquery.js | JavaScript | Ebben a fájlban található a keretrendszer magja. Általam nem lett módosítva |
| columnfilter.js | JavaScript | Ez a fájl tartalmazza a szűrő alapjait. Szintén nem lett általam módosítva. |
| filter.func.js | JavaScript | Ebben a fájlban kerül összepárosításra a HTML kód és a keretrendszer, valamint a szűrő modul függvényei a HTML tartalommal. |

2. táblázat – a fájlok leírását tartalmazó táblázat (saját ábrázolás)

Fájlrendszer szinten pedig a következőképpen helyezkednek el:

/konyvtar/

|-tablazat.html

|-jquery.js

|-columnfilter.js

|-filter.func.js

Ahogy azt a korábbi példánál már láthattuk, itt is meg kell mondani a böngészőnek, hogy a HTML dokumentumon kívül, melyek azok a fájlok, amiket még szeretnénk betölteni.

Mivel a keretrendszer úgy lett kialakítva, hogy a betöltés után automatikusan inicializálódjon, így nincs szükség arra, hogy kézzel megtegyük. Egyszerűbben fogalmazva, amikor a HTML dokumentumon belül a böngésző megtalálja a definiált objektumhivatkozásokat, akkor azokat automatikusan betölti. A keretrendszer innentől kezdve aktív és használatra kész.

af_1.jpg

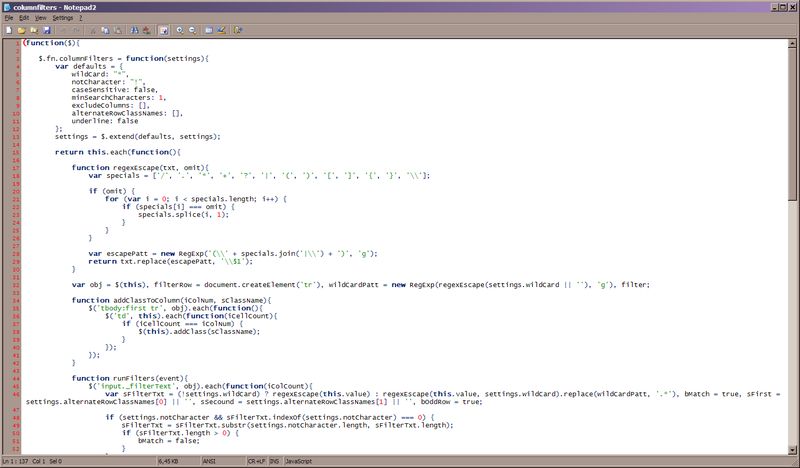
14. ábra – a HTML forráskódban található objektumhivatkozások (saját ábrázolás)

Maga a jQuery keretrendszer rendelkezik egy belső mutatóval, ami egy függvény keretén belül folyamatosan vizsgálja, hogy a betöltés állapota hol tart. Ez a függvény folyamatosan hamis értéket ad vissza mindaddig, amíg a teljes dokumentum, beleértve minden hivatkozott objektum betöltésre nem kerül. Amint a betöltés állapota kész, a függvény igaz értéket fog visszaadni.

Ez nekem azért jó, mert így, amíg a függvény nem adja vissza az igaz értéket, addig minden más általam írt függvény nem kerül inicializálásra, a böngészőnek a keretrendszer nem engedi futtatni a mini programokat. Ennek azért nagy a jelentősége, mert így a felhasználó ebből semmit nem vesz észre, másrészről a kód is nagyobb biztonságban fog futni, ha már minden szükséges input a programok rendelkezésére állnak. A 15. ábrán a keretrendszer összetett és bonyolult, egyben tömörített forráskódja tekinthető meg.

A keretrendszer betöltése után a böngésző betölti a letöltött modult is. Amint ez megtörtént, a modul – mivel a keretrendszer már korábban betöltésre került – automatikusan hozzáfér a keretrendszer szolgáltatásaihoz speciális címzéseken keresztül, de nekünk ezzel úgyszintén nem kell törődni, hiszen a böngésző motorja helyettünk automatikusan megteszi.

A 15. ábrán szeretném megmutatni a szűrő motor forráskódját, valamint szeretném néhány fontos részletre felhívni a figyelmet.



15. ábra – a modulárisan telepíthető szűrő forráskódja (saját ábrázolás)

A forráskód talán önmagában annyira nem is érdekes. Viszont érdemes egy kicsit közelebbről szemügyre venni a kód legelejét.

$.fn.columnFilters = function(settings){

var defaults = {

wildCard: "\*",

notCharacter: "!",

caseSensitive: false,

minSearchCharacters: 1,

excludeColumns: [],

alternateRowClassNames: [],

underline: false

};

settings = $.extend(defaults, settings);

A forráskód e része nagyon fontos a számomra, ugyanis a szűrő fejlesztője gondolt olyan eshetőségekre is, ami az alkalmazása során jól jöhet. Gyakorlatilag ez a pár sor a szűrő konfigurációjáért felelős. A beállítások leírását a következő, 3. táblázatban olvashatjuk.

|  |  |  |
| --- | --- | --- |
| Attribútum | Alapértelmezett beállítás | Leírás |
| wildcard | \* | A joker karakter állítható be vele. A csillag bármilyen karakter lehet. |
| notCharacter | ! | Azokat a feltételeket, amiket ezzel a karakterrel kezdünk, nem lesznek benne az eredményhalmazban. |
| caseSensitive | false | Beállítja, hogy a keresés kis-nagybetű érzékeny legyen, vagy sem. |
| minSearchCharacters | 1 | Minimum karakterszám a keresésnél. |
| excludeColumns | - | Megadhatjuk, hogy az adott táblázat mely oszlopaiban nem kívánunk keresni. Az oszlop sorszám 0-tól indul. Pl. ha megadjuk az 1-est, akkor a táblázat 2. oszlopában nem fogunk tudni keresni. |
| alternateRowClassNames | - | Megjelenítésnél van jelentősége, megadhatjuk, hogy mi legyen az adott sor osztályneve. Később ennek az osztálynak különféle formázást tudunk adni. |
| underline | false | Engedélyezzük az alávonást, vagy sem. |

3. táblázat – beállítások részletes leírása (saját ábrázolás)

Ezeket a beállításokat két oldalról is megközelíthetjük. Lehetőségünk van globális szinten szabályozni a paramétereket. Itt kell átírni az alapértelmezett értékeket. Ekkor minden szűrő inicializáláskor ezek az értékek lesznek beállítva. Ez főleg akkor hasznos, ha több helyen is ugyanazokkal a beállításokkal szeretnénk használni a szűrőt.

A második megközelítés az, hogy minden egyes szűrő inicializáláskor, ott a szűrő definiálásakor adjuk meg ezeket a paramétereket. Ekkor a beállítások csak az adott szűrőre lesznek hatással. Ennek akkor lehet jelentősége, ha több helyen szeretnénk a szűrőt használni, viszont minden egyes szűrőnél más beállítások szükségesek.

Szóval, be van töltve a keretrendszer, a szűrő, a HTML tartalom teljes egészében megérkezett a böngészőhöz. Itt az idő, hogy alkalmazzuk a táblázaton a szűrőt.

Erre a feladatra szolgál az a pici forráskód, ami a filter.func.js fájlban szerepel. Ennek a függvények a feladata, hogy a HTML dokumentumban található táblázatra inicializálja a szűrőt a keretrendszer segítségével. A forráskódot a 16. ábra tartalmazza.

kep.jpg

16. ábra – a szűrő inicializálását elvégző script (saját ábrázolás)

A 16. ábrán jól látható, a korábban már említett várakoztató szkript. A $(document) objektum azonosítja a keretrendszer számára a teljes HTML dokumentumot. A ready függvény pedig egészen addig várakoztatja a benne szereplő programrész lefutását, amíg a böngésző a teljes tartalmat be nem töltötte.

Amint ez megtörtént, a program belép abba az ágba és elkezdi inicializálni a szűrőt.

Első lépésként szükség van az adattáblázat azonosítására. Ezt a feladatot a $(”table#content”) kifejezéssel tehetjük meg. Fontos megjegyezni, hogy a jQuery-ben minden elemazonosítás egy dollárjellel kezdődik, majd ezután zárójelek között kerül kifejezésre az adott HTML elem. A mi esetünkben tehát egy táblázat, aminek content az azonosítója. Számos más módszerrel is azonosításra kerülhetnek az objektumok, néhány példát a 4. táblázat foglal össze.

|  |  |
| --- | --- |
| HTML kód | jQuery által kezelt kód |
| <table id=”content”> | $(”table#content”) |
| <table class=”content”> | $(”table.content”) vagy $(”table”).filter(”content”) |
| <table> | $(”table”) |

4. táblázat – adott HTML elem azonosítási módszerei (saját ábrázolás)

Második lépésként meg kell mondanom a szűrőt létrehozó kódnak, hogy melyik elemet kell keresnie a HTML dokumentumon belül, tehát melyik táblázatra kell a szűrőt beállítani.

