

Szegedi Tudományegyetem
Informatikai Tanszékcsoport

**Minőségkinyerés borkóstolási adatokból, web és
android alkalmazás fejlesztés**

Szakdolgozat

Készítette:
Varga Tamás
Programtervező Informatikus
hallgató

Témavezető:
Dr. Csendes Tibor
tanszékvezető egyetemi tanár

Szeged
2015

Tartalomjegyzék

Feladatkiírás	4
Tartalmi összefoglaló	5
Bevezetés	6
1. Borkóstoló algoritmusok	7
1.1. CoHITS	7
1.2. Hamming	9
1.3. Koszinusz	9
1.4. Precedencia	9
1.5. Összefüggőségi	9
2. A weboldal	11
2.1. Iterációk	12
2.1.1. Első mérföldkő	12
2.1.2. Második mérföldkő	12
2.1.3. Harmadik mérföldkő	13
2.1.4. Negyedik mérföldkő	14
2.1.5. Ötödik mérföldkő	14
2.1.6. Hatodik mérföldkő	14
2.1.7. Hetedik mérföldkő	15
2.2. Adatbázis	15
2.2.1. scores tábla	16
2.2.2. wines tábla	17
2.2.3. users tábla	17
2.3. CSS	19
2.4. PHP	21
2.4.1. PHP Data Objects (PDO)	23
2.4.2. Állapotkezelés és követés	25
2.5. JavaScript	27
2.5.1. AJAX	28
2.5.2. jQuery	29
2.5.3. Numeric Javascript	30
2.6. Grafikonok	30
2.6.1. Google Charts	31
2.7. Statikus tartalom	32
2.8. Dinamikus tartalom	33
2.8.1. php-login	33

2.8.2. demo.php - Demó	33
3. A mobil alkalmazás	34
3.1. Android	34
4. A weboldal és a mobil alkalmazás összefűzése	35
5. Tesztelés	36
5.1. Regisztrációs és bejelentkeztető rendszer	36
5.2. Demo adatkezelésének ellenőrzése	36
5.3. Algoritmusok ellenőrzése kis adatokon	36
5.4. Algoritmusok ellenőrzése ismert eredményekkel	36
6. Összefoglalás	37
7. Egyebek	38
7.1. Verziókezelés	38
7.1.1. Git	38
7.1.2. GitHub	38
7.1.3. A választás	39
7.1.4. Repozitóriumok	39
7.2. Környezetek	39
8. Függelék	41
8.1. A program forráskódja	41
Nyilatkozat	42
Irodalomjegyzék	43

Feladatkiírás

A hallgató feladata egy olyan web és hozzá tartozó android alkalmazás készítése, amely képes borkóstolás során gyűjtött bor értékelések alapján több különböző algoritmus használatával a kóstolók rangsorolására. A web és android alkalmazásoknak képeseknek kell lenniük egymás közötti szinkronizációra, több felhasználó kezelésére valamint a bevitt adatok és az alkalmazás állapotaink tárolására. Az android alkalmazásnak offline módban is használható kell hogy legyen.

Tartalmi összefoglaló

A téma megnevezése ♦ Minőségkinyerés borkóstolási adatokból, web és android alkalmazás fejlesztés

A feladat megfogalmazása ♦ Web és android alkalmazás fejlesztése, amely képes bor értékelések alapján már létező algoritmusok által rangsorolni a borkóstolókat. A mobil alkalmazás felhasználó barát megvalósítása, mely bármilyen körülmény mellett valós idejű adat bevitelt is garantál.

A megoldási mód ♦ A web alkalmazás a következő címekről érhető el:

– `http://bor.tvarga.hu`

Az android alkalmazás letölthető bármelyik fent említett URL -ről a jobb oldali floating menün keresztül, vagy az alábbi közvetlen linken: `borkostolas.apk`

Alkalmazott eszközök, módszerek ♦

– Web alkalmazás:

Programozási nyelvek: PHP, JavaScript, MySQL

Fejlesztői környezetek: SublimeText, PhpStorm

– Android alkalmazás:

Programozási nyelvek: Java

Fejlesztői környezetek: Android Studio

– Verziókezelés: GitHub

Kulcsszavak ♦ Android, PHP, JavaScript, Borkóstolás

Bevezetés

Borkóstolók rangsorolásával illetve minőségkinyeréssel valamint borversenyek lebonyolításával már előttem is többen foglalkoztak. Bár még van kutatni való ezen a területen, jól működő algoritmus már több is ismert ilyen jellegű feladatok megoldására. Ezért ennek a szakdolgozatnak nem is újabb algoritmusok kifejlesztése vagy már meglévők továbbfejlesztése a fő célja. Mivel található több olyan web alkalmazás is, mely egy specifikus egyszerű algoritmust vagy talán egy sokat tesztelt komplexebbet használ, mégsem létezik egy közismert, modern és felhasználóbarát alkalmazás, amely egy igényes felhasználói felületen keresztül bemutatná ezen algoritmusok működését lehetővé téve hogy nyomon kövessük akár saját laikus eredményeinket neves szakértőkhöz viszonyítva.

Mivel a mai világban a mobil eszközök egyre nagyobb teret hódítanak, az is célt volt hogy egy olyan mobil (android) alkalmazás társuljon a web alkalmazáshoz mely egy könnyen kezelhető intuitív felületen keresztül lehetővé tegye a rendszer használatát, bizonyos funkciókat még offline üzemmódban is elérhetővé téve, kielégítve napjaink átlagos felhasználóinak igényeit.

A dolgozat elején röviden bemutatom az alkalmazás által használt algoritmusokat. Majd rátérek a weboldal és a mögötte álló rendszerek felépítésére, a hozzájuk társuló fejlesztői környezetek és eszközök rövid ismertetésére. Ezt majd a mobil (android) alkalmazás bemutatás követ, melynél szintén kitérek a rendszer felépítésére illetve a megvalósítást lehetővé tevő eszközökre, erőforrásokra. Logikusan ezt a két alkalmazási felület (web és mobil) összefűzésének megvalósítása követi. Az összefoglalás előtt néhány teszten keresztül bemutatom a rendszer megbízhatóságát. Legvégül pedig kitérek néhány olyan dologra mely segítette a munkámat, mint például verziókezelő rendszerek használata.

1. fejezet

Borkóstoló algoritmusok

Ezen szakdolgozat keretében a CoHITS algoritmus került implementálásra, mely mellé csatlakozott néhány Borkóstolás projekt [1] keretében használt már Szilárd Iván által implementált algoritmus, mint a Hamming, Koszinusz, Precedencia, Összefüggősségi. Ezeknél az algoritmusoknál a PageRank egy hasonlósági mátrixon propagál, ennek elemeit a borkóstolók páronkénti értékeléseinek inverz távolságából kapjuk a hozzájuk illő távolság mértékek alkalmazásával (Hamming távolság, koszinusz távolság, precedencia távolság, összefüggősségi távolság) [1].

1.1. CoHITS

A CoHITS algoritmus a PageRank és a HITS algoritmuson egy kiterjesztett változata [5]. A PageRank algoritmust Sergey Brin és Larry Page fejlesztette ki a Google kereső használatához [2] keresési találatok fontossági rangsorolásnak meghatározása céljából. Tőlük függetlenül Jon Kleinberg egy hasonló koncepcióval állt elő mely ugyan ilyen jellegű feladatot látott el [3].

Legyen X és Y a kóstolók és borok. Ugyan abból a p^0 értékből indulunk ki minden $x_i \in X$ kóstolónál. Ekkor legyen $w'(\overrightarrow{x_i y_j})$ az az értékelés amit y_j bor kapott az x_i kóstolótól és legyen $w(\overrightarrow{x_i y_j}) = w'(\overrightarrow{x_i y_j}) / \sum_{j \in Y} w'(\overrightarrow{x_i y_j})$ a normalizáltja. Továbbá legyen q_j^0 érték (az y_j borra) az értékelések átlaga. Majd definiáljuk a $w(\overleftarrow{x_i y_j})$ az alábbi képen: tegyük fel hogy y_j bort l különböző kóstoló kóstolta és legyen

$$D = \sum_{i \in X} |q_j^0 - w'(\overrightarrow{x_i y_j})|, \quad (1.1)$$

az összegek különbsége az átlagtól az y_j borra tekintve. Végül, legyen

$$w(\overleftarrow{x_i y_j}) = \frac{|\sum_{i \in X} |q_j^0 - w'(\overrightarrow{x_i y_j})||}{(l-1) D} \quad (1.2)$$

Ekkor $\sum_{i \in X} w(\overleftarrow{x_i y_j}) = 1$ tehát minden $w(\overleftarrow{x_i y_j})$ súlyra lehet úgy tekinteni mint egy átmeneti valószínűség y_j -ből x_i -be. Az 1.1 ábra a súlyok kiszámítására mutat egy konkrét példát.

A két kóstoló közti súly x_i és x_j úgy definiálható mint egy rejtett átmeneti valószínűség 1 által definiálva. Ekkor a $p = W_{xp}$ HITS egyenlet $p = (p_1, p_2, \dots, p_m)$ megoldás megadja a kóstolók rangsorát.



1.1. ábra. A gráf súlyai amikor a Kóstoló 1 a 20, 30 és 70 értékeléseket adott a Bor 1, Bor 2 és Bor 3 borokra, valamint a Bor 1 a 20, 30 és 70 értékeléseket kapott a Kóstoló 1, Kóstoló 2 és Kóstoló 3 aktól.

