XVI. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)
Kolozsvár, 2013. május 23–26.

RegionRank:
egy új típusú internetes keresési szolgáltatás

Szerzők:
Láng Máté-László
Babeş-Bolyai Tudományegyetem, Kolozsvár, Matematika és Informatika Kar, Informatika szak, 3. évfolyam
Pál Levente
Babeş-Bolyai Tudományegyetem, Kolozsvár, Matematika és Informatika Kar, Informatikai Matematika szak, 3. évfolyam
Sebesi Zoltán
Babeş-Bolyai Tudományegyetem, Kolozsvár, Matematika és Informatika Kar, Informatika szak, 3.évfolyam

Témavezetők:
dr. Simon Károly
egyetemi adjunktus, BBTE, MIK, Magyar Matematika és Informatika Intézet
képzési tanácsadó, Codespring Kft.
Török-Vistai Tamás
projektmenedzser, Codespring Kft.

Kivonat

A dolgozatban bemutatott RegionRank projekt célja egy új típusú internetes keresési szolgáltatás megvalósítása. A térképszolgáltatásokon alapuló szoftver különböző kiválasztott szempontok alapján egy hőtérképet generál, amelyen jól láthatóak az adott területnek azok a részei, amelyek legjobban megfelelnek a keresés céljának. A felhasználó saját igényei szerint választhatja ki a kereséshez alapot szolgáltató szempontokat és súlyozhatja is azok fontosságát.

Lényeges újítása a projektnek, hogy túllép a POI (Point of Interest) fogalmon és bevezeti a ROI (Region of Interest) fogalmát. A keresések alapjaként földrajzi pontok mellett tetszőleges alakzatok által reprezentált régiók is szolgálhatnak. A ROI-k „hagyományos” módon történő megjelenítése is lehetséges, ennek optimalizálására a szoftver klaszterezési algoritmusokat alkalmaz.

A RegionRank egy egységes API-t biztosít a különböző forrásokból származó, különböző formátumú adatok kezelésére, és könnyen bővíthető új adatforrások bevonásával.

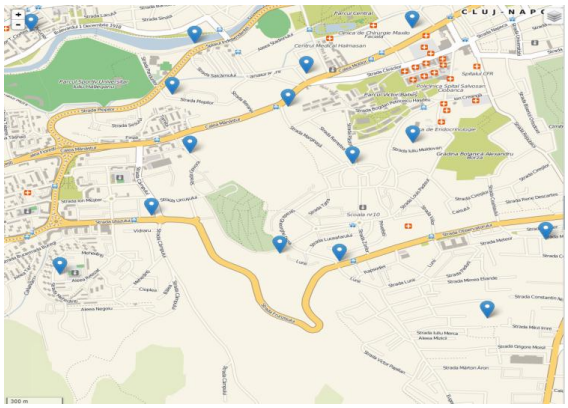
Tartalomjegyzék

Bevezető

A RegionRank projekt célja egy új látásmódnak megfelelő, új típusú internetes keresési szolgáltatás megvalósítása.

Az ötlet szemléltetéséhez tekintsünk egy konkrét példát: egy külföldi vendégdiák, aki nem ismeri Kolozsvárt, ki szeretne menni este megnézni egy filmet, vacsorázni, illetve sétálni, és mindezt szeretné nagyjából egy környéken megtenni, minimalizálva a tömegközlekedési eszközökön eltöltött időt. A jelenleg elérhető keresési szolgáltatásokat használva külön-külön rákereshetne a mozikra, éttermekre, parkokra stb, például valamilyen térképszolgáltatásokon alapuló keresők segítségével. Ez relatíve sok időt venne igénybe és ráadásul ezután neki kellene összevetni a keresési eredményeket és azok alapján megpróbálni meghozni egy megfelelő döntést.

A példa általánosítható: hasonló helyzetben van például egy fiatal házaspár, aki eladó vagy kiadó ingatlant keres bizonyos szempontok alapján (ár, alacsony bűnözési mutatók, zöldövezetek, forgalom, iskola stb).