Korábban már említettem, hogy a keretrendszer és a szűrő betöltődése után, már lehet hivatkozni a különféle függvényekre, mert a böngésző ebben a segítségemre van. A saját fájlomban – ennek köszönhetően – már csak annyi a dolgom, hogy egy függvényt hívok meg a korábban azonosított elemre. Ez a függvény a columnFilter() névre hallgat.

Még egy kis kitérő. A jQuery másik nagy előnye, hogy a különféle függvényeket láncba lehet fűzni. Ez azért jó, mert így minden objektumot elég egyszer definiálni a programban és így jelentősen kevesebb fizikai helyet foglal maga a forráskód, valamint a kód futása is jóval rövidebb idő alatt fog végbemenni. Nem utolsó szempont, hogy a fejlesztő mindent egy helyen talál meg.

Például, ha van egy paragrafus, ami csak egy képre kattintva fog megjelenni és a megjelenés közben még valamilyen effekt is lejátszásra kerül, akkor annak kezelése a láncolhatóság miatt roppant egyszerűvé válik.

$(”img#showp”).ready().click(

function() {

$(”p#content”).slideIn(”slow”).show();

}

);

17. ábra - példa program a láncolhatóság szemléltetésére (saját ábrázolás)

A 17. ábrán látható forráskód szerintem remekül megmutatja, hogy mennyire jól használható a láncolhatóság.

A 17. ábra magyarázata. Azonosítom a HTML dokumentumon belül a kép objektumot. Megvárom a ready() függvénnyel, hogy teljesen betöltődjön, majd a click() függvényen belül definiálom a paragrafusra vonatkozó szabályokat, eseményeket.

A click() függvény azt mondja meg a keretrendszernek, hogy csak akkor indítsa el a benne leírt programot, amikor a kattintás érkezik arra az objektumra ahova kötve lett.

Amikor elindul a belső szkript, akkor megkeresi azt a paragrafust, aminek content az azonosítója (lásd: 4. táblázat), majd aktivál rajta egy effektet, aminek van egy bemenő paramétere, és a legvégén megjeleníti a paragrafust.

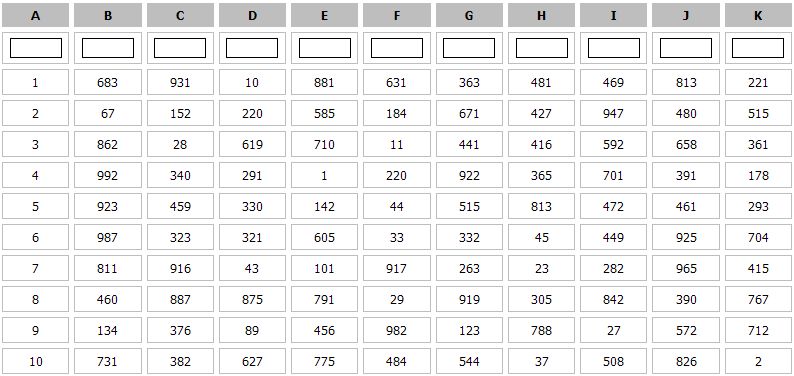
Tehát, most már van teljesen betöltődött HTML dokumentumunk, amely tartalmazza szükséges OLAP táblázatot. A dokumentum mellett sikeresen betöltöttük a szükséges JavaScript objektumokat, majd azokat sikeresen inicializáltuk.

Hogyan is működik mindez együtt? Rendkívül egyszerűen, ám azért egy kevés magyarázatot igényel a téma.

Amikor minden sikeresen betöltődött, a columnFilter függvény a definiált táblázatot el kezdi feldolgozni és azon belül megkeresi a <thead></thead> részt. Ez azért fontos, mert a táblázat fejlécében a szűrő minden oszlop tetejére elhelyez egy beviteli – úgy nevezett input – mezőt. Minden mező a saját oszlopára vonatkozik, tehát amikor valamit beírunk az egyik ilyen mezőbe, akkor a keresési feltétel csak az adott oszlopra fog vonatkozni.

Mikor a beviteli mezők sikeresen létre lettek hozva az egyes oszlopokban, a szűrő programkód beviteli értékékre vár. Ezt az állapotot a 18. ábra mutatja be.

A beviteli értékek megadása tetszőleges, egyik mező sem rendelkezik „kötelező” címkével.



18. ábra - a definiált táblázaton inicializálásra került a szűrő (saját ábrázolás)

Hogyan működik a keresés? Egyszerűen. Amikor elkezdjük gépelni a feltételeket, akkor a program minden egyes karakterleütés után újrafuttatja a keresést. Ezt a folyamatot a 19. ábra mutatja be a következő oldalon.

Maga a kereső algoritmus egy nagyon régi, ám nagyon jól bevált módszeren alapul, ez pedig nem más, mint a reguláris kifejezés.

. ábra – a keresés mechanizmusa (saját ábrázolás)

A reguláris kifejezésen alapuló szövegazonosítást a számítástechnikában már elég régóta használják, mert néhány, jól kitalált szabállyal igen könnyen lehet szöveget, szövegrészletekre, vagy karakterekre keresni.

Például a [A-Za-z0-9\_] szabály csak olyan szövegeket fog visszaadni eredményül, amiben csak az angol ABC betűi, 0-tól 9-ig a számok, valamint az alávonás jelek valamelyike, vagy valamilyen kombinációi szerepelnek.

A reguláris kifejezések működését nehéz némi tapasztalat hiányában megérteni elsőre, de ha az ember egyszer ráérez a logikájára és mellé megtanulja, hogyan kell a szabályokat formázni, mik az alapvető vezérlő karakterek és utasítások, onnantól kezdve sokkal hatékonyabban lehet használni klasszikus szövegkeresésre, mint bármilyen más technológiát.

Ezen felül pedig külön öröm, hogy a reguláris kifejezést nem csak a JavaScript, hanem az összes többi, valamit magára adó programozási nyelv is támogatja, alkalmazza nap, mint nap.

Ezzel egy időben el is érkeztünk az első fejlesztési szakasz végéhez. Van egy működő keresőnk, ami a megadott táblázatra automatikusan ráépül, felhasználva a kötött struktúrát. Könnyen kezelhető, mert nincsenek bonyolult szabályok, több szintű feltételrendszert megalkotó felületek. Az implementálása már egy kész, meglévő rendszerbe sem kimondottan bonyolult, hiszen a struktúra és a logika már ezen a téren is adott, pusztán alkalmazni kell.

Egy ilyen szintű kereső beillesztése egy már meglévő rendszerbe, nem vesz el több időt a fejlesztő életéből, mint ha meginna egy kávét.

Ráadásul, az adatok manipulálása a felhasználó számítógépén történik, így a változás azonnali, az eredmény azonnali és nem vagyunk veszélyeztetve, egy esetleges web kiszolgáló hibának, hiszen a web kiszolgálótól csak az adatokat kaptuk, minden egyéb folyamat és eljárás már helyben kerül megvalósításra.

# Mérések

* **Használat közben** – képernyőképek
  + első teszt
  + második teszt
  + harmadik teszt
* **Tesztelési tapasztalatok** – mennyivel egyszerűbb az eszközzel egy adathalmazban keresni, tapasztalatokkal és véleményekkel

## Használat közben

A tesztelés érdekében írtam egy pici programot, ami egy 1000 soros táblázatot generál. A táblázat fejlécei a következőek:

* Sorszám – szám
* Név – szöveg
* Születés dátuma – vegyes (szöveg és szám)
* Kor – szám
* Telefonszám – vegyes (szimbólumok és szám)
* Cím – vegyes (szöveg és szám)
* Kedvenc szám – szám

A minta-adatbázis összesen 10 embert tartalmaz (plusz a hozzájuk tartozó egyéb adatokat), amiből egy véletlenszám-generátorral választom a táblázatot alkotó sorokat.

A teszteléssel és a fejlesztői használattal azt szeretném mérni, hogy egy 1000 soros táblázattal a szűrő hogyan boldogul, milyen futási sebességek érhetőek el, valamint az első működő kódban milyen alapvető hibák találhatóak.

A tesztelés során több dolgot is kipróbáltam, ezeket a teszteket az 5. táblázat tartalmazza.