Tehát látható, hogy az így elő állított algoritmust alkalmazni lehet egy borkóstolási gráfra kóstolók rangsorolásának céljából az ő képességeik és szakmai hozzáértésük alapján. Általánosságban az figyelhető meg, hogy az így előálló rangsor jobban teljesít mint más egyszerű statisztikai algoritmusok. Ki tudja szűrni a hozzá nem értő kóstolókat [5].

1.2. Hamming

A Hamming-távolságot (1.3) többnyire vektorok, karaktersorozatok valamint bitminták közötti eltérések kimutatására használják. A Hamming-távolság csak azt mondja meg, hogy hány helyen nem illeszkedik az egyik minta a másikra. Legyen S_1 és S_2 két azonos hosszú minta.

$$D_{Hamming}(S_1, S_2) = \sum_{p=0}^{|S_1|} S_1(p) \neq S_2(p) \quad (1.3)$$

Ez a távolság nem lesz megfelelő mérték vizsgálathoz mivel nem túl informatív.

1.3. Koszinusz

Legyen π^1 és π^2 két permutáció-vektor. Ekkor ezek koszinusz távolságát az (1.4) képlet szerint számíthatjuk ki, ahol $\|\cdot\|$ az euklideszi normát jelöli.

$$D_{cos}(\pi^1, \pi^2) = 1 - \frac{\pi^1 \cdot \pi^2}{\|\pi^1\| \cdot \|\pi^2\|} \quad (1.4)$$

1.4. Precedencia

A precedencia távolság meghatározásához definiálnunk kell először a precedencia mátrix fogalmát. Egy π , n hosszúságú permutáció precedencia mátrixa egy olyan bináris elemekből álló $n \times n$ -es P mátrix, melyben $P_{\pi_i \pi_j} = 1, i < j$ egyébként $P_{ij} = 0$, azaz a permutáció elemeinek sorrendjét kódoljuk egy bináris mátrixszal. A precedencia távolságot két π^1 és π^2 permutáció között (P_1 és P_2 a nekik megfelelő precedencia mátrix) a precedencia mátrixuk egyező nem nulla elemeinek számából számíthatjuk ki (1.5) szerint.

$$D_{prec}(\pi^1, \pi^2) = \frac{n(n-1)}{2} - \sum_{i=1}^n \sum_{j=1}^n P_{ij}^1 P_{ij}^2 \quad (1.5)$$

1.5. Összefüggősségi

Az összefüggősségi távolság a szomszédos elemek azonos előfordulásából számítható. Legyen a két permutációnk π^1 és π^2 , n hosszúságúak, és a szomszédos elemek azonos előfordulásának száma n_{adj} .

Szemléletesen: Legyen N_π a π permutáció összefüggősségi mátrixa. Kezdetben minden eleme legyen 0, majd a π permutáció minden i elemére az $N_{\pi i}(\pi_i, \pi_i + 1) = 1$. A π^1

és π^2 permutációra kiszámoljuk az N_{π^1} és N_{π^2} mátrixokat. n_{adj} az N_{π^1} és N_{π^2} mátrixban azonos cellában lévő 1-esek száma.

Ekkor az összefüggősségi távolság (1.6) szerint számítandó.

$$D_{adj}(\pi^1, \pi^2) = n - n_{adj} - 1 \quad (1.6)$$

[1]

2. fejezet

A weboldal

A web oldal kialakításánál számomra fontos szempont volt, hogy az egy letisztult könnyen kezelhető felületet prezentáljon a felhasználók felé. Ennek érdekében már a kezdetektől folyton szem előtt tartottam hogy a stilisztikai elemek hogyan viszonyulnak egymáshoz, illetve külön figyelmet fordítottam arra, hogy a CSS (Cascading Style Sheets) konzisztens megjelenést biztosítson a web alkalmazás minden oldalán. Ez azért is jó döntés volt, mert így sokkal könnyebb volt kisebb finomításokat végezni, mint ha a stilisztikai definíciók közvetlen a PHP fájlokban található HTML tagokban szerepeltek volna.

A web alkalmazás alapvetően három részből épül fel:

- Statikus oldalak
- Dinamikus oldalak
- Az android alkalmazást ellátó szolgáltatások

Státikus oldalnak akkor nevezünk egy weblapot, ha annak a tartalma nem változik. Attól is független, hogy a látogató kicsoda, hanyadszor jelentkezik be vagy hogy honnan jelentkezik be stb..., tehát a tartalom konzisztens minden egyes látogatás alkalmával. Ez a megközelítés főképp akkor előnyös ha valami olyan tartalmat akarunk nyújtani a felhasználóknak amin egyáltalán nem áll szándékunkban változtatni, vagy csak nagyon ritkán.

Dinamikus oldalakról akkor beszélünk amikor valamilyen feltétel vagy beavatkozás közvetlenül befolyásolja azt hogy a felhasználó által megtekintett tartalom pontosan micsoda is. Legtöbbször akkor használatos, ha egy adatbázis tartalmát akarjuk megjeleníteni, vagy olyan felületet hozunk létre amely interakciót tesz lehetővé a felhasználó és a web alkalmazás között. Rengeteg programozás nyelv létezik ennek a célnak az ellátására. Én a PHP-t és a JavaScript-et választottam.

Az android alkalmazást szolgáltatásokkal ellátó rész fő feladata hogy a szerveren található adatbázishoz elérést biztosítson, mivel android alkalmazások nem tudnak közvetlenül csatlakozni egy távoli adatbázishoz, ezért kell ez a közbülső réteg amely kiszolgálja a mobil eszköz kéréseit, illetve olyan tartalmak elérését teszi lehetővé amelyek közvetlenül valamilyen limitációs okokból nem lennének elérhetőek. Sokszor ezeket egy böngészőből megnyitva nem látunk semmit, ez teljesen természetes, mivel itt a hangsúly nem azon van hogy emberek számára értelmezhető/olvasható tartalmat biztosítsunk, hanem fő célunk a web és a mobil alkalmazás közötti kommunikációs csatorna megteremtése, tehát egy olyan környezet ahol csak gépek beszélnek egymással.

2.1. Iterációk

Az oldal megalkotásához egy agilis iterációs dizájn megközelítést alkalmaztam, melyen során egy rövid kezdeti tervezés után elkezdtem a tényleges munkát ami abból állt, hogy implementáltam az adott feladatot, majd az így elkészült munkát tesztelve prezentáltam a megrendelőnek (konzulensnek) A következő lépés a tesztek során előkerülő hibák megbeszélése és azok javításának lehetséges orvosolása valamint az alkalmazás új funkciókkal való bővítése volt. Ez az utolsó három lépés, tehát az implementálás, tesztelés és hibajavítás valamint további funkciókkal való bővítés képezte az iterációs dizájn megközelítesem ciklikus magját. Az iterációk során felhasznált eszközökre és technológiák ismertetésére majd csak az iterációs mérföldkövek ismertetése után fogok kitérni.

2.1.1. Első mérföldkő

Az első mérföldkő folyamán először megbeszéltem a megrendelővel hogy milyen elvárásokkal rendelkezek illetve vázoltam az általam tervezett munkamenetet. Továbbá kiválasztottam a fejlesztéshez szükséges fejlesztői környezeteket, valamint ezek és a használatukhoz elengedhetetlen eszközöket előkészítettem a projekt munkában való felhasználáshoz. Ekkor történt meg a felhasználói felület megjelenési terveinek elkészítése. Webszerver, domain és adatbázis létrehozás illetve telepítés. A kész oldalszerkezeti tervekkel 2.1 neki kezdtem a grafikus elemek vizualizációjának valamint implementáltam az oldalszerkezet statikus elemeit mint: Menü rendszer, fejléc, alsó információs sáv, kapcsolatok és egyéb hír csatornák megjelenítésért felelős elemeket.

2.1.2. Második mérföldkő

A második mérföldkő során került sor a megrendelőtől kapott dokumentumok és források feldolgozására majd azok feltöltése a projektbe főként statikus HTML elemekként.



2.1. ábra. Első oldalszerkezeti terv

Továbbá végrehajtása a megrendelő által kért grafikus és szerkezeti változtatásokat mint: Alsó információs sáv átmozgatása és re-faktorizációja egy oldalsó lebegő mozgó elemmé (*floating sidebar*), menüpontok átnevezése, fejléc logó frissítése, A *borkostolasEredmenyek* oldal átdolgozása az alábbi módon: Táblázatok kettéválasztása. Vízszintes görgető sáv hozzáadása a borok táblázat felső és alsó részéhez, melyek együtt mozognak egymással így megoldva a táblázat nagyságából fakadó kezelhetőségi problémát. Az előre láthatólag bekövetkező grafikai felépítés esetleges módosításának érdekében több stílus elemet is újabb osztályokba rendeztem, tehát egy átfogóbb re-faktorizáció történt mely mint később kiderült a további stilisztikai elemek hozzáadását és stilizálását is megkönnyítette.

2.1.3. Harmadik mérföldkő

A harmadik mérföldkő fő feladata az azt megelőző konzultáció során átbeszélt *demo* oldal elkészítése volt. Tehát itt került sor a CoHITS algoritmus megismerése a hozzá tartozó szakirodalom áttanulmányozására majd az algoritmus implementálásának megkezdésére a projektbe. Ezt követte a korreláció és átlagtól való átlagos eltérés számolásának implementálása, de mint később kiderült ezek az algoritmusok nem tűrték jól a hiányos bemeneti adatokat ezért végül az általuk generált kimenet nem került felhasználásra. A mérföldkő végére a CoHITS algoritmus implementálásának befejezése és tesztelése maradt. Ezen mérföldkő keretén belül ismerkedtem meg a Google grafikakészítő API-val, majd azt felhasználtam a CoHITS algoritmus, korreláció és átlagtól való átlagos eltérés adatsorok reprezentációjához.

