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

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 mellet 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 létezek 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 célom 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, Összefüggőségi. 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 $w'\left(\overrightarrow{x_i y_j}\right)$ az az értékelés amit y_j bor kapott az x_i kóstolótól és legyen $w\left(\overrightarrow{x_i y_j}\right) = w'\left(\overrightarrow{x_i y_j}\right) / \sum_{j \in Y} w'\left(\overrightarrow{x_i y_j}\right)$ a normalizáltja. Továbbá legyen q_j^0 érték (az y_j borra) az értékelések átlaga. Majd definiáljuk a $w\left(\overleftarrow{x_i y_j}\right)$ az alábbi képen: tegyük fel hogy y_j bort l különböző kóstoló kóstolta és legyen

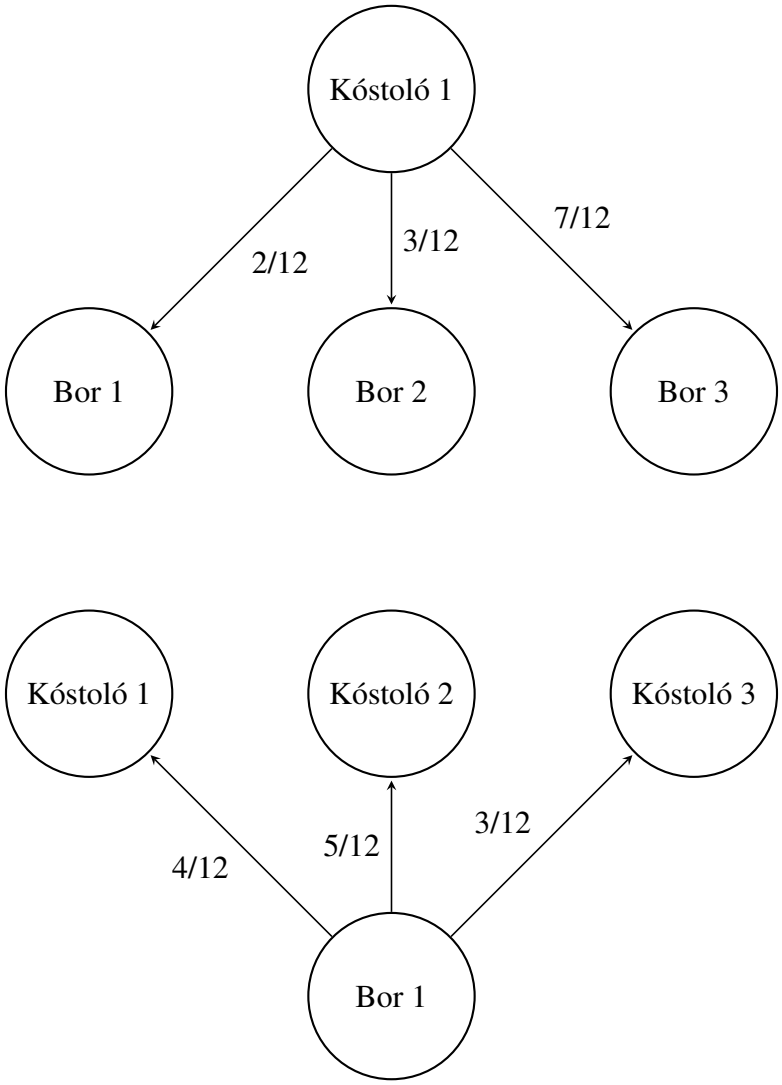
$$D = \sum_{i \in X} |q_j^0 - w'\left(\overrightarrow{x_i y_j}\right)|, \tag{1.1}$$

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

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

Ekkor $\sum_{i \in X} w\left(\overleftarrow{x_i y_j}\right) = 1$ tehát minden $w\left(\overleftarrow{x_i y_j}\right)$ 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 $p = (p_1, p_2, ..., p_m)$ 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.

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

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.

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

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.

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

1.5. Összefüggőségi

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 összefüggőségi mátrixa. Kezdetben minden eleme legyen 0, majd a π permutáció minden i elemére az $N_{pi}\left(\pi_i, \pi i + 1\right) = 1$. A π^1

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

Ekkor az összefüggősségi távolság (1.6) szerint számítandó.

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

[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 (**C**ascading **S**tyle **S**heets) 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