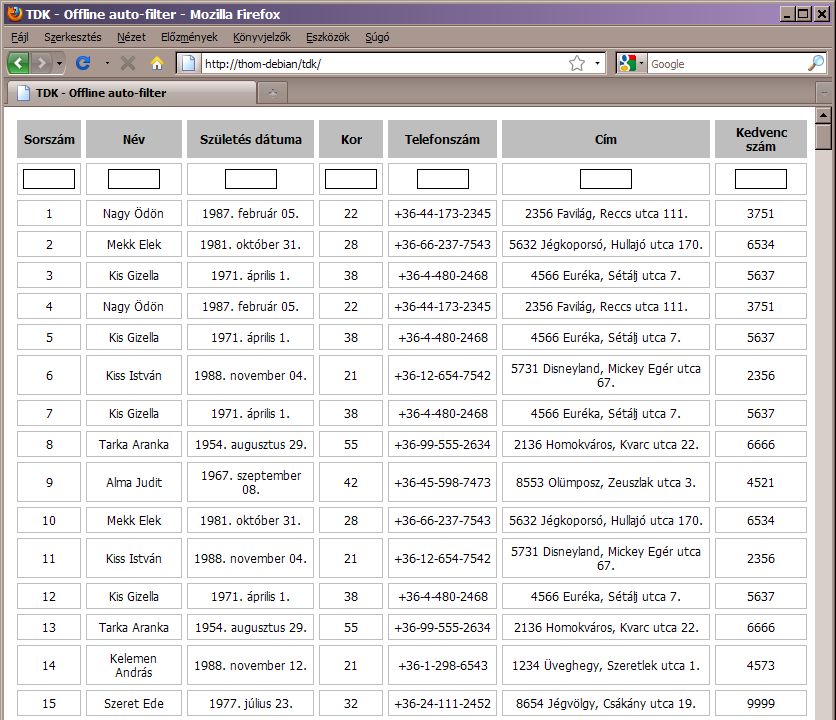
|  |  |
| --- | --- |
| Teszt sorszáma | Teszt leírása |
| 1. teszt | Egy oszlopra vonatkozó szűrés; minden olyan sorra vagyunk kíváncsiak, amelyekben „Kelemen András” szerepel. |
| 2. teszt | Két oszlopos szűrés; azokra az oszlopokra vagyunk kíváncsiak, amelyekben nem „Alma Judit” szerepel és a kedvenc szám 4000 felett van. |
| 3. teszt | Azokra az emberekre vagyunk kíváncsiak, akiknek a kedvenc száma 2000 felett van, a sorszám 100 felett van és nem „Szabó Zoltán”-ról van szó. |

5. táblázat – a tesztek leírását tartalmazó táblázat (saját ábrázolás)

### Első teszt

Az első teszt megkezdése előtt teljes egészében megvártam, míg az oldal betöltődik. Az induló adatvagyon mindhárom teszt esetében ugyanaz volt.

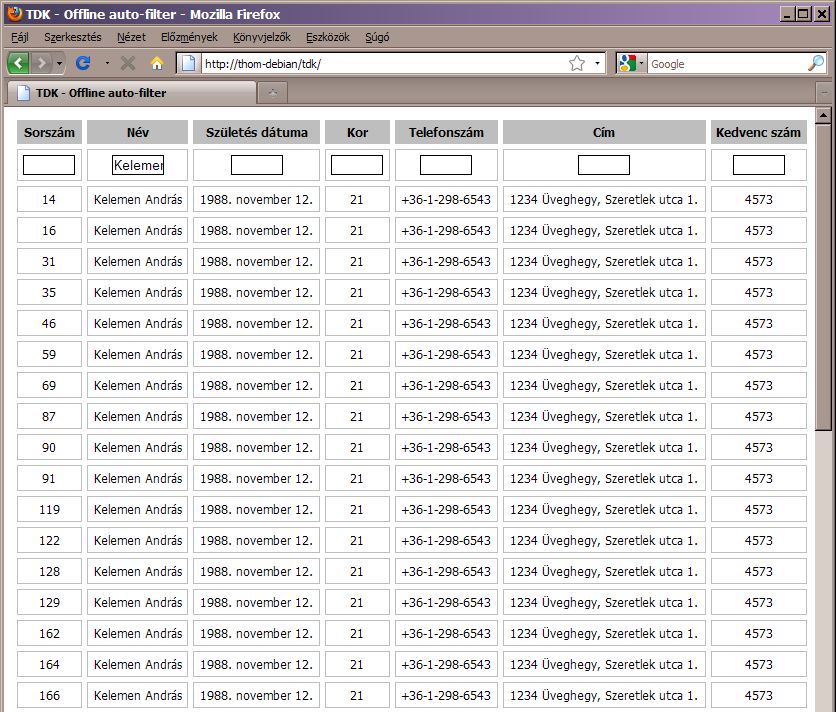
Tehát, az induló adatvagyonról készült képet a 20. ábra tartalmazza.



20. ábra – az induló adatvagyon, mely 7 oszlopban és 1000 sorban helyezkedik el (saját ábrázolás)

A szűrés megkezdéséhez a felhasználónak csupán annyi dolga van, hogy az általa keresett információt a megfelelő oszlop tetején található input mezőbe beírja. Amint a gépelés befejeződött, a kereső elkezd dolgozni (lásd 19. ábra). Az első teszt feltételének begépelése után az eredmény halmazt a 21. ábra mutatja meg.

Az első teszt teljes futási ideje a gépelés abbahagyása után: 7,5 mp (saját eredmény).

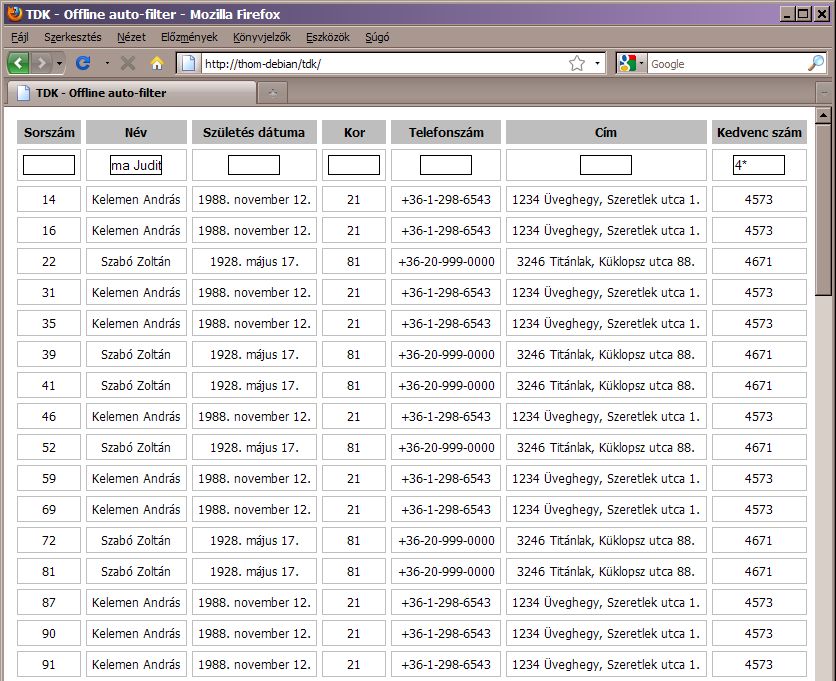


21. ábra – az első teszt eredményhalmaza a szűrés után (saját ábrázolás)

### Második teszt

A kiindulási adatvagyont nem változtatva, elvégeztem a második tesztet. A „név” oszlopba begépeltem „Alma Judit” nevét, melyet egy felkiáltó jellel kezdtem. Erre azért volt szükség, mert a szűrő a jelből fogja tudni, hogy a begépelt adatra nincs szükség a szűrés eredményhalmazában. Második lépésként pedig a „kedvenc szám” oszlop tetején pedig a „4\*” kulcsszót adtam meg. A csillag karakter pedig úgynevezett joker karakter, tehát minden olyan sor benne lesz az eredményhalmazban, ami egy 4-s számmal kezdődik. Az utána szereplő számok nem fontosak.

A második teszt szűrési eredményét a 22. ábra mutatja.



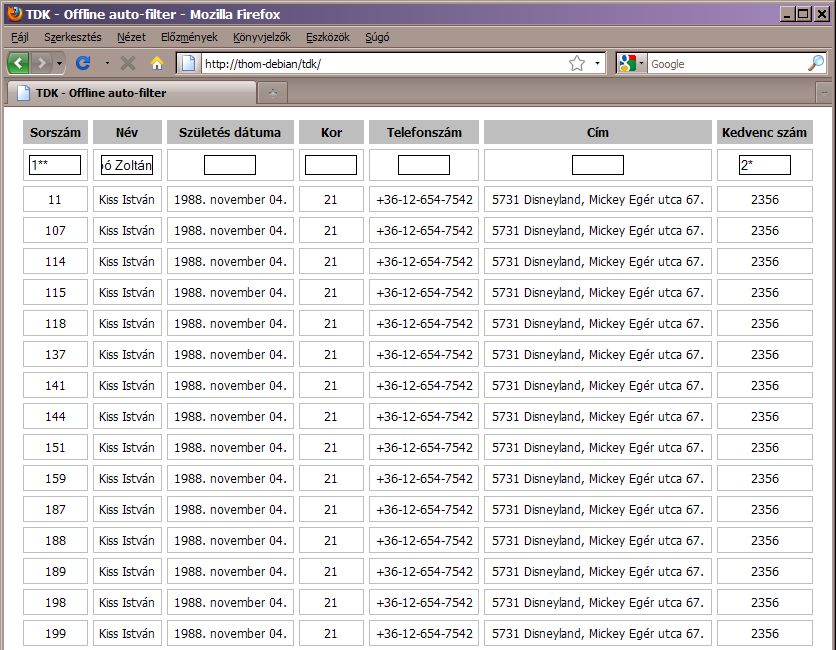
22. ábra – a második teszt szűrési eredménye (saját ábrázolás)

Érdekes módon a második teszt több szűrési szabályt tartalmaz, mégis gyorsabban futott le, mint az első teszt. A második teszt futási ideje 6,9 másodperc (saját mérés).