2.1.4. Negyedik mérföldkő

A negyedik mérföldkő során egy másik projektből[1] származó algoritmusok és azok már kész implementálásának feldolgozása és megismerése volt a fő feladat. Át kellett tekinteni a kapott kódbázist abból a célból, hogy ezeket is mint majd további opciókat integráljam a rendszerbe. Az említett algoritmusok: Hamming, Koszinusz, Precedencia, Összefüggősségi. De még mielőtt ebbe bele kezdtem, kísérleteztem egy kicsit egy új grafikonrajzoló API (Charts.js) használata mely könnyebben, stílusosabban és személyre szabhatóbban reprezentálja az adatokat. Miután megkaptam a fent említett már implementált algoritmusokat, elkezdtem azokat integrálni az én projektem kódbázisába. Sajnos ez nem ment teljesen gördülékenyen, szükség volt a kapott kódbázisból előkerülő implementációs és működési problémákkal kapcsolatos kérdések átgondolására valamint konzultálásra ezekkel kapcsolatban, majd a hibák javítása. A sikeres hibaelhárítás és javítások után jött az algoritmusok és demo oldal működésének tesztelése.

2.1.5. Ötödik mérföldkő

Az ötödik mérföldkő bebizonyította, hogy érdemes volt jól megtervezni és szétválasztani az egyes stilisztikai elemeket, mivel a megrendelő ismét a menü rendszer valamint a fejléc kép ismételt átrendezését és szerkesztését kérte. Továbbá új teszt adatok használatát javasolta a *demo* menüpontban, amelyek már a friss adatokat tükrözik. Implementálásra került egy visszaigazolást kérő felhasználói felület mely fő célja a beviteli mátrix véletlen felülírásának meggátolása volt. Később a bevitel metódus megváltoztatása miatt ennek a funkcionalitása elévült. A korrelációt számoló kód működéséből fakadó hiba ki lett javítva, így már nem állt meg a számolás, ha a kapott hiányos adatsoron nem lehetett egyszerűen korrelációt számolni. Valamint megtörtént a tesztelések során előkerült további hibák kijavítása is.

2.1.6. Hatodik mérföldkő

A hatodik mérföldkő vissza hozta a régi grafikonrajzoló API-t (Google charts API), valamint megtörtént a grafikonokat megjelenítő elemek stilizálásnak egységesítése. Fő ok a visszaváltásra az volt, hogy az értékeket mint címkéket megjeleníthetővé tegyük az oszlopok mellett. A borszakértők személyiségi jogainak védele érdekében a nevük anonimizálásra került. A web alkalmazás új funkciókkal bővült: Regisztráció, belépés. Ezt követte a *demo* oldal teljes átdolgozása így már az hozzá kapcsolódik a felhasználói rendszerhez. Ebből fakadóan a *demo* oldal már csak regisztráció illetve bejelentkezés után használható. így a felhasználók a neves borszakértők által kóstolt borokat saját maguk is tudják értékelni.

2.1.7. Hetedik mérföldkő

A hetedik és egyben utolsó mérföldkő nagy részét a mobil android alkalmazás elkészítése és tesztelése tette ki, de itt valósult meg az őt kiszolgáló web szolgáltatások implementálása valamint egy alap adminisztrációs felület létrehozása. Legvégül pedig elvégeztem némely kód csinosítást és re-faktorizációt a célból, hogy a kódom könnyebben olvasható és általában véve jobb tetszést keltsen az emberben.

2.2. Adatbázis

Annak ellenére hogy a web oldal legelső változata perzisztens adat tárolás nélkül üzemelt és az implementált algoritmus tesztelése részben e nélkül zajlottak, nem jelenti azt hogy a végső változathoz ez kimaradt volna, vagy esetleg egy nem teljesen kidolgozott adat tároló hierarchiával rendelkezne. Pontosan az ellenkezője az igaz. Az adatbázis táblák létrehozása előtt jelentős időt fordítottam azok megtervezésére, szem előtt tartva azt hogy a tervezett alkalmazásoknak milyen igényei lesznek. Az adatbázis relációs sémáját a 2.2 ábra ábrázolja. Fontos még megemlíteni, hogy az adatbázis összes táblája *utf8_unicode_ci* alapértelmezett karakter kódolást alkalmaz, mely biztosítja, hogy többek közt a Magyar nyelvi sajátos karakteri is megfelelően tárolódjanak.



2.2. ábra. Adatbázis model

Az adatbázist kiszolgáló szervernek az Oracle által fejlesztett MySQL-t választottam. Ezt ismerem a legjobban, valamint az sem elhanyagolható, hogy nyílt forráskódú talán ezért is rengeteg web szolgáltató által nagyon kedvelt gyakran alkalmazott megoldás.

2.2.1. scores tábla

Ahogy az értékeléseket tartalmazó *scores* táblát létrehozó szkript 2.1 kódrészlet is tükrözi, látható, hogy a táblában a felhasználók azonosítóját *user_id* tartalmazó oszlop és a borok azonosítóját *wine_id* tartalmazó oszlop is mint *FOREIGN_KEY* szerepel amik természetesen a borokat *wines* és a felhasználókat *users* tartalmazó táblák megfelelő oszlopaire hivatkoznak, ezek segítik az adatbázis konzisztenciájának megőrzését, mivel amikor egy felhasználó vagy bor törlésre kerül akkor az *ON DELETE CASCADE* megkötés hatására az értékeléseket tartalmazó *scores* táblából azok a sorok is törlődni fognak amik így már nem létező azonosítókra hivatkoznának. Itt talán még érdemes kitérni arra is hogy takarékosági okokból az értékelések maximum nyolc számjegyből állhatnak amiben a decimális pontosságot három tizedes jegyig engedélyezzük.

Kódrészlet 2.1. Az értékeléseket tartalmazó (*scores*) táblát létrehozó parancs

```
CREATE TABLE IF NOT EXISTS 'scores' (  
  'user_id' int(11) DEFAULT NULL COMMENT 'user' 's_id',  
  'wine_id' int(11) DEFAULT NULL COMMENT 'wines' 's_id',  
  'score' decimal(8,3) DEFAULT NULL COMMENT 'score_given_by_  
    ↳ the_user_id_user_for_the_wine_id_wine',  
  'timestamp' timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON  
    ↳ UPDATE CURRENT_TIMESTAMP,  
  FOREIGN KEY ('user_id') REFERENCES users('user_id') ON  
    ↳ DELETE CASCADE,  
  FOREIGN KEY ('wine_id') REFERENCES wines('wine_id') ON  
    ↳ DELETE CASCADE  
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=  
    ↳ utf8_unicode_ci COMMENT='score_data';
```

Valamint az is jól látható hogy egy időbélyeg oszlopban *timestamp* nyomon követjük majd az értékelések módosításának időpontját, ez elengedhetetlen a web és android alkalmazás között való effektív szinkronizációhoz. Végül talán még érdemes arra kitérni hogy melyik mező milyen értéket vehet föl és mik azok alapértelmezett értékei: a *user_id* és a *wine_id* helyzetét már tisztáztuk, nem maradt más mint az értékeléseket tartalmazó *score* és a módosítások időbélyegét nyomon követő *timestamp*. A *score* mező értéke akár *NULL* is lehet így lehetővé téve a felhasználók számára hogy ha esetleg meg-

gondolták magukat akkor törölnék a már meglévő értékelésüket. Ez lényegében csak egy praktikus megoldás mert így amikor az alkalmazás egyszerre több értékelés eredményét próbálja módosítani akkor azt meg tudja tenni egy *UPDATE* SQL paranccsal, nem kell ennek az esetnek a külön kezelésére egy *DELETE* utasítást írni. Valóban így több hely foglalódik le az adatbázisban, de ne felejtjük el, hogy olyan borok értékeléséről beszélünk amiket az adott felhasználó egyszer már értékelt. Én valószínűbbnek tartom azt hogy valaki a kitörölt bort később újra értékeli mint az hogy soha többet nem tér vissza hozzá. A *timestamp* mező alapértelmezett értéke a *CURRENT_TIMESTAMP* ami másodperc pontossággal tartalmazza az időt alábbi formátumban: $Y - m - d h : i : s$ ahol Y az év m a hónap d a nap mindegyik numerikus, majd h az óra i a perc és s a másodperc két karakteres formátumban. *NULL* helyett pedig $0000 - 00 - 0000 : 00 : 00$ fog szerepelni a mezőben.