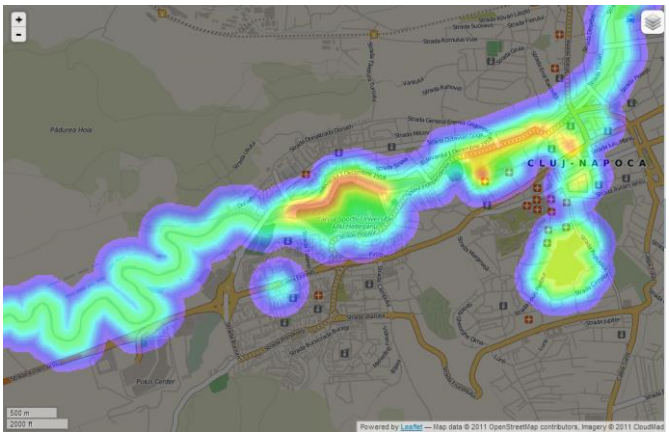
1. ***ábra:*** *példa kolozsvári vendéglátóipari egységek hagyományos módon történő keresésére a RegionRank projekt segítségével. Több különböző keresés eredményeit összevetve kellene meghozni a döntést az említett helyzetekben.*

A jelenleg rendelkezésre álló keresési szolgáltatásokkal szemben egy sokkal gyorsabb és kényelmesebb megoldást kínál a RegionRank alkalmazás. A kiválasztott szempontok alapján generál egy hőtérképet, amelyen jól láthatóak a városnak azok a részei, amelyek legjobban megfelelnek a keresés szempontjainak, illetve azok a részek is, ahol „próbálkozni sem érdemes”. A felhasználó saját igényei szerint választhatja ki a kereséshez alapot szolgáltató szempontokat és súlyozhatja is az egyes szempontok fontosságát.

Néhány lehetséges kritérium kategória, amelyek alapján a keresés megvalósulhat:

- környéken található POI-k (iskolák, kórházak, színházak, múzeumok, mozik, vendéglők, bevásárlóközpontok, szabadidős és sportkomplexumok stb)
- Közlekedési hálózat (főútak, tömegközlekedés, bicikliútak, balesetekkel kapcsolatos és egyéb mutatók stb)

- Környezet (szenyezettség, zajszint, zöldövezetek, mobilhálózati lefedettség, közbiztonsággal, árakkal kapcsolatos és egyéb mutatók stb)
- Lakosság (politikai nézetek, vallási felekezetek, egészségügyi, iskolázottsággal kapcsolatos és egyéb mutatók stb)
- Időfüggő kritériumok (időjárás, események, időszakos ajánlatok stb)



2. ***ábra****:összetett keresés eredményének hőtérkép alapú megjelenítése (a kolozsvári szorakozóhelyeket, parkokat és folyókat tartalmazó minta adathalmaz alapján a pirossal jelzett területek felelnek meg a legjobban a példában említett keresési kritériumoknak)*

Egy lényeges újítása a projektnek, hogy a térképszolgáltatásokon alapuló RegionRank túllép a hagyományos POI (Point of Interest) fogalmon és bevezeti a ROI (Region of Interest) fogalmát. A keresések alapjaként az egyszerű, földrajzi koordinátákhoz kötött helyek (éttermek, mozik stb) mellett tetszőleges alakzatok által reprezentált régiók is szolgálhatnak (parkok, útszakaszok, adott bűnözési mutatóval jellemezhető városnegyedek stb). A hőtérkép alapú megjelenítés mellett a ROI-k „hagyományos" módon történő megjelenítése is lehetséges, és ennek optimalizálására a szoftver klaszterezési algoritmusokat is alkalmaz.

Jelenleg számos olyan kezdeményezés létezik, amelyek a különböző közérdekű adatok nyilvánossá tételét célozzák. Példaként megemlíthető a European Public Sector Information Program[1], vagy az American Open Government Initiative[2]. Egyre több ország és szervezet igyekszik az ilyen adatokat elérhetővé tenni.

Problémát jelent, hogy a különböző forrásokból származó adatok formátuma nem egységes, nem standardizált és a hozzáférési lehetőségek is eltérőek. A RegionRank projektnek célja, ezt a problémát is áthidalni. Egy egységes API-t biztosít a különböző forrásokból származó, különböző formátumú adatok kezelésére, és könnyen kiterjeszthető, bővíthető új adatforrások bevonásával.

A RegionRank szoftverrendszer egy a fenti céloknak megfelelő prototípus, amely jelenlegi állapotában három nagyobb részből tevődik össze: egy szerver és két webes

[1] http://ec.europa.eu/information_society/policy/psi/index_en.htm
[2] http://www.data.gov/

kliensalkalmazás. A szerver felelős az adatok tárolásáért és biztosítja a megfelelő adathozzáférési réteget. Hozzáférést biztosít a különböző külső adatforrásokhoz, és ezek alapján automatikusan frissíti saját adatbázisát. Tartalmazza az üzleti logika réteget, amely elvégzi a biztosított funkcionalitásokkal kapcsolatos műveleteket, számításokat (keresés, klaszterezés stb). Egy egységes API-t biztosít a kliensalkalmazások számára, megoldva az ezekkel történő kommunikációt.

Az adminisztrátorai számára egy megfelelő webes