### Harmadik teszt

A harmadik teszt már három szűrési szabályt tartalmaz. Először beírtam „Szabó Zoltán” nevét egy felkiáltó jellel kezdve. A sorszám oszlopban beírtam, hogy „1\*\*” mely csak a 10 feletti sorokat hozza fel, majd legvégül a kedvenc számnál „2\*”-ot gépeltem be. A gépelés elvégzése után a szűrő munkának látott.

A harmadik teszt szűrésének eredményét a 23. ábra szemlélteti.



23. ábra - a harmadik teszt szűrési eredménye (saját ábrázolás)

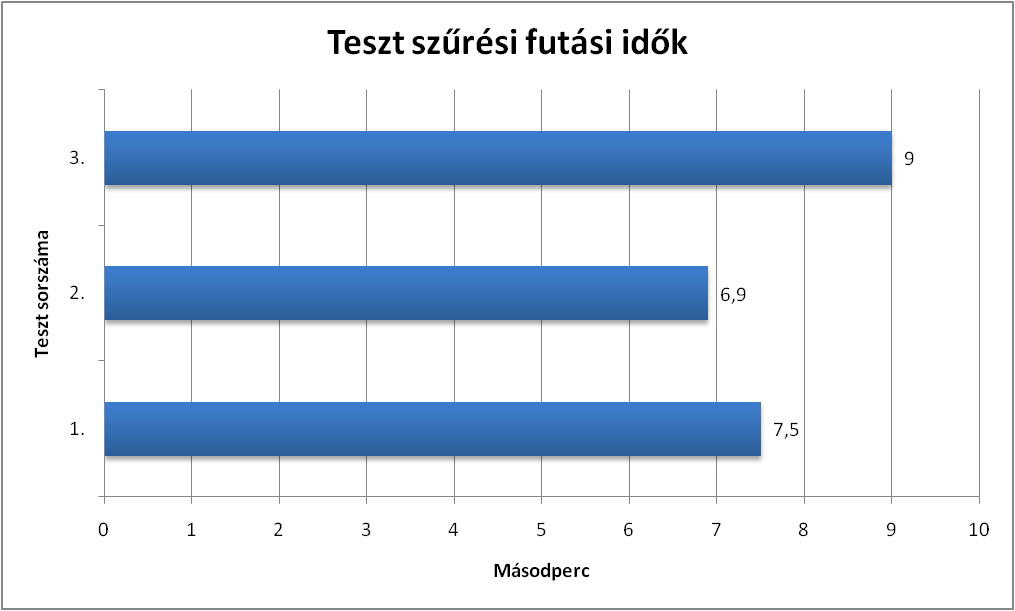
A harmadik teszt lefutása a plusz egy feltétellel, összesen 9 másodpercet vett igénybe (saját mérés).

## Tesztelési tapasztalatok

A tesztelés során a következő tapasztalatok alakultak ki. Bár az alkalmazás még rendkívül kezdetleges állapotban van, már most, ebben a formában komoly eredményeket lehetne vele produkálni.

Az is egyértelműen bebizonyosodott, hogy a kényelmi szolgáltatása is rendkívül vonzó, mert nem kell a felhasználónak kijelöléssel, másolással, transzformációkkal bajlódnia, mindent készen kap. Felbecsülhetetlen érték.

A script-futási sebességét nem tartom vészesen soknak, egyik teszt esetében sem. Nyilvánvaló, hogy soha nem fogja felvenni a versenyt egy ilyen, táblázatkezelő alkalmazással. Viszont azt sem szabad elfelejteni, hogy bár egy kicsit várni kell a szűrés eredményére a feltételek megadása után, de gondoljunk csak bele, hogy melyik a rosszabb: 7 másodpercet várni, vagy irracionálisan sok időt ráfordítva kézzel kigyűjteni a szükséges adatokat?

A 24. ábra egy grafikonon mutatja meg az tesztek futási idejét egymáshoz viszonyítva.

24. ábra – a tesztek futási ideje egy közös grafikonon (saját mérés és ábrázolás)

## Második fejlesztési szakasz

Okulva az első szakasz után végrehajtott tesztekből, előkerült pár olyan funkció, ami még tovább növelné az eszköz használatát és természetesen funkcionalitását. Az alábbi lista foglalja össze a további, implementálásra előterjesztett funkciókat:

* **kisebb-nagyobb-egyenlő funkció:** van, amikor nem elég a számokban úgy keresni, hogy csak azokat a számokat keresem, ami tartalmazza a megadott számjegyeket. Ilyen példa lehet valamilyen intervallumos szűrés.
* **gombnyomásra történő keresésindítás:** a tesztelők szemszögéből gyakran idegesítő probléma volt, hogy nem tudtak egymás után több mezőt kitölteni, mert amint végeztek az egyikkel, a kereső automatikusan elindult
* **a keresési feltételeknek megfelelő találatok darabszáma:** voltak olyan szituációk, amikor valaki nem a szűrt tartalomra volt kíváncsi, hanem arra, hogy a megadott paraméterek alapján pontosan hány találat van az adott rendszerben, OLAP jelentésben.
* **egyedi formázhatóság:** egy olyan integrációs vélemény is született, mely inkább esztétikai probléma mintsem funkcionális, de mindenképpen említésre méltó. Már korábban kifejtettem, hogyan is születnek meg a kereső mezők az adott táblázatban. Mivel ezeket a mezőket a JavaScript hozza létre, így nincs mód egyedi formázás megadására. Ennek a hibának a kiküszöbölése is fontos lett.
* **Adott adatok sorszámozása:** a MIAU kapcsán került elő a probléma. A lényeg, hogy ha valaki hasonlóságelemzést szeretne végezni egy OLAP kockából kieső adathalmazon, akkor első lépésként lekérdezi a szükséges adatokat, majd azt a saját igényei szerint rendezi. A MIAU-n elérhető hasonlóságelemző szoftver bemeneti paraméterként úgynevezett rangsorskálát vár (ami természetesen Excel-ben gond nélkül létrehozható). A rangsorskála lényege, hogy valamilyen irány szerint (növekvő-csökkenő) a megadott adatokból egy rangsort készít. Természetesen csak szám alapú adatokon működik. Majd a hasonlóságelemző e skála alapján kezd el elemezni és összehasonlítani, majd a végén a számított eredményt megjeleníteni.  
    
  Az volt az ötlet, hogy az alkalmazást használva amolyan mediátorként be lehetne illeszteni az OLAP jelentések és a hasonlóságelemző motor közé. Ezért vált fontossá a szűrt, szám alapú adatokból keletkeztethető rangsorskála.
* **induló adatsorok mutatása/elrejtése:** szintén a MIAU berkein belül futó kémia verseny (MY-X, 2010) eredménytáblázata kapcsán merült fel a probléma. Ha viszonylag sok adatrekordról van szó, – ahogy már korábban említettem – könnyen előfordulhat, hogy az oldal már rég el kezd megjelenni a böngészőben, viszont az összes adat még nem érkezett meg. Ekkor jött ez az igény, hogy jó lenne, ha konfigurációból vezérelhetővé válna az induláskor megjelenő rekordok és csak az első keresést követően jelennének meg az első rekordok (természetesen azok, amik a keresési paramétereknek megfelelnek).

Megvizsgálva a megjelent hiányosságokat rádöbbentem, hogy olyan alapvető funkciók lettek kifelejtve az alkalmazásból, amiknek igenis van létjogosultsága. A funkciók között vannak könnyebben és nehezebben megvalósíthatóak is.

### Kisebb-nagyobb-egyenlő funkció

A kisebb-nagyobb funkció bár első nekifutásra mindenképpen az egyszerű esetek közé tartozik, valójában nem volt az. Eleve, a könnyű integrálhatóság miatt minden olyan funkcionalitást, ami nem árt, ha könnyen testre szabható, konfigurálhatóvá tettem. Nem történt ez máshogy ezzel a funkcióval sem. A konfigurációs állományba további három beállítási attribútum jelent meg. Fontos megjegyezni, hogy itt a beállítási lehetőség pusztán a vezérlőkarakterre vonatkozik.

var defaults = {

smallerThan: "<",

greaterThan: ">",

equalTo: "="

}

Ennek tükrében a lista jelenleg így néz ki:

25. ábra – új változók létrehozása a globális konfigurációban (saját ábrázolás)

A 25. ábrán látható a két új attribútum a beállításokban, melyben könnyen és egyszerűen lehet definiálni azt a három karaktert, mely a keresési folyamatban reprezentálja a kisebb és nagyobb szimbólumokat. Természetesen a funkció csak azokon az oszlopokon fog sikeresen lefutni, melyekben csak és kizárólag számok szerepelnek.