2.2.2. wines tábla

A borokat tartalmazó *wines* tábla egyáltalán nem bonyolult. A tábla indexelését a *wine_id* segítségével történik amely minden egyes új bor hozzáadásánál automatikusan növekszik eggyel az *AUTO_INCREMENT* -nek köszönhetően. Az évjáratot *wine_year* és az árat *wine_price* tartalmazó mezőkön kívül az összes többi megadása kötelező. Az évjárat egy maximum négy jegyű egész szám lehet, míg az ára akár hatvannégy hússzú is lehet. Jelenleg a borok ára Magyar Forintban értendő. Ezek a megkötések főképp a projekt fejlesztése alatt megismert borkóstolási versenyek és adatbázisok szerkezeti felépítése miatt lettek így meghatározva. A többi mező kötelező szöveggel való feltöltését az alábbi táblázat 2.1 határozza meg:

Mező	Tartalom
wine_name	A bor neve
wine_winery	A pincészet neve
wine_location	A bor származási helye
wine_composition	A bor összetétele

2.1. táblázat. A borokat leíró tábla kötelező mezőinek felépítése

2.2.3. users tábla

A felhasználókra vonatkozó adatokat nyomon követő tábla neve az adatbázisban *users*. A tábla elsőre talán túl bonyolultnak tűnhet, de minden mezőnek fontos feladata van, melyek lehetővé teszik egy kedvelt felhasználó kezelő modul használatát [6]. A borokat tároló *wines* táblához hasonlóan itt is van egy azonosító még pedig a *user_id* amely a

tábla indexelését látja el és automatikusan növekszik eggyel az *AUTO_INCREMENT*-nek köszönhetően amikor egy új felhasználó regisztrál a rendszerbe. A *user_id* nem az egyetlen egyedi *UNIQUE* azonosító a táblában, a felhasználó nevét tároló *user_name*, valamint a regisztrált felhasználó által megadott e-mail címet tároló *user_email* mező is egyedi ezzel azt meggátolva, hogy valaki már egy létező e-mail címmel vagy felhasználó névvel újból regisztráljon. Ezek voltak a regisztráció során kötelezően megadandó mezők egy része, persze később még szó lesz a jelszó kezelésről is. A regisztráció után már aktivált felhasználó státuszt a *user_active* követi nyom, úgy hogy azokhoz akik még nem látogatták meg a regisztrációt visszaigazoló e-mailben kapott linket, hozzájuk nullát rendel, akik pedig ezt már megették egyet kapnak, ez arra szolgál hogy csak olyan e-mail címeket zároljon a rendszer amelyek már regisztráció után aktivált felhasználó fiókhoz tartoznak. A felhasználót azonosító süti kulcs a *user_rememberme_token* mezőben tárolódik, mely két hétig érvényes, ezzel lehetővé téve a web alkalmazás számára, hogy felismerje az ismételt látogatókat még akkor is ha már más *HttpSession* be tartoznak. A hibás bejelentkezések számát a *user_failed_logins* mező tárolja, többek között ennek pontos felhasználását később fogom részletezni a felhasználó kezelő rendszer részletes ismertetésénél. A regisztráció során használt ip cím a *user_registration_ip*-ben tárolódik el. Adminisztrátori jogosultság megadását teszi lehetővé a *user_permission* mező, melynek alapértelmezett értéke 0, 1 pedig az adminisztrátori jogosultságot szimbolizálja. A még nem tárgyalt mezőket két csoportra bonthatjuk, az egyik mely valamilyen hash stringet tartalmaznak mint a *user_password_hash*, *user_activation_hash*, *user_password_reset_hash*. A másik csoport pedig olyan mezők amelyek valamilyen tevékenységet nyomon követő idő bélyeget tárolnak mint a: *user_password_reset_timestamp*, *user_last_failed_login*, *user_registration_datetime*. Nézzük először az utóbbit, az idő bélyeg típust tároló mezőket. A *user_password_reset_timestamp* arra szolgál hogy nyomon tudjuk követni, hogy a felhasználó vagy esetleges támadó mikor igényelt jelszó újra beállító e-mailt. A *user_last_failed_login* mező a legutolsó hibás bejelentkezés időpontját tárolja. A *user_registration_datetime*-ben tárolt időpontra pedig majd a regisztrációt visszaigazoló és aktiváló e-mail linkjének kezelésekor lesz szükség. Végül röviden azokról a mező típusokról amelyek valamilyen hash stringet tárolnak, mivel arra, hogy mi a hash és miért de főképp milyen módon van használva, majd többek között később térek ki a felhasználó kezelő rendszer részletes ismertetésénél. A *user_password_hash* tárolja a felhasználó jelszavát hashelt formátumban. A regisztráció után küldött aktivációs e-mailhez használt aktivációs link validitásának ellenőrzésére szolgáló hasht a *user_activation_hash* tárolja. *user_password_reset_hash* pedig az új jelszót igénylő e-mail link hitelesítésére használt hasht tárolja.

2.3. CSS

CSS avagy Cascading Style Sheets egy stilisztikai leíró nyelv melynek célja hogy leírja miképp formázódjanak és jelenjenek meg egy dokumentum elemei amelyeket egy markup nyelven írtak. Markup nyelvnek nevezzük az olyan nyelveket melyek szintaktikailag megkülönböztethetők a dokumentum szövegétől, jelenes esetben ez a HTML, mely előre definiált prezentációs sémákkal rendelkezik, ami azt jelenti hogy egy adott specifikáció adja meg, hogy az adatokat milyen módon jelenítse meg a rendszer.

A CSS-t először Hakon Wium Lie terjesztette elő 1994 Októbar 10. -én [8] a weblapok stílusát leíró nyelvként a W3C (World Wide Web Consortium, a világháló szabványaiért felelős nemzetközi szervezet) publikus levelezőlistáján. A CSS 1 specifikációja elkészültekor még nem volt olyan böngésző amely teljesen támogatta volna azt. Két évvel később a CSS 2 többek között bemutatta elemek abszolút, relatív és fixált pozicionálását, valamint bevezette a z tengely mentén való indexelés lehetőségét. A CSS 3 tól nem beszélhetünk W3C által elfogadott szabványról, ez a verzió főképp abban tér el elődjétől, hogy egy moduláris felépítést követ. Minden modul kiterjeszti valamilyen módon, vagy új képességekkel látja el a korábbi CSS 2 -es funkciókat, ezzel megőrizve a visszafelé kompatibilitást.

Tagadhatatlan, hogy a CSS a webfejlesztés egyik fő sarokköve. Létrehozásának fő oka az volt, hogy a HTML oldalak tartalmi illetve prezentációs része könnyen szétválasztható legyen. Ez a szeparáció azért is fontos, mert így nagyobb flexibilitást nyerhetünk a prezentáció területén, valamint lehetővé válik a HTML oldalak formázottságának megosztása egy egységes CSS fájlban köszönhetően. Ezzel csökken a tartalmi oldalak komplexitása, valamint kevesebb lesz bennük az ismétlődés. Előnyök között mindenképpen említésre méltó, hogy így könnyebben lehetséges ugyan azt a markup nyelv által leírt dokumentumot több különböző stílusban megjeleníteni attól függően, hogy kinek, illetve milyen célra szánt az adott tartalom. Pontosabban ez alatt azt értjük hogy a megjelenítés különböző lehet akár egy képernyőn, nyomtatásban, hangosan (képernyő felolvasáskor) vagy Braille billentyűs eszközökön. Továbbá az is lehetséges, hogy a megjelenítendő tartalmat illeszkedjen az adott eszköz képernyő méretéhez.

Számomra mint már korábban említettem a CSS fő előnye az volt, hogy a web alkalmazás stilisztikai felépítést egy fájlból könnyedén tudtam módosítani. De sajnos nem csak előnye vannak a CSS használatának. Az ember könnyen bele tud futni néhány igen bosszantó limitációba, mint például elemek vertikális rendezése. Míg a vízszintes elrendezés általánosságban könnyen kezelhető, a vertikális sokszor nem túl intuitív módon valósítható csak meg, ha egyáltalán megvalósítható. Egyszerű feladatok mint például egy elem vertikálisan való középre zárása, vagy a lábléc limitálása hogy ne legyen magasabb mint a viewport alja, vagy nem intuitív komplikált szabályrendszert igényel, vagy olyan

szabályokat igényel melyek nem túl bonyolultak viszont kevésbé támogatottak. Viszont az kétségtelen, hogy használatának előnyei túlnyomó részben vannak hátrányaihoz képest. Némelyikbe azért konkrétan én is bele futottam és jelentős kísérletezgetésbe telt mire sikerült olyan szabályrendszert alkotni a leíró nyelvben mely a számomra elfogadható megjelenítést tette lehetővé. A kulcs az volt hogy minden fő elemet egy konténerbe kellett tegyek amelyek egy *clear : both*; tulajdonsággal rendelkeznek majd ezeken belül már könnyebben tudtam pozicionálni a további elemeket. Ennek megértéséhez azt kell tudni, hogy a CSS -ben három fő pozicionálási mód van.

Normal flow ♦ Az *inline* elemek úgy vannak elrendezve mint a betűk egy szavakat tartalmazó szövegben, tehát egymás után a rendelkezésre álló területen amíg van szabad hely, ha elfogy a hely akkor egy új sorban folytatódnak. *Block* elemek vertikálisan rakhatók össze, mint a paragrafusok vagy mint a felsorolások egyes pontjai. A *normal flow* lehetővé teszi *inline* elemek vagy *blokkok* relatív pozicionálást.

Float ♦ Egy lebegő elem kikerül a *normal flow*-ból és balra vagy jobbra igazodik amennyire csak lehet a rendelkezésre álló szabad területen. A többi tartalom majd ennek az oldalán lebeg.

Absolute positioning ♦ Egy abszolút módon elhelyezett elem nem foglal helyet és nincs hatással a *normal flow* típusú elemekre. Egy meghatározott saját területet foglal el a konténerében a többi elemtől függetlenül.

Position ♦ Nem meglepő módon négy pozíció van: *top* - fel, *bottom* - le, *left* - balra, *right* - jobbra. Ugyan csak négy lehetséges értéke lehet a *positio* tulajdonságnak. Amennyiben egy elem *static* -től eltérő módon van elhelyezve akkor a további tulajdonságok *top*, *bottom*, *left*, *right* eltolás illetve pozíció definiálásra szolgálnak. A négy lehetséges érték:

1. **Static**: Az alapértelmezett érték amely a *normal flow* -ba teszi
2. **Relative**: Az elem a *normal flow* -ba kerül majd abból a pozícióból eltolódik. A további elemek úgy kerülnek elrendezésre mint ha az eredeti elem ott maradt volna.
3. **Absolute**: Abszolút pozicionálást definiál. Az elem a legközelebbi nem *static* elemű ősehez képest lesz pozicionálva.
4. **Fixed**: Az elem abszolút pozicionálásra kerül egy fixált pozícióban a képernyőn még akkor is ha az egész dokumentum el van görgetve. az elemet

Float és clear ♦ A *float* tulajdonságnak három különböző értéke lehet. Abszolút vagy fixen pozicionált elemek nem lehetnek *float* típusúak. Más elemek normálisan lebegnek a többi *float* típusú elem körül, hacsak azt az *clear* tulajdonságuk meg nem gátolja.