Statikus 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

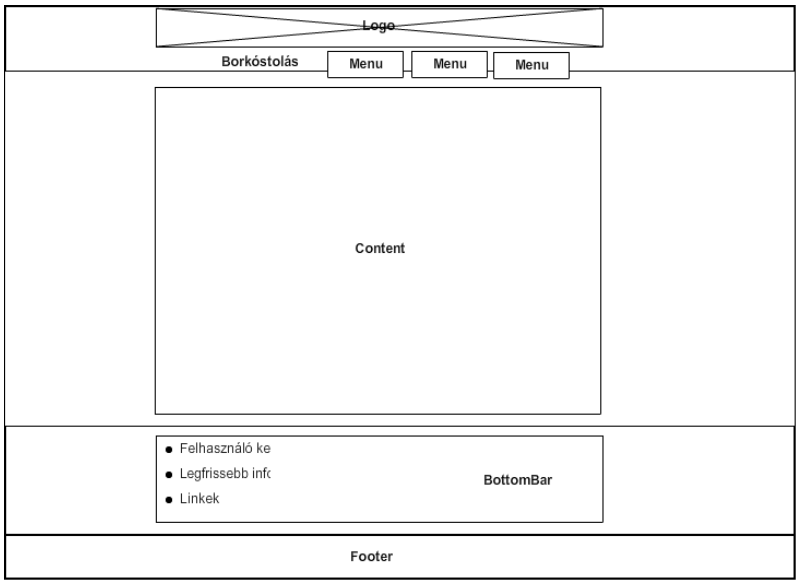
Az oldal megalkotásához egy agilis iterációs dizájn megközelítést alkalmaztam, melyen során egy rövid kezdeti tervezés után elkezdtem a tényleges munkát ami abból állt, hogy implementáltam az adott feladatot, majd az így elkészült munkát tesztelve prezentáltam a megrendelőnek (konzulensnek) A következő 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 megközelítésem 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 ($floatingsidebar$), 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 módosítása könnyítésének 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 grafikonkészítő 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, Összefüggőségi. 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 valamit 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

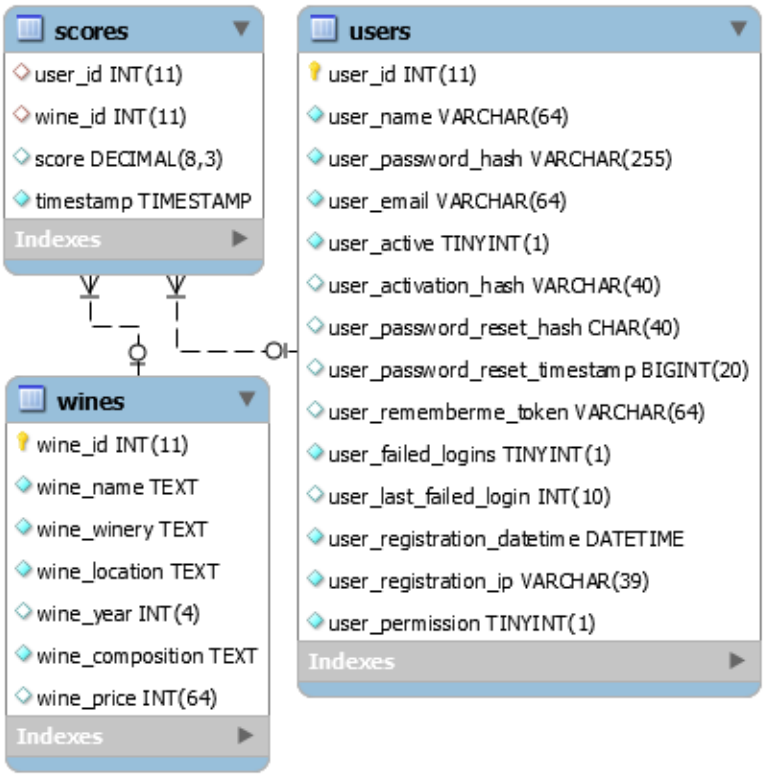
A hatodik mérföldkő vissza hozta a régi grafikonrajzoló API-t (Google charts API), valamint megtörtént a grafikonokat megjelenítő elemek stilizálásnak egységesítése. Fő ok a visszaváltásra az volt, hogy az értékeket mint címkéket megjeleníthetővé tegyük az oszlopok mellett. A borszakértők személyiségi jogainak védése 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 változatból ez kimaradt volna, vagy esetleg egy nem teljesen kidolgozott adat tároló hierarchiával rendelkezne. Pontosan az ellenkezője az igaz. Az adatbázis táblák létrahozása 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

2.2.1. scores tábla

Ahogy az értékeléseket tartalmazó *scores* táblát létrehozó szkript 2.1 kód részlet 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 oszlopaira 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 takarékossági okokból az értékelések maximum nyolc számjegyből állhatnak amiben a decimális pontosságot három tizedes jegyig engedélyezzük.

Kód részlet 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 meggondolták magukat akkor töröljék 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 felejtsük el, hogy olyan borok értékeléséről beszélünk amiket az adott felhasználó egyszer már értékelt. Én valószínűbbnek tartom azt hogy valaki a kitörölt bort később újra értékel 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 történek 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 hosszú 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. 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 megetették 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.

2.3. php

2.4. JavaScript

2.5. JQuery

2.6. Grafikonok

2.6.1. Google Charts

2.7. Statikus tartalom

2.7.1. index.php - Főoldal

2.7.2. borkostolasEredmenyek.php - Eredmények

2.7.3. modszerek.php - Módszerek

2.7.4. kapcsolat.php - Kapcsolat

2.8. Dinamikus tartalom

2.8.1. php-login

Hashelés ♦ A hashelés

Felhasználói állapot nyomkövetése ♦ Nyomkövetés

2.8.2. demo.php - Demó

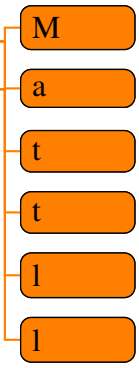
3. fejezet

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



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 összefésülhetőek, 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 procedúra 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.

7.2. 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. április 28. .....

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*, vol. 30, no 1, 107-117, 1998

[3] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, vol. 46, no. 5, 604-632, 1999

[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 *http://opensource.org/licenses/MIT MIT License*