Azért gondoltam erre a módszerre, mert mi történik akkor, ha az integrálási folyamatba mondjuk egy másik rendszer is beépül, ami szintén használná a kisebb-nagyobb jeleket, és például a két rendszer közti kommunikáció az URL-ben történik? Valószínűleg a két rendszer közötti átjárás problémássá válhat, így ezeket a gondokat könnyűszerrel megoldhatjuk, ha újradefiniáljuk, melyik szimbólum fogja elindítani az adott funkciót.

Magának a funkciónak az implementálása már egy fokkal nehezebb volt. Először is, a funkció alapja egy al-algoritmus volt a fő, keresési algoritmusban. Minden egyes cikluskörben az aktuálisan vizsgált sorban lefutásra kerül.

Az algoritmus folyamatosan vizsgálja, hogy az adott keresési mezőben megadott karakterlánc első karaktere megegyezik-e a kisebb-nagyobb-egyenlő változóban beállított értékkel, és ha igen, akkor az adott sor megjelenik a keresési eredmények között, ellenkező esetben nem.

Ha valamilyen oknál fogva az első vezérlő karakter érvényesnek bizonyulna, de az utána következő karakter nem szám, akkor úgy tekint rá a program, mint ha nem is adtak volna meg az adott oszlopra vonatkozó keresést és figyelmen kívül hagyja a beérkező paramétert (például: „>nagyobb”).

### Gombnyomásra történő keresésindítás

A gombnyomásra történő keresésindítás nem tartozik a bonyolultabb funkciók közé, azonban az implementálása után drasztikusan javul a felhasználói élmény. Egész egyszerűen arról van szó, hogy a keresést nem minden billentyűleütés után indítjuk el, hanem amikor már minden paraméter késznek mondható. Ennek kettős vonatkozása is van:

* **Overhead problémák:** már korábban említettem, hogy a jelenlegi megoldás erőforrás tekintetében nem egy központi számítógéptől származnak, hanem az adott felhasználó saját számítógépétől. Az overhead esetében arról van szó, hogy valamilyen ok miatt az adott program adott funkciójának használatához több erőforrásra van szükség, mint egyébként kellene. Egy jól megtervezett program esetében az overhead jelenség számát egész alacsonyra le lehet csökkenteni, pusztán egy kisebb méretű tervezésre van szükség
* **Kényelmi funkciók:** kisebb adatvagyon esetében nem probléma, ha minden karakterleütés után a JavaScript értelmező motor el kezd működni, azonban egy több-száz vagy ezer soros táblázatnál, melynek az oszlopszáma is több tízre tehető már macerás és lassú.  
  Ennek tükrében, ha egy gombnyomásra indul el a keresés, akkor a szűrésnek csak egyetlen egyszer kell lefutnia és máris készen kapjuk a teljes eredményhalmazt. Kisebb hátránya ennek a metódusnak, hogy újabb és újabb keresés esetén, mindig meg kell nyomni a gombot.

A gomb alapú keresés bekapcsolása esetén (melynek funkcionalitása szintén egy, a beállításokban definiált attribútumtól függ [26. ábra]).

…

isButton: false

…

26. ábra – a gomb alapú keresést beállító attribútum (saját ábrázolás)

A 26. ábrán jól látható, hogy a beállítás kétféle értéket vehet fel, melyek true vagy false. Ezt a programozási körökben logikai vagy igaz-hamis változónak (angol nevén boolean) hívják.

Ha true-ra van állítva, akkor a szűrőt létrehozó programkód a dokumentumban egy úgy nevezett input mezőt hoz létre, majd a keresést indító függvényt erre a létrehozott objektumhoz társít. Innen tudja a program, hogy a gomb kattintására kell a keresési algoritmust elindítani. Ha false, akkor a korábban vázolt módszerrel, tehát minden gombleütés után elindul a keresés.

Ahhoz, hogy a gomb alapú keresést használni tudjuk, a HTML oldalon létre kell hozni egy megadott nevű objektumot. A program ebbe az objektumban fogja elhelyezni a keresést indító gombot. A gomb bárhova elhelyezhető, a lényeg, hogy a program megtalálja.

### Találati darabszám

Számos alkalommal mikor valaki egy OLAP jelentéssel vagy nagy adatvagyonnal rendelkező táblázattal dolgozik, akkor van, hogy valójában arra kíváncsi, hogy a megadott kondíciókkal lefuttatott keresésnek hány elem felel meg.

Ezt a funkciót roppant egyszerű volt implementálni. Az alap koncepció úgy működik, hogy amikor egy keresés elindul, akkor a keresést végrehajtó ciklusban, amikor egy elem megfelel a keresési feltételnek, akkor egy, a programon belül definiált változó értékét mindig eggyel megnöveljük, majd a dokumentum legvégén magába a dokumentumba, a megadott elembe a JavaScript segítéségével kiíratjuk a végeredményt. Új keresés esetén természetesen a számláló újraindul. Lényegében létre kell hozni egy olyan objektumot (ami lehet div, table, p, akármi) a megadott azonosítóval és onnantól kezdve látni fogjuk, hogy hány sor felelt meg a keresési feltételeknek.

### Egyedi formázhatóság

Bár ebben az esetben esztétikai funkcionalitásról beszélünk, mégis fontos lehet. Ha valaki alkalmazza a programot a saját termékében vagy weboldalán, akkor fontos lehet, hogy a program által generált keresőmezők megjelenésükben alkalmazkodjanak az őt alkalmazó felület stílusaihoz, megjelenéséhez. Éppen ezért került bele a globális konfigurációba még egy változó, melyet a 27. ábra szemléltet.

searchBoxStyle: "fparam"

27. ábra – a keresőmezőkre vonatkozó stíluslap definíció (saját ábrázolás)

Használata és működése – ragaszkodva az elvekhez – egyszerű. Ebben a paraméterben kell megadni azt a CSS[[3]](#footnote-4) stíluslap osztályt, ami leírja a keresőmezők megjelenését (betűméret, betűstílus, keret, háttér stb.).

A program, amikor létrehozza a mezőket megnézni, hogy az adott attribútum rendelkezik-e értékkel. Ha igen, akkor a keresőmezők legenerálásakor automatikusan beilleszti a szükséges HTML kódot a mezők attribútumai közé. Így a mezők már a CSS-ben definiált szabályok szerint fognak megjelenni (28. ábra).

<input type=”text” class=”fparam” attr1=”val” attrn=”val” />

28. ábra – a szükséges HTML specifikus attribútum CSS szabályok használatához (saját ábrázolás)

Ha nem rendelkezik a beállítás, akkor egyszerűen figyelmen kívül hagyja és nem generálja bele a szükséges kódot (29. ábra).

<input type=”text” attr1=”val” attrN=”val” />

29. ábra – hiányzik a „class” attribútum így nincs CSS szabály alkalmazása (saját ábrázolás)

### Sorszámozás

Ez a funkció rendkívül nagy szerepet kapott. Azért született meg, mert a MY-X-en futó hasonlóságelemző program elsődleges adatbeviteli módja a rangsorskálán alapul. A rangsorskála lényege, hogy az adott adatokat valamilyen rendezési irány szerint rangsorolja. Természetesen ez csak számokon működik. Logikája a következő: definiáljuk, hogy melyik számérték reprezentálja az adott adatvagyonban a legfontosabb értéket. A legfontosabb adatsor így megkapja az első pozíciót. A második pozíciót a valamivel gyengébb fontossági mutatóval rendelkező adatsor kapja meg. Tehát, az adatok fontosság alapján kapnak egy sorszámot. Mikor a rangsorskála kész van, akkor a kész listát a hasonlóságelemző programba betolva a rendszer elkezdhet elemezni.

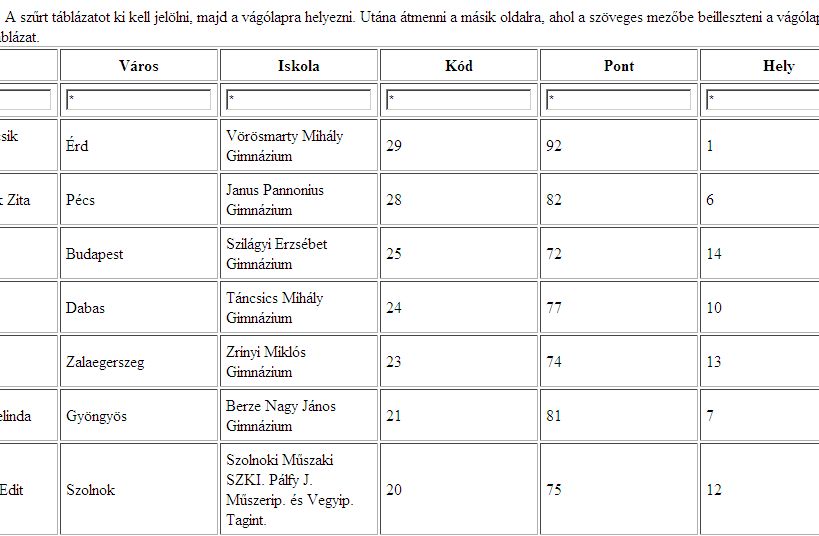
Sajnos ennek a funkciónak a megoldására egy másik modult kell a jelenlegihez csatolni, mert a szűrőt inicializáló oldalnak a bonyolultsága és komplexitása miatt eléggé zsúfolt és nehezen értelmezhető JavaScript kódot eredményezne.