1. **left**: Az elem annak a vonalnak a bal oldalán jelenik meg ahol eredetileg jelent volna meg, a többi elem a jobb oldalán lesz.
2. **right**: Az elem annak a vonalnak a jobb oldalán jelenik meg ahol eredetileg jelent volna meg, a többi elem a bal oldalán lesz.
3. **clear**: Arra kényszerít egy elemet hogy az *clear* által meghatározott elem bal oldalán legyen *clear : left* vagy jobb oldalán *clear : right* vagy esetleg mindkét oldalán *clear : both*.

Kellő kísérletezés és gyakorlás után azért elérhető egy olyan struktúra amely ezeket megfelelően egymásba ágyazva a kívánt kinézetet nyújtja.

Érdemes még azért a CSS szintaktikájával is tisztában lenni. Röviden egy CSS szabály egy *selector*-ból és egy deklarációs blokkból épül fel.

Kódrészlet 2.2. CSS szintaxis

```
h2 { color : #bb1f42 ; font-size : 1.2em ; }
```

Konkrétan a 2.2 kódrészletben látott CSS kódnál a *h2* a *selector* őt követi a deklarációs blokk amely két deklarációt tartalmaz. Egy deklarációnak két része van, egy a tulajdonságot megnevező része, ez van a : előtt, és egy azt követő rész amely az értéket tartalmazza. Lehetséges még több *selector* összecsoportosítsa, melynek köszönhetően jócskán csökkenthető az ismétlődő kód. Ugyan ezt segíti elő az is, hogy a CSS alapvetően támogat öröklődést, így egy felső konténeren definiált tulajdonság mindig lefelé a gyermekei felé propagál. Természetes a kevesebb kódnak más jó mellékhatása is van, mint a gyorsabban töltődő weboldal és kisebb sávszélesség használat. A *selector* több típusú is lehet és ezeket kombinálni is lehet. A 2.2 kódrészletben látottat *elementselector* -nak hívják, mely egy adott típusú elemet választ ki. A *classselector* az olyan elemekre fog vonatkozni amik az adott *class* attribútummal rendelkeznek. Ilyenkor egy pontot teszünk a *selector* elé. Némely elem rendelkezhet különböző típusokkal is, pl.: egy *checkbox* típusú *input* elemre a *selector* az alábbi módon nézne ki: *input[type = checkbox]*.

2.4. PHP

A PHP (*PHP : Hypertext Preprocessor*) egy széles körben elterjedt nyílt forráskódú általános célokra kifejlesztett szkript nyelv amely különösképpen alkalmas webes fejlesztésre, valamint HTML -be ágyazható. Ezt az alábbi példa 2.3 kódrészlet szemlélteti.

Kódrészlet 2.3. PHP Hello World!

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD_HTML_4.01_Transitional//  
    ↪ EN"  
"http://www.w3.org/TR/html4/loose.dtd">  
<html>  
  <head>  
    <title>Példa</title>  
  </head>  
  <body>  
    <?php  
      echo "Hello_Világ!";  
    ?>  
  </body>  
</html>
```

A PHP kód egy különleges nyitó és záró `<?php ?>` struktúrában helyezkedik el, ezzel lehetővé téve, hogy ki és be lépjünk a PHP-ba. A PHP-t azt különbözteti meg a kliens oldali szkriptektől mint például a JavaScript, hogy a kód a szerver oldalon hajtódik végre, itt generálódik a HTML tartalom melyet majd a webszerver küld a kliensnek. Az is megoldható, hogy egy webszervert úgy konfiguráljunk, hogy az a PHP-t tartalmazó HTML fájlokat is dolgozza fel. Ekkor a felhasználó nem fogja megtudni, hogy mi neki most éppen egy PHP által generált tartalmat mutatunk.

Annak ellenére, hogy a PHP egy szkript nyelv robusztus alkalmazások létrehozására is használható, erre kiváló példa többek között a Facebook vagy akár a Wikipedia. Személy szerint ez előtt nem sokat foglalkoztam a PHP -val csak kisebb weboldalak létrehozására használtam, nem igazán követtem nyomon a nyelv fejlődését mely eredetileg a PHP/FI névvel lett létrehozva 1994 -ben Rasmus Lerdorf által. Ez az első változat egy egyszerű Common Gateway Interface (CGI) bináris állomány volt a C programozási nyelven írva melynek fő feladata az ő személyes honlapjának egy eszköze volt. Ez már nagyon régen volt és a nyelv rengeteg változtatáson és javításon esett át. Most már a PHP 5. verziójánál tartunk. Én is ezt a verziót használtam, mely 2004 -ben lett kiadva. Fő vezérlő magja a *Zend Engine* 2.0 amely egy tucatnyi új szolgáltatás között egy új objektum modellt is tartalmaz. Jelenleg több tucatnyi fejlesztő dolgozik a PHP továbbfejlesztésével mely továbbá pár tucat másik fejlesztőnek ad munkát akinek a feladata a PHP -hez kapcsolódó támogató projektek fejlesztése és dokumentálása. Ez csak egy korábbi éveken alapuló becslés, de nyugodtan mondhatjuk, hogy a PHP több tíz vagy akár száz millió domain-re van telepítve világszerte [7].

A PHP nyelv általános felépítésére és szintaxisára nem térek ki, egyrészt azért mert

már egy elég régi nyelv melynek alapjai régóta ismertek, valamint azért is mert elég közel a *C* nyelvhez, mely még ismertebb. Viszont mindenképpen ejtenék pár szót a nyelv sajátosságáról főképp arra fókuszálva, hogy hogyan történik az állapotkövetés illetve adatbázis elérés.

2.4.1. PHP Data Objects (PDO)

A PHP Data Objects (PDO) a PHP adatbázis elérést nyújtó interfésze. Minden adatbázis meghajtó amely implementál egy PDO interfészt elérhetővé tehet az adott adatbázisra specifikus funkciókat. Természetesen önmagában a PDO alkalmatlan adatbázis kezelésre, mindenképpen egy konkrét adatbázis meghajtót kell használni. Mivel a web alkalmazást egy *MySQL* adatbázis szerver szolgálja ki ezért én a hozzá tartozó *PDO_MYSQL*-t használtam. A PDO alapértelmezetten együtt jár a PHP 5.1-el és elérhető PECL kiterjesztésként is PHP 5.0 alatt, de mivel a PDO -nak szüksége van PHP 5 magjában bemutatott objektum orientáltságra, ezért a PDO nem fut korábbi PHP verziókon.

Kódrészlet 2.4. PHP PDO Connector használata

```
private function databaseConnection() {
    if ($this->db_connection != null) {
        return true;
    } else {
        try {
            $this->db_connection = new PDO('mysql:host=' . DB_HOST
                ↪ . ';dbname=' . DB_NAME . ';charset=utf8',
                ↪ DB_USER, DB_PASS);
            return true;
        } catch (PDOException $e) {
            $this->errors[] = MESSAGE_DATABASE_ERROR . $e->
                ↪ getMessage();
        }
    }
    return false;
}
```

A 2.4 kódrészlet a bejelentkeztető modul egyik metódusán keresztül mutatja be, hogy hogyan lehet a PHP PDO -vel adatbázis kapcsolatot létrehozni. Először is ellenőrzésre kerül, hogy létezik-e már felépített adatbázis kapcsolat, nyilván ha már igen akkor nincs mit tenni, de ha még nem, akkor létrehozunk egy új PDO objektumot, úgy hogy a konstruktorának átadjuk az adatbázis címét, nevét, azt a felhasználót amelyikkel csatlakozni

szeretnénk és természetesen a hozzá tartozó jelszót. Ezek a kódrészletben sorra a *DB_HOST*, *DB_NAME*, *DB_USER*, *DB_PASS* melyek egy külön konfigurációs fájlban kerültek definiálásra. Itt fontos még megemlíteni, hogy az adatbázis nevének megadásakor egyben megadjuk a karakter kódolást is, mivel ennek elhanyagolása biztonsági problémákat okozhat [9]. Természetesen a kódunknak készen kell állni esetleges nem várt események bekövetkezésére, ezért mind ez egy *try – catch* blokkban zajlik le, a remélhetőleg nem fellépő hibaüzeneteket pedig egy az azok tárolására létrehozott tömbben tároljuk, amiből majd később könnyen ki olvashatók, ha esetleg értesíteni akarunk valakit róluk.

Az adatbázis kapcsolatot nagyon egyszerű bezárni ahogy a 2.5 kódrészlet is mutatja. Ez nagyon fontos is, hogy mindig bezárjuk miután befejeztük a használatát, mivel sok szolgáltató limitálja, hogy hány aktív kapcsolatot kezel az adatbázis szerverük.

Kódrészlet 2.5. PHP PDO kapcsolat lezárása

```
db_connection = null;
```

Miután sikeresen létre hoztunk egy aktív adatbázis kapcsolatot SQL utasításokat a 2.7 kódrészletben látott módon indíthatunk. A példában éppen lekérjük egy felhasználó összes információját, majd az így kapott információkat egy objektumban adja vissza a *fetchObject()* metódus. Tehát például a *user_id* a 2.6 kódrészletben látottak szerint lesz elérhető. Használatos még többek között a *fetchAll()* ami az SQL lekérdezés által visszaadott összes sort egy egy tömbben adja vissza. Hasznos lehet még ismerni a *rowCount()* metódust, ami megadja az SQL parancs által érintett sorok számát.

Kódrészlet 2.6. PHP PDO eredmény kezelés

```
$user_id = $result_row->user_id;
```

Vegyük észre, hogy az SQL lekérdezésbe nem konkaténáljuk bele a változókat, ha ilyesmit tennénk akkor az alkalmazásunkat nagyon könnyen feltörhetnék, hiszen éppen ilyen programozói hibákat kihasználva működnek az *sqlinjection* jellegű támadások. Tehát ha változókat akarunk használni az SQL lekérdezésünkben, akkor a *prepare()* metódusban a behelyettesítendő helyre kettősponttal megelőzött változónevet kell írni, melyet majd a *bindValue()* metódussal fogunk ténylegesen a mi változónkhoz kötni. Ez első paraméterként egy stringet vár, ami az SQL lekérdezésben szereplő változó helyét határozza meg. A második paramétere a mi PHP változónk aminek az értékét az SQL parancsban szeretnénk használni. A harmadik és egyben utolsó na meg opcionális paraméter pedig az első paraméterre vonatkozóan definiálja annak adat típusát. Az *execute()* -hez nem kell sokat hozzáfűzni, egyszerűen végre hajtja a fentiek szerint előkészített SQL parancsot.

Kódrészlet 2.7. PHP PDO SQL lekérdezés

```
$query_user = $this->db_connection->prepare( 'SELECT_*_FROM_  
    ↪ users_WHERE_user_name_=_:user_name' );  
$query_user->bindValue( ':user_name', $user_name, PDO::  
    ↪ PARAM_STR );  
$query_user->execute();  
$result_row = $query_user->fetchObject();
```

2.4.2. Állapotkezelés és követés

Állapot kezelésre illetve nyomkövetésre minden kicsit is bonyolultabb felhasználókat kezelő alkalmazásnak szüksége. Mivel a HTTP egy állapot nélküli protokoll ezért amikor egy web szerver végrehajtotta egy kliens kérését a kapcsolat a kettő között megszakad. Tehát más szóval a szerver nem tudja felismerni hogy egy kérés sorozat ugyan attól a kientstől származik, viszont ez egy nagyon fontos dolog főképp akkor amikor különböző oldalak között navigálunk a web alkalmazásunkban. Ennek a problémának az orvosolására többféle megoldás is létezik. Konkrétan en a PHP nyelvben az állapotkezelésnek négy főbb fajtáját használtam.

Sütik avagy *cookies* alapvetően olyan stringek amik több mezőt is tartalmazhatnak. A válasz fejlécében a szerver egy vagy több sütit küldhet a böngészőnek. A süti *value* mezője rejtje a tényleges adatokat melyben bármilyen adat tárolható (azért van néhány limitáció), de teljesen alkalmas a felhasználó azonosítására és hasonló feladatokra. Egy sütit a *setcookie()* paranccsal küldhetünk a kliens böngészője felé. Ennek a függvénynek a szignatúráját a 2.8 kódrészlet mutatja.

Kódrészlet 2.8. PHP setcookie függvény

```
setcookie(name [, value [, expire [, path [, domain [,  
    ↪ secure ]]]]);
```

Tehát ez parancs hozza létre magát a süti stringet a megadott argumentumok alapján, majd létre hogy egy *Cookie header* -t is ezzel a stringgel és az értékével. Fontos megjegyezni, hogy mivel a sütik a válasz *header* részében kerülnek elküldésére ezért a *setcookie()* parancsot még az előtt meg kell hívni mielőtt a *body* bármely része is elküldésre kerülne. A *setcookie()* paraméterei a következők:

- **name**: A süti egyedi neve. Nem tartalmazhat üres karaktereket (*whitespace*) vagy ; -t. Természetesen használható több süti is, különböző névvel és attribútumokkal.
- **value**: A string amely a süti tényleges tartalma.

- **expire**: A süti elévülésének időpontja. Amennyiben ez nincs megadva, akkor a böngésző a sütit a memóriában tárolja nem pedig a lemezen így amikor a böngésző bezárásra kerül a süti is eltűnik. Az idő formátum 1970 Január 1. -től számított másodpercekben értendő GMT időzónába. Például a web alkalmazás két hétig emlékezhet azokra a felhasználókra akik bepipálták a "Tarts bejelentkezve (2 hétig)" dobozját. Az erre használatos elévülési érték a 2.9 kódrészletben láttak alapján generálhatjuk.
- **path**: A böngésző csak azoknak az oldalaknak fogja oda adni a sütit amelyek ezen elérési útvonal alatt állnak. Alapértelmezetten ez az a könyvtár ahol a jelenlegi oldal van. Tehát ha a `/valahol/valami/demo.php` állítja be a sütit és nem definiáljuk ezt az elérési út változót akkor a süti visszaküldésre kerül a szervernek minden olyan oldalon amelynek az URL részében megtalálható a `/valahol/valami/`.
- **domain**: A böngésző csak az itt megadott domainnel rendelkező URL -ek számára fogja vissza adni a sütit. Alapértelmezett értéke a szerver host neve.
- **secure**: Alapértelmezett értéke *false*. Viszont ha ez igazra van állítva akkor a süti csak *https* kapcsolaton keresztül lesz elküldve.

Kódrészlet 2.9. PHP cookie expire generálása 2 hét múlva

```
$expire = time() + 1209600;
```

Láttuk hogyan lehet sütit létre hozni, azoknak milyen a felépítése, már csak azt kell tudni, hogy ezeket hogyan kaphatjuk meg a kliens böngészőtől. PHP -ban ez nagyon egyszerűen történik a `_COOKIE` tömb változó használatával. A kulcs a süti neve, az érték pedig a süti *value* mezőjében tárolt tényleges érték. A web alkalmazás a sütit arra használja, hogy lehetővé tegye a felhasználók számára azt, hogy ne kelljen újra bejelentkezniük két hétig.

A felhasználók állapotának nyomkövetésére az egyik fő módszer a *session* követés. A *session* lényegébe véve egy változó ami a szerveren tárolódik és alapértelmezetten az életciklusa addig tart amíg be nem zárjuk a böngészőnket. A legtöbb *session* beállít a felhasználó számítógépén egy *user – key* változót ami valahogy így szokott kinézni: `365211cf14ert0dede5a562e2f3a1e42`. Ez után amikor egy új oldal kérdezzük le a *session*-t akkor a felhasználó számítógépén tárolt kulcs egyezésének ellenőrzése fog megtörténni, amennyiben azok megegyeznek, akkor az a *session* kerül felhasználásra, ha nem akkor egy új *session* kezdődik. A *session* tartalma a `_SESSION` PHP globális tömbben tárolódik el, miután elindítottunk egy *session* -t a `session_start()` paranccsal. Ez után ezt nyugodtan kezelhetjük úgy mint egy globális változót. A *session* befejezéséhez a `session_destroy()` parancsa használandó, a `session_unser()` pedig törli az összes *session*

változót. A web alkalmazás ezt a módszert használja a felhasználó állapotának nyomkövetésére. Más szóval a *session* -ben tárolódnak a felhasználó adatai.

Egy másik módszer a *hidden* avagy rejtett űrlap illetve mezők használata. A PHP ugyan úgy kezeli a rejtette űrlapokat mint a normálisakat, tehát azok értékei ugyan úgy elérhetőek a *_GET* illetve *_POST* tömbökből. Az algoritmusok számolásával foglalkozó androidos felülete ezt a módszert használja ki a felhasználó azonosítására.

A harmadik általam használt állapotkövetési technika az URL újraírási módszer volt. Ennek az az alapelve, hogy az URL tartalmaz dinamikusan módosítható extra információkat, mint például a felhasználó egyedi azonosítóját. Az android alkalmazás ezt a módszert használja a bor értékelések szinkronizációjára a mobil eszközön elérhető személyes adatbázis és a távoli szerveren lévő nagy adatbázis között.

2.5. JavaScript

A JavaScript mint ahogy a neve is sejteti egy szkript nyelv mely főképp böngészőn belüli használatra lett tervezve. JavaScriptet általában felhasználói felülettel való interakció során használunk, de más használata is lehetséges többek között szerver oldali programoz, játék fejlesztés vagy asztali alkalmazás fejlesztés.

Néhány szintaktikai, standard könyvtári és persze névbeli hasonlóságon kívül a Java-nak nem rokona, a kettő igen különböző szemantikával rendelkezik. A JavaScript szintaktikája igazából a *C* nyelvből származik, míg a szemantikájára és tervezésére főképp a *Self* és *Scheme* programozási nyelvek voltak hatással [12].

A JavaScriptet eredetileg Brendan Eich fejlesztette ki a Netscape -nél végzett munkája során. A fő motiváció a nyelv megalkotásánál az volt hogy Microsofttal szemben konkurenciát teremtsenek a web technológiák és platformok között. A Netscape -nek egy olyan könnyűsúlyú interpretált nyelvre volt szüksége ehhez amely kiegészítette a Java-t és nem csak a professzionális programozókat vonzaná. A programozási nyelv a végleges JavaScript nevet akkor nyerte el amikor azt integrálták a Netscape böngésző második verziójába. 1997 Júniusában végül az Ecma International (European Computer Manufacturers Association egy non-profit szabvány alkotó szervezet) ECMAScript néven elfogadta a Netscape által beadott JavaScript-et. Később bele került a *ISO/IEC – 16262* szabványba is. A jelenlegi is használt ECMAScript 5.1 kiadására 2011 Júniusában került sor [13].

Mára a JavaScript az egyik legnépszerűbb webes programozási nyelvvé vált annak ellenére, hogy eredetileg sok hivatásos programozó bírálta a nyelvet többek között azért is mert annak a célközönsége főképp web programozók és egyéb "amatőrök" voltak. Végül az *AJAX* technológia megjelenése az ő érdeklődésüket is felkeltette. Ennek ered-

ményeként született több JavaScript alapú keretrendszer és külső könyvtár mely tovább finomította a JavaScript nyelv által nyújtott programozó praktikákat.

2.5.1. AJAX

AJAX avagy **A**synchronous **J**avaScript **A**nd **X**ML -nak hívjuk azokat a kliens oldali web fejlesztési technológiákat amelyek segítségével aszinkron web alkalmazásokat hozhatunk létre. Ez alatta azt értjük, hogy az AJAX segítségével a web alkalmazások adatokat küldhetnek és fogadhatnak aszinkron módon (a háttérben) a szervertől a nélkül hogy azok befolyásolnák a már létrehozott oldal kinézetét vagy működését. Adatok fogadhatók az *XMLHttpRequest* objektumon használatával ami elnevezésének ellenére nem limitálja csak és kizárólag XML használatát, manapság sokszor inkább JSON (JavaScript Object Notion, JavaScript alapú nyílt szabvány ami ember által könnyen olvasható szöveges formátumban tartalmaz adatokat) használt. A technológiának napjaikban is van még néhány hátulütője, ilyen a dinamikusan frissített web oldalak könyvjelzőzése. Tehát nehézkes lehet az alkalmazás egy adott állapotába visszatérni, ennek a megoldására több módszer is létezik, melyek között a legelterjedtebb az *URL fragment identifier* (az URL "#" utáni része) használata az alkalmazás állapotának nyomkövetésére [14].

A 2.10 kódrészletben bemutatott példán keresztül látható, hogy a borkóstolás web alkalmazáson belül hogyan használtam az AJAX technológiát.

Kódrészlet 2.10. AJAX példa kódrészlet