Éppen ezért döntöttem úgy, hogy a problémának a megoldását két részre bontom. Az első rész alkotja az OLAP rendszerből kieső adatokon végrehajtható szűrést. A cél, hogy olyan állapot keletkezzen, amiből a rangsorskálát elő akarjuk állítani. Ez a modul az elsődleges tesztek és utólagos fejlesztések után a helyére került.

A második modul fogja a rangsorskálát felépíteni az első modulban leszűrt adatokból. A technológiai lépéseit ennek a folyamatnak a következő lista tartalmazza:

* az OLAP rendszerből keletkezett adatok leszűrése a felhasználó igényei szerint
* a kész táblázatot kijelölni és a vágólapra másolni
* megnyitni a rangsorskáláért felelős weboldalt (amire például egy hivatkozással megy át a felhasználó)
* azon az oldalon a meghatározott szövegbeviteli mezőbe beilleszteni a vágólapra helyezett szűrt tartalmat
* a modul a meghatározott módszerrel feldolgozza az adatokat és elkészíti a sorszámozást
* utolsó lépésben pedig az elkészített sorszámozott listát megjeleníti az oldal

A problémát a fenti lépéseken keresztül oldottam meg. Először is, le kell szűrni az adatokat a saját igényeink szerint. Miután a szűréssel végeztünk, meg kell határoznunk azt az oszlopot, aminek az értékei alapján elkészíttetjük a listát. Ezt a legkönnyebben úgy tehetjük meg, hogy a kiválasztott oszlop fejlécére az egérrel rákattintunk. Ekkor a szűrő sorba rendezi az adott oszlop értékei alapján a táblázatot. Ha egyet kattintunk, akkor növekvő skálát kapunk, ha kettőt, akkor csökkenőt (30. ábra).



30. ábra – részlet egy táblázatból, az adatok már a „Kód” oszlop alapján vannak rendezve csökkenő sorrendbe (saját ábrázolás)

Miután a szűrés és a rendezés véget ért, ki kell jelölni az egész táblázatot, majd a tartalmát a vágólapra helyezni (Ctrl+C billentyűkombináció), majd átmenni a táblázat tetején megtalálható oldalra a hiperhivatkozás segítségével (31. ábra).



31. ábra – a vágólapról beillesztett adat, ezt fogja a JavaScript feldolgozni (saját ábrázolás)

A függvény úgy dolgozik, hogy a beillesztett adatokat először feldarabolja és tömböket gyárt belőle. A sorokat az „új sor” (\n) karakter alapján különíti el, a cellákat pedig a tabulátor jel (\t) segítségével.

Ezek után egy ciklussal végigiterál a létrehozott tömbön és újra összerakja azt, immáron sorszámozva (32. ábra).



32. ábra – a sorszámozott táblázat a bemenő adatok alapján (az első oszlopban látható az adott sorszám) (saját ábrázolás)

Végül a kész táblázat tartalmát újra a vágólapra helyezve már lehet használni a MIAU szerverén található hasonlóságelemző modulban.

### Mutat/elrejt funkció

Ez a funkció részben kapcsolódik a gombra kattintással elkezdhető funkcióhoz. Ahogy annak, úgy ennek is többnyire az overhead jelenséghez kapcsolódik.

Lényege, hogy amikor a felhasználó gépe elkezdi fogadni az OLAP rendszerből érkező adatvagyont, könnyen megeshet – főleg ha nagy mennyiségről van szó – hogy a böngésző már rég elkezdi feldolgozni a HTML kódot, pedig az teljes egészében még nem érkezik meg. Ennek a kiküszöbölésére született meg ez a képesség, hogy az inicializálás során a JavaScript kód alapértelmezetten elrejti az összes sort és csak az első keresés elindítása után jelenik meg az adat.

Kézenfekvő megoldás volt ennél az igénynél is, hogy ezt a képességét a programnak is szabályozni tudjuk a konfigurációs blokkból (33. ábra).

rowDisplay: true

33. ábra – a sorok mutatásáért vagy elrejtéséért felelős attribútum (saját árázolás)

Látható, hogy a gomb alapú keresést beállító attribútumhoz hasonlóan, ez is logikai változó.

# Következtetések

Alapvetően úgy gondolom, hogy a rendszer fejlesztése során sikerült olyat alkotnom, amit a jelenlegi BP-k nem tudnak. Természetesen nem azért nem tudják, mert nehéz lenne kivitelezni (a dolgozat témájából láthatjuk, hogy nem az), hanem azért, mert eddig még senkinek, nem volt rá igénye, hogy egy ilyen rendszer megszülessen.

Sikerült az általam és a tesztelők által támasztott igényeket kielégíteni ráadásul úgy, hogy nem volt szükség táblázatkezelő rendszerre sehol az adott folyamatban. A kényelmi funkciók nagyon jól működnek (üres sorok, gombra induló keresés), a manipulációs lehetőségek kiszélesítésével ténylegesen gyorsítottam az adatok feldolgozási sebességén.

Véleményen szerint az alkalmazásban, illetve magában a módszerben is komoly lehetőségek rejlenek. Beszéltem már arról, hogy jelenleg a hallgatók szemszögéből nézve miért nehéz adott témakörben szakpolitizálni. Ennek a módszernek a segítségével, ha nem is megváltani, de mindenképpen segíteni lehetne őket. Számos weboldal, ha implementálná a fenti oldalakon kifejtett módszert, akkor az adattárházak által birtokolt adathalmazokhoz való hozzáférés nagyban leegyszerűsödne.

A vállalati szektor területein is lehetne létjogosultsága. Sőt, az alkalmazást tovább fejlesztve akár önálló alkalmazásként is eladható lenne, melynek segítségével akár kész jelentések is készíthetőek lennének.

Ugyancsak a továbbfejlesztési lehetőségek közé tartozhat a jobb testreszabhatóság lehetősége, mondjuk valamilyen konfigurációs fájl alkalmazásával, amelyet az első alkalmazásakor hozná létre pár kérdés után a program.

A webes lét következményeként egy kevés hozzáfejlesztés után – mondjuk egy API-t (Alkalmazásprogramozási felület - Wikipédia, 2004) megalkotva – elősegíthető lenne az alkalmazás, más technológiákhoz történő integrációja.

Gyakorlatilag a lehetőségek száma végtelen. Egyetlen kérdést kell csak feltenni: van-e rá igény a piacon? Ha igen, akkor a második kérdés pedig az, hogy milyen ár-érték arányszámokkal érdemes csinálni? Lényegében, mennyit ér az általa előállított információ?

# Összefoglalás

Szerte a világon akármilyen témakörben is dolgozunk, mindenhol találkozni fogunk online analitikus eszközökkel (OLAP) vagy jelentésekkel, melyek az adott témakörben többdimenziós lekérdezéseket fognak magukba foglalni. Bárkivel előfordulhat, hogy az adott számítógépen, az adott környezetben kell ilyen jelentésekkel dolgozni, de nem biztos, hogy rendelkezésre áll valamilyen táblázatkezelő szoftver (Microsoft Office, Open Office, stb.).

Az alkalmazást megalkotásával párhuzamosan haladva megmutattam, hogy mik voltak azok a körülmények, tények, lehetőségek és eszközök, amiket alkalmazva és/vagy figyelembe véve az alkalmazás megszületett.

Láthattuk, hogy a – technológiának köszönhetően – a felhasználási terület igen széles körben mozog.

Azt is láthattuk, hogy a teljes alkalmazás – bár még a tudása elég kezdetleges – már most nagyon sok weboldal adatainak a szűrésében részt vehetne, hiszen implementálása rendkívül egyszerű és gyors.

Összességében pedig: mindenképpen megérte azt az időt, amit eddig ráfordítottam. Azt kaptam, amit kapni szerettem volna és olyan formában, amiben szerettem volna. Már most van olyan weboldal (http://miau.gau.hu/), ahol ebben az állapotában alkalmazzák, a visszajelzések pedig egytől-egyig azt mondják, hogy megérte és könnyítette a keresést az adathalmazban.

A keresés és a tesztek során azt is láttuk, hogy szavakra, számokra, jelekre, de akár szótöredékekre is lehet keresni, a különféle speciális vezérlőkarakterek esetében pedig már most lehetőségünk van speciális szűrés megalkotására (tagadó és joker karakter, kisebb-nagyobb-egyenlő értékek).