```
var response ;
var xmlhttp=new XMLHttpRequest() ;
xmlhttp.onreadystatechange=function() {
    if (xmlhttp.readyState==4 && xmlhttp.status==200){
        response = xmlhttp.responseText ;
    } else {
        return ;
    }
}
xmlhttp.open("POST",url , false) ;
xmlhttp.send() ;
```

A *xmlhttp.readyState == 4* azt jelenti, hogy a kérés sikeresen befejeződött és arra válasz is érkezett, míg a *xmlhttp.status == 200* egyszerűen az "OK"-t jelenti (404 a nem található oldalt jelentené). Annak ellenére, hogy a *GET* metódussal szemben a *POST* lassabb, az jóval biztonságosabb olyan esetekben amikor felhasználók által megadott bemenetet továbbítunk a szerver felé, valamint a *POST* -nak nincs méret béli

korlátozása.

2.5.2. jQuery

A jQuery tagadhatatlanul a legnépszerűbb JavaScript könyvtár. Népszerűségének több oka is van, melyek közül nem elhanyagolható az hogy ingyenes és nyílt forráskódú. A jQuery létrehozásánál az egyik fő szempont az volt, hogy lehetővé tegye HTML dokumentumok elemei közti navigációt, mint a DOM (**D**ocument **O**bject **M**odel - többek közt HTML objektumok reprezentálására és interakciójára használt nyelv független konvenció) elemek kiválasztását, animációk létrehozását, események kezelését, na meg persze AJAX alkalmazások fejlesztését. Előnyei közé sorolható még az is, hogy erősen a HTML és JavaScript szeparálására buzdít az által, hogy nagyon könnyűvé tesz esemény kezelők DOM-okhoz való csatolását a helyett hogy HTML esemény attribútumokkal kellene hogy küszködjünk, így könnyen elválhat a JavaScript kód a HTML markuptól.

A könyvtár használatához a 2.11 kódrészletben látott módon először be kell azt töltenünk.

Kódrészlet 2.11. jQuery betöltése

```
<script type="text/javascript" src="https://code.jquery.com  
↔ /jquery-1.11.1.min.js"></script>
```

Természetesen más forrásból is betölthetjük, akár a saját szerverünkről is.

A jQuery-nek két felhasználási stílusa van:

1. A "\$" függvényen keresztül, amely a jQuery objektum egy factory metódusa. Ezeket a függvényeket sokszor parancsoknak hívják és egymás után láncolhatók mivel mindegyikül jQuery objektumokkal tér vissza.
2. A "\$." prefixált függvényen keresztül. Ezek segéd függvények, amelyek nem közvetlen a jQuery objektummal érintkeznek.

jQuery -ben több DOM elem betöltése illetve manipulálása általában azzal kezdődik, hogy meghívjuk a "\$" függvényt egy CSS selector stringgel. Ez majd egy jQuery objektumot fog referencia szerint visszaadni amely tartalmazza az összes erre illeszkedő HTML elemet az oldalon. `$(".div.valami")` például egy olyan jQuery objektumot ad vissza amely tartalmazza a *valami* CSS osztály összes *div* elemét. Ez után akár közvetlenül a visszakapott halmazon hívhatjuk a jQuery függvényeket, vagy azok valamely elemén. Egy tipikus használati mód látható a 2.12 kódrészletben, mely azt mutatja be hogyan indíthatunk parancsokat miután a böngésző befejezte az oldal betöltését.

Kódrészlet 2.12. jQuery tipikus használat

```
$(window).on('load', function () {  
    // jQuery kód  
});
```

2.5.3. Numeric Javascript

A *Numeric Javascript* egy JavaScript könyvtár mely segítségével numerikus analízis-béli számításokat végeztethetünk kliens oldalon a web böngészőben. Annak köszönhetően hogy csak a JavaScript nyelvet használja fel a könyvtár több típusú böngészőben is futtatható, természetesen feltéve, hogy azok támogatják a JavaScript-et. A *Numeric Javascript* használata sokat segít abban hogy a számolás igényes feladatokat ne a szerveret terheljék, kis kapacitással bíró szerverek esetén vagy esetleg olyankor amikor az alkalmazást egy limitált erőforrásokkal rendelkező felhasználó futtatja a szerven az ilyen megközelítés elengedhetetlen. A könyvtárat Sébastien Loisel készítette és MIT licenccel publikálta [15]. A könyvtár használatához egyszerűen csak be kell töltenünk azt a már megszokott módon (2.13 kódrészlet) és máris használhatjuk a könyvtár által nyújtott metódusokat.

Kódrészlet 2.13. Numeric Javascript betöltése

```
<script type="text/javascript" src="http://numericjs.com/  
    ↪ numeric/lib/numeric-1.2.6.min.js"></script>
```

A könyvtár csomagból főképp a sajátérték számítást végző *eig* metódust használtam fel a *CoHITS* algoritmus implementálása során.

2.6. Grafikonok

Az eredmények igényes megjelenítés megkövetelt valamilyen típusú grafikonkezelést. Mivel több nagyon könnyen felhasználható és jól működő *JavaScript* publikus könyvtár létezik már ennek a feladatnak az ellátására, úgy gondoltam, hogy egy-kettőt kipróbálok mielőtt véglegesen valamelyik használata mellett döntenék. Kis kutató munka után arra jutottam, hogy a választásom a *GoogleCharts* [10] és a *Charts.js* [11] két grafikon készítő API közül az egyikre fog esni. Míg számomra úgy tűnt, hogy a *Charts.js* használata talán könnyebb, stílusosabban és személyre szabhatóbban reprezentálja az adatokat apróbb probléma volt, hogy az értékek mint címkék megjelenítése nem volt alapból támogatva az oszlop diagram osztályban. Persze akár kiterjeszthettem volna ezt az osztályt és saját magam implementálom az oszlopok adatokkal való címkézését, de mielőtt ebbe bele vágtam volna azért megnéztem hogy a *GoogleCharts* rendelkezik-e hasonló szolgáltatással. Mint kiderült igen, így a végső választásom erre a grafikon kezelő *JavaScript*

API -ra esett annak ellenére, hogy talán stilisztikai szempontból alapértelmezetten nem ajánl fel olyan nagy mértékű testre szabhatóságot mint a *Charts.js*.

2.6.1. Google Charts

A Google Charts egy grafikonokat vizualizáló eszköz, amit leggyakrabban *JavaScript*-tel szoktak használni. Először be kell tölteni a Google Chart könyvtárat és a hozzá szükséges AJAX API-t mint az a 2.14 kódrészletben is látható.

Kódrészlet 2.14. Google Charts API betöltése

```
<script type="text/javascript" src="https://www.
    ↪ google.com/jsapi"></script>
<script type="text/javascript">
    google.load('visualization', '1.0', {'packages':[
    ↪ corechart'}]);
</script>
```

Majd a bemeneti adatokat úgy kell rendezni ahogy az a könyvtár osztályban definiálva van, ami konkrét esetünkben egy *JavaScript* tömböket tartalmazó tömböt jelent, melynek első eleme az adatok típusainak megnevezését tartalmazza. A tömb többi eleme pedig az így definiált sorban megadott értékeket kell hogy tartalmazzak minden egyes megjelenítendő grafikon elemre vonatkozóan. Ez nagyon intuitív ha a 2.2 táblázatban látottak szerint gondolunk a *GoogleCharts* által várt bemeneti formátumra, csak sor folytonosan leírva.

Kóstoló	CoHITS
Kóstoló 1	100
Kóstoló 2	99.853
Kóstoló 3	92.123
Kóstoló 4	70.482

2.2. táblázat. Példa a *GoogleCharts* által megjelenítendő adatok szerkezetére

Az így leírt adatsor tényleges előkészítését a *GoogleCharts* számára a 2.15 kódrészletben látott utasítás végzi el.

Kódrészlet 2.15. Google Charts tömbből google data table készítés