Bár ott még az alkalmazás fejlesztési állapota nem tart, hogy esetleg a joker karaktert a tagadó karakterrel együtt használva tudjunk szűrni, de akár ez is lehet egy továbbfejlesztési lehetőség – a sok többi között.

Összességében úgy gondolom, hogy az általa nyújtott lehetőségek és az eredményekből kiindulva megérné a továbbfejlesztést.

# Irodalomjegyzék

*Alkalmazásprogramozási felület - Wikipédia*. (2004. február 1). Letöltés dátuma: 2010. szeptember 27, forrás: Wikipédia: http://hu.wikipedia.org/wiki/API

*Data mining - Wikipedia, the free encyclopedia*. (2002. február 28). Letöltés dátuma: 2010. augusztus 9, forrás: Wikipedia: http://en.wikipedia.org/wiki/Data\_mining

*File Transfer Protocol - Wikipédia*. (2005. február 13). Letöltés dátuma: 2010. augusztus 9, forrás: Wikipédia: http://hu.wikipedia.org/wiki/FTP

*FlavorZoom*. (2010. szeptember 29). Letöltés dátuma: 2010. szemptember 29, forrás: FlavorZoom: http://projects.gregweber.info/demo/flavorzoom.html

Hendrix, N. (2009. június 10). *Using jQuery To Manipulate and Filter Data*. Letöltés dátuma: 2010. augusztus 29, forrás: Nettuts+: http://net.tutsplus.com/tutorials/javascript-ajax/using-jquery-to-manipulate-and-filter-data/

*HTML - Wikipédia*. (2004. június 10). Letöltés dátuma: 2010. augusztus 9, forrás: Wikipédia: http://hu.wikipedia.org/wiki/Html

*HTTP - Hypertext Transfer Protocol Overview*. (2010. augusztus 9). Letöltés dátuma: 2010. szeptember 1, forrás: World Wide Web Consortium (W3C): http://www.w3.org/Protocols/

*https - Wikipédia*. (2007. december 28). Letöltés dátuma: 2010. augusztus 9, forrás: Wikipédia: http://hu.wikipedia.org/wiki/Https

Ifj. Zettner, T. (2008. december 11). *Firefox 3.1: gyors, gyors, de nem eléggé - IT café Szoftver hír*. Letöltés dátuma: 2009. október 24, forrás: ITCafé: http://itcafe.hu/hir/firefox\_3\_1\_gyors\_gyors\_de\_nem\_elegge.html

*Java (programozási nyelv) - Wikipédia*. (2004. június 8). Letöltés dátuma: 2010. augusztus 9, forrás: Wikipédia: http://hu.wikipedia.org/wiki/Java\_%28programoz%C3%A1si\_nyelv%29

*JavaScript - Wikipédia*. (2004. november 25). Letöltés dátuma: 2009. október 23, forrás: Wikipédia: http://hu.wikipedia.org/wiki/Javascript

*jQuery: The Write Less, Do More, JavaScript Library*. (2009. október 24). Letöltés dátuma: 2010. augusztus 9, forrás: jQuery: The Write Less, Do More, JavaScript Library: http://jquery.com/

Loyola, R. (2010. június 10). *Benchmarks: Safari 5 shows JavaScript boost*. Letöltés dátuma: 2010. szeptember 7, forrás: Macworld: http://www.macworld.com/article/151140/2010/06/safari5\_speed.html

*MIAU: 1998-2010: Magyar Internetes Alkalmazott/Agrár Informatikai Újság*. (1988. szeptember 1). Letöltés dátuma: 2010. szeptember 27, forrás: MIAU: 1998-2010: Magyar Internetes Alkalmazott/Agrár Informatikai Újság: http://miau.gau.hu/miau2009/index.php3

*MY-X*. (2010. március 20). Letöltés dátuma: 2010. szeptember 26, forrás: MIAU: http://miau.gau.hu/myx-free/olap/curie/curie\_2009.htm

*Nemzeti Kutatási és Technológiai Hivatal*. (2010. szeptember 27). Letöltés dátuma: 2010. augusztus 9, forrás: Nemzeti Kutatási és Technológiai Hivatal: http://www.nkth.gov.hu/

*Online analytical processing - Wikipedia, the free encyclopedia*. (2003. február 27). Letöltés dátuma: 2010. augusztus 9, forrás: Wikipedia: http://en.wikipedia.org/wiki/Olap

*Online and offline - Wikipedia, the free encyclopedia*. (2002. február 25). Letöltés dátuma: 2010. augusztus 9, forrás: Wikipedia: http://en.wikipedia.org/wiki/Online\_and\_offline

*PHP - Wikipédia*. (2004. szeptember 21). Letöltés dátuma: 2010. augusztus 9, forrás: Wikipédia: http://hu.wikipedia.org/wiki/PHP

*PicNet Table Filter Demo*. (2010. október 29). Letöltés dátuma: 2010. október 29, forrás: PicNet IT Services Sydney: http://www.picnet.com.au/resources/tablefilter/demo.htm

Stachowiak, M. (2010. április 7). *Surfin' Safari - Blog Archive - Announcing SunSpider 0.9.1*. Letöltés dátuma: 2010. június 17, forrás: Surfin' Safari: http://webkit.org/blog/1046/announcing-sunspider-0-9-1/

*Standards Only (sorted by date) - W3C*. (2009. október 23). Letöltés dátuma: 2010. augusztus 9, forrás: World Wide Web Consortium (W3C): http://www.w3.org/TR/tr-date-stds.html

*SunSpider JavaScript Benchmark*. (2009. október 24). Letöltés dátuma: 2009. október 24, forrás: SunSpider JavaScript Benchmark: http://www2.webkit.org/perf/sunspider-0.9/sunspider.html

*The Idea Mill*. (2009. október 1). Letöltés dátuma: 2010. október 29, forrás: jQuery tableFilter normal table test: http://ideamill.synaptixgroup.com/jquery/tablefilter/tabletest.htm

*Világháló - Wikipédia*. (2004. november 6). Letöltés dátuma: 2010. augusztus 9, forrás: Wikipédia: http://hu.wikipedia.org/wiki/Vil%C3%A1gh%C3%A1l%C3%B3

*Web browser - Wikipedia, the free encyclopedia*. (2001. november 2). Letöltés dátuma: 2010. augusztus 9, forrás: Wikipedia: http://en.wikipedia.org/wiki/Web\_browser

# Ábrajegyzék

1. táblázat – a BP jelenlegi tudásának összehasonlítása a fejlesztés alatt álló rendszerrel (saját ábrázolás) 5

1. ábra - a böngészők JavaScript feldolgozásának sebessége. Az Y-tengelyen az adott böngésző neve és verziószáma, míg az X-tengelyen a SunSpider benchmark program által mért összpontszám olvasható. A kisebb érték a jobb. (Ifj. Zettner, 2008) 15

2. ábra - a böngészők JavaScript feldolgozási sebessége 2010 tavaszán. A tesztprogram által adott pontszám látható a képen (a kisebb érték a jobb). (Loyola, 2010) 15

3. ábra – első lépésként a böngésző megkapja az adatokat a web-kiszolgálótól (saját ábrázolás) 19

4. ábra – második lépésként a felhasználó elvégzi, a szükséges műveleteket majd visszaküldi az adatokat a web-kiszolgálónak (saját ábrázolás) 19

5. ábra – a web-kiszolgáló alkalmazza a kért változtatásokat, újra generálja a HTML kódot és visszaküldi a böngészőnek a kész tartalommal (saját ábrázolás) 19

6. ábra – a kapcsolat nélküli megoldás. Magára a kapcsolatra csak addig van szükség, amíg az adatok meg nem érkeznek a böngészőbe (saját ábrázolás) 21

7. ábra – részlet egy OLAP táblából a http://miau.gau.hu/myx-free/olap/olap2/2\_olap\_m.php3 oldalról 23

8. ábra – egy táblázat HTML forráskódja (saját ábrázolás) 25

9. ábra – Megjelenített táblázat a web böngészőben (saját ábrázolás) 26

10. ábra - HTML forráskód, ami tartalmazza a JavaScript fájl hivatkozását és egy paragrafust, valamint egy gombot (saját ábrázolás) 27

11. ábra – a 10. ábrán látható forráskód megjelenítve a böngészőben (saját ábrázolás) 28

12. ábra – a JavaScript függvény forráskódja (saját ábrázolás) 28

13. ábra – módosított betűméret a script lefutása után (saját ábrázolás) 28

2. táblázat – a fájlok leírását tartalmazó táblázat (saját ábrázolás) 30

14. ábra – a HTML forráskódban található objektumhivatkozások (saját ábrázolás) 30

15. ábra – a modulárisan telepíthető szűrő forráskódja (saját ábrázolás) 32

3. táblázat – beállítások részletes leírása (saját ábrázolás) 33

16. ábra – a szűrő inicializálását elvégző script (saját ábrázolás) 34

4. táblázat – adott HTML elem azonosítási módszerei (saját ábrázolás) 34

17. ábra - példa program a láncolhatóság szemléltetésére (saját ábrázolás) 35

18. ábra - a definiált táblázaton inicializálásra került a szűrő (saját ábrázolás) 36