```
var data = google.visualization.arrayToDataTable(graphData)
    ↪ ;
```

Végül példányosítás után megjeleníthető az általunk választott grafikon a kívánt HTML `<div>` azonosítóban, de még mielőtt ezt megtennénk lehetőségünk van a megjelenítés

módjának testreszabására mint az a 2.16 kódrészletben is látszik, konkrétan például az oszlopok által reprezentált értékek kirajzolására azok mellé.

Kódrészlet 2.16. Google Charts megjelenítésének beállítása és a példányosított objektum beszúrása a megfelelő `<div>`-be

```
var view = new google.visualization.DataView( data );
view.setColumns([0, 1,
    { calc: "stringify",
      sourceColumn: 1,
      type: "string",
      role: "annotation" }
]);
var chart = new google.visualization.BarChart( document.
    ↪ getElementById( "graph" ) );
chart.draw( view );
```

A különböző grafikon típusok mint *JavaScript* osztályok vannak megvalósítva. Általában az alapértelmezett kinézet már önmagában is igényes megjelenítést biztosít, de lehetőség van ennek testreszabására is. Ez úgy történik meg, hogy a grafikonok eseményeket tesznek elérhetővé amelyekre csatlakozva akár komplexebb adat táblákat is beágyazhatunk a web oldalunkba. A grafikonok a HTML5/SVG technológiájával kerülnek renderelésre így biztosítva a több akár régebbi böngésző, vagy mobil eszköz támogatását. Minden grafikon típust adat táblájának feltöltéséért a már látott *DataTable* osztály felelős, ami megkönnyíti a grafikon típusok közötti váltogatást amíg a nekünk legjobban megfelelő kinézetűt keressük. A *DataTable* rendelkezik olyan metódusokkal amelyek az adatok rendezését, módosítását vagy akár szűrését is lehetővé teszik és közvetlenül a böngészőből fel lehet őket tölteni, de akár adatbázisból vagy bármilyen olyan forrásból ami támogatja a *ChartToolsDataSource* protokollt. Ennek a protokollnak van egy SQL szerű lekérdező nyelve mely többek között az *GoogleSpreadsheet* -ekben is implementálva van.

2.7. Statikus tartalom

Mivel a web alkalmazás bármely részén elérhető egy oldalsó felhasználó kezelő lebegő menü ezért ténylegesen nem lehet egyik oldalra sem azt mondani, hogy teljes mértékben csak statikus tartalommal rendelkezik, mégis az alábbi oldalakat ebbe a kategóriába sorolnám főképp azon okból kifolyólag, hogy tényleges felhasználó interakció csakis ezen a már említett lebegő oldalsó sávon történik, a többi tartalom még a felhasználó állapotától függően sem változik.

index.php - Főoldal ♦ Ez az oldal egy a Népszabadságban megjelent cikket tartalmaz.

borkostolasEredmenyek.php - Eredmények ♦ Az adatbázisban szereplő adatok egy kivonatát mely neves borkóstolók értékeléseit mutatja. Az itt látható eredmények csak ezen értékek figyelembevételével lettek generálva a CoHITS algoritmus által, így ez az oldal mindenképpen csak mint egy példa jellegű bemutatóként szolgál.

modszerek.php - Módszerek ♦ Eredetileg a használt algoritmusok bemutatására szolgált volna, de végül idő hiányában csak ezen szakdolgozatra mutató link lett belőle.

kapcsolat.php - Kapcsolat ♦ Információs oldal melyen keresztül a látogató felveheti a kapcsolatot az alkalmazás készítőjével.

2.8. Dinamikus tartalom

2.8.1. php-login

Hashelés ♦ A hashelés

Felhasználói állapot nyomkövetése ♦ Nyomkövetés

2.8.2. demo.php - Demó

3. fejezet

A mobil alkalmazás

Android alkalmazás fejlesztés

3.1. Android

Android VC ndroid register, passwordReset alk about threading UI/new thread mi futhat
hol és mi nem ávoli adatbázis elérés ocal SQLLite adatbázis okális változok kezelése
SQL/properties vagy mifene

M

a

t

t

l

l

4. fejezet

A weboldal és a mobil alkalmazás összefűzése

A weboldal és a mobil alkalmazás összefűzése

5. fejezet


Tesztelés

5.1. Regisztrációs és bejelentkeztető rendszer

5.2. Demo adatkezelésének ellenőrzése

5.3. Algoritmusok ellenőrzése kis adatokon

5.4. Algoritmusok ellenőrzése ismert eredményekkel

ellékelni az excel filet és arra hivatkozni, valamint az oldal kezdetleges verziójáról egy  m
kép

6. fejezet

Összefoglalás

7. fejezet

Egyebek

7.1. Verziókezelés

Verziókezelés röviden

Ez alatt több verzióval rendelkező adatok kezelését értjük. Leggyakrabban a szoftverfejlesztésben használnak verziókezelő rendszereket fejlesztés alatt álló dokumentumok, tervek, forráskódok és egyéb olyan adatok verzióinak kezelésére, amelyeken több ember dolgozik egyidejűleg vagy amelyen több fizikai helyről dolgoznak.

7.1.1. Git

A **Git** egy nyílt forráskódú, elosztott verziókezelő szoftver, amely a sebességre helyezi a hangsúlyt melyet eredetileg Linus Torvalds fejlesztette ki a Linux kernel fejlesztéséhez. Az elosztottság abban valósul meg, hogy a Git minden fejlesztő helyi munkaváltozatában rendelkezésre bocsátja a teljes addigi fejlesztési történetet, és a változtatások másolása mindig két repository között történik. Ezeket a változtatásokat mint külön ágakat importálják és összefésülhetők, hasonlóan a helyben létrehozott fejlesztési ágakhoz. Ez azért jó, mert így minden munkamásolat egy teljes értékű repository teljes verziótörténettel és teljes revíziókövetési lehetőséggel, amely nem függ a hálózat elérésétől vagy központi szervertől.

7.1.2. GitHub

A GitHubot eredetileg Tom Preston-Werner, Chris Wanstrath és PJ Hyett hozta létre a kódmegosztási procedura szimplifikálásának érdekében, mára a világ legnagyobb kód távoli kód repository szolgáltatójává nőtt [4].

7.1.3. A választás

Már a kezdetektől nyilvánvaló volt számomra, hogy valamilyen verziókezelő rendszer használata elengedhetetlen lesz, mivel gyakran nem csak egy specifikus munkaállomásról szoktam dolgozni. Ez nem csak azért jó, mert nagyobb flexibilitást nyújt térben és időben, hanem azért is mert valamilyen szinten szimulálja azt amikor valaki egy projekt munkában egy közös kódbázison dolgozik. Korábbi munkáim és tanulmányaim során már néhány ilyen rendszerrel is megismerkedtem, többek között SVN, Git, Mercurial. A szakdolgozathoz tartozó alkalmazások készítéséhet végül a Git -et választottam mint verziókezelő rendszer, melyhez távoli repository szolgáltatónak a GitHub társult. A választás fő támpontjai között talán a felhasználó barátságot és könnyű automatizálhatóságot említeném meg.

7.1.4. Repozitóriumok

A szakdolgozat, web és android alkalmazás forráskódja az alábbi linkeken érhető el:

- Szakdolgozat: https://github.com/TomVarga/borkostolas_szakdolgozat
- Web alkalmazás: https://github.com/TomVarga/borkostolas_web_interface
- Android alkalmazás: https://github.com/TomVarga/borkostolas_android_app

7.2. Környezetek

SublimeText - linterek

PhpStorm

Android Studio

Külön fájlban elkészített grafika beillesztését a

8. fejezet

Függelék

8.1. A program forráskódja

Nyilatkozat

Alulírott Varga Tamás programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Tanszékcsoport Számítógépes Optimalizálás Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Informatikai Tanszékcsoport könyvtárában, a helyben olvasható könyvek között helyezik el.

Szeged, 2015. május 1.

.....

aláírás

Irodalomjegyzék

- [1] Borkóstolás projekt. <http://www.inf.u-szeged.hu/~tnemeth/osa/>
- [2] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, **30**(1998), no 1, 107-117.
- [3] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, **46**(1999), no. 5, 604-632.
- [4] GitHub - <https://github.com/about>
- [5] London András és Csendes Tibor. HITS based network algorithm for evaluating the professional skills of wine tasters
- [6] Panique. <http://www.php-login.net>
- [7] History of PHP. <http://php.net/manual/en/history.php.php>
- [8] Lie, Hakon W (10 Oct 1994). Cascading HTML style sheets - a proposal
- [9] http://wiki.hashphp.org/PDO_Tutorial_for_MySQL_Developers#Connecting_to_MySQL
- [10] Google Charts homepage <https://google-developers.appspot.com/chart/>
- [11] Charts.js homepage <http://www.chartjs.org/>
- [12] ECMAScript Language Overview. 2007-10-23. p. 4. <http://www.ecmascript.org/es4/spec/overview.pdf>
- [13] ECMAScript Language Specification <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>
- [14] <http://blog.onthewings.net/2009/04/08/deep-linking-for-ajax/>
- [15] Sébastien Loisel. Numeric Javascript. <http://www.numericjs.com/>