19. ábra – a keresés mechanizmusa (saját ábrázolás) 37

5. táblázat – a tesztek leírását tartalmazó táblázat (saját ábrázolás) 40

20. ábra – az induló adatvagyon, mely 7 oszlopban és 1000 sorban helyezkedik el (saját ábrázolás) 40

21. ábra – az első teszt eredményhalmaza a szűrés után (saját ábrázolás) 41

22. ábra – a második teszt szűrési eredménye (saját ábrázolás) 42

23. ábra - a harmadik teszt szűrési eredménye (saját ábrázolás) 43

24. ábra – a tesztek futási ideje egy közös grafikonon (saját mérés és ábrázolás) 44

25. ábra – új változók létrehozása a globális konfigurációban (saját ábrázolás) 46

26. ábra – a gomb alapú keresést beállító attribútum (saját ábrázolás) 48

27. ábra – a keresőmezőkre vonatkozó stíluslap definíció (saját ábrázolás) 49

28. ábra – a szükséges HTML specifikus attribútum CSS szabályok használatához (saját ábrázolás) 49

29. ábra – hiányzik a „class” attribútum így nincs CSS szabály alkalmazása (saját ábrázolás) 50

30. ábra – részlet egy táblázatból, az adatok már a „Kód” oszlop alapján vannak rendezve csökkenő sorrendbe (saját ábrázolás) 51

31. ábra – a vágólapról beillesztett adat, ezt fogja a JavaScript feldolgozni (saját ábrázolás) 52

32. ábra – a sorszámozott táblázat a bemenő adatok alapján (az első oszlopban látható az adott sorszám) (saját ábrázolás) 52

33. ábra – a sorok mutatásáért vagy elrejtéséért felelős attribútum (saját árázolás) 53

# Fogalomtár

**API:** olyan felület vagy interfész, amely lehetőséget ad valamilyen „közös nevező” formában a kommunikációt más programokkal. Ezek általában kész függvények, melyeknek megvan a saját funkciója és csak alkalmazni kell őket.

**Adatbányászati technológiák:** (data-mining technologies) olyan technológiák gyűjtőneve, amikkel sok adatból információt állít elő. Napjainkban egyre fontosabb szakterület.

**Benchmark:** A számítástechnikában benchmarking szó alatt azt értjük, amikor különféle teszteket futtatunk egy adott program, vagy objektum teljesítményének megállapítása érdekében.

**FTP:** olyan protokoll, mely lehetővé teszi fájlok módosítását illetve cseréjét TCP/IP alapú hálózatokon. Például az Internet is TCP/IP alapú hálózat.

**HTML:** egy leíró nyelv, melyet weboldalak készítéséhez fejlesztettek ki, és mára már internetes szabvánnyá vált a W3C támogatásával.

**HTTPS:** kombinálták a HTTP protokollt az SSL/TLS protokollal, így a protokollon áthaladó adat végig titkosítva mozog.

**Java:** a Java egy objektumorientált programozási nyelv, amit a Sun Microsystems kezdett el fejleszteni a 90-es években. Fejlesztése a mai napig tart, számos célterületen remekül szolgál.

**JavaScript:** A JavaScript programozási nyelv egy objektumalapú szkript nyelv, amelyet weblapokon elterjedten használnak. A JavaScriptet először 1997–99 között szabványosította az ECMA „ECMAScript” néven. A jelenleg is érvényes szabvány az ECMA-262 Edition 3 (1999. december), ami a JavaScript 1.5-nek felel meg. Ez a szabvány egyben ISO szabvány is.

**OLAP:** olyan megközelítési módszer, amely segítségével több-dimenziós lekérdezéseket hajthatunk végre. Eredménye általában valamilyen mátrix vagy pivot táblázat.

**PHP:** nyílt forráskódú szkript nyelv, melynek alapjait Rasmus Lerdorf fektette le 1994-ben. Mára már a PHP Group keze alatt működik, többnyire dinamikus weboldalak megalkotására használják a mai napig.

**CSS:** olyan leíró fájl, mely meghatározza azoknak az elemeknek a megjelenését, amelyek rendelkeznek CSS hivatkozással.

**Overhead jelenség:** az informatikában overhead-nek nevezik azt a jelenséget, amikor valamilyen úton-módon az adott feladat elvégzéséhez indokolatlanul nagy erőforrásigényre van szükség, legyen szó sávszélességről, számítási kapacitásról, vagy időről. Általában az overhead-et strukturált, jól átgondolt program vagy programrészlet, megoldási módszerrel lehet megszűntetni.

# Rövidítések

**API** – **A**pplication **P**rogramming **I**nterface

**FTP** – **F**ile **T**ransfer **P**rotocol

**HTML** – **H**yper**T**ext **M**arkup **L**anguage

**HTTP** – **H**yper**T**ext **T**ransfer **P**rotocol

**HTTPS** – **H**yper**T**ext **T**ransfer **P**rotocol **S**ecure

**NKTH** – **N**emzeti **K**utatási és **T**echnológiai **H**ivatal

**OLAP** – **O**n**L**ine **A**nalytical **P**rocessing

**PHP** – **H**ypertext **P**re-**P**rocessor

**SSL** – **S**ecure **S**ockets **L**ayer

**TCP/IP** – **T**ransmission **C**ontrol **P**rotocol/**I**nternet **P**rotocol

**TLS** – **T**ransport **L**ayer **S**ecurity

**URI** – **U**niform **R**esource **I**dentifier

**W3C** – **W**orld **W**ide **W**eb **C**onsortium

**CSS** – **C**ascading **S**tyle **S**heet

# Függelékek

**Konzultációkon való részvétel igazolása**

**A hallgató neve:**

Bures Tamás

**A belső konzulens neve és beosztása:**

Pitlik László, egyetemi docens, SZIE GTK TKI

**A témát kiadó önálló oktatási szervezeti egység neve:**

Szent István Egyetem Gazdaság- és Társadalomtudományi Kar, TATA Kiválósági Központ és Informatikai Intézet

Nevezett hallgató a 20.../20... tanévben a diplomamunka készítésével kapcsolatos konzultációkon rendszeresen részt vett. Az elkészített dolgozatot

**„Auto-szűrő fejlesztése OLAP jelentések utólagos, offline tovább-feldolgozására”**

címmel bemutatta. A dolgozatnak a Záróvizsgához kapcsolódó bírálati eljárásra való beadásával egyetértek.

Gödöllő, 20... év ........................... hó .................. nap

.......................................

konzulens aláírása

**Nyilatkozat**

Alulírott …………………………………………………………. a Szent István Egyetem Gazdaság- és Társadalomtudományi Kar ……………………………………………….. ...szak

..................................................szakirány .........................tagozat végzős hallgatója nyilatkozom, hogy a ..................................................................………………………………………………. …………………………………………………………………………………………………. címmel védésre benyújtott diplomadolgozat/szakdolgozat saját munkám eredménye, amelynek elkészítése során a felhasznált irodalmat a szerzői jogi szabályoknak megfelelően kezeltem.

Gödöllő, 20… év …………… hó ……… nap

(a hallgató aláírása)

**Diplomadolgozat/Szakdolgozat rövid bemutatása**

**A diplomaterv készítőjének neve:** Bures Tamás

**A diplomaterv címe:**

Auto-szűrő fejlesztése OLAP jelentések utólagos, offline tovább-feldolgozására

**A témát kiadó önálló szervezeti egység neve:**

Szent István Egyetem Gazdaság- és Társadalomtudományi Kar,TATA Kiválósági Központ és Informatikai Intézet

**A belső konzulens neve és beosztása:**

Pitlik László, egyetemi docens, SZIE GTK TKI

**Kulcskifejezések:**

OLAP, JavaScript, utólagos feldolgozás, szűrés-rendezés, böngésző

**A dolgozat rövid leírása:**

A dolgozat egy adott probléma megoldásáról szól. Ez a fejlesztés egy olyan böngészőben futó alkalmazást eredményezett, mely rendkívül rugalmasan illeszthető gyakorlatilag bármilyen olyan rendszerre, ahol a felhasználó valamilyen táblázattal, jelentésekkel dolgozik. Szűrés, rendezés és sorszámozás, ez a három célterület amire a JavaScript alkalmazás fókuszál.

1. **URI:** állandó erőforrás azonosító [↑](#footnote-ref-2)
2. **File:** fájl – a „file:” prefixum azt mondja meg a böngészőnek, hogy az utána következő tartalom a számítógép helyi háttértárán található. Ez kerül használatra, amikor egy weboldalt lementünk a saját gépünkre, majd onnan nyitjuk meg a tartalmat. [↑](#footnote-ref-3)
3. CSS: Cascading Style Sheet [↑](#footnote-ref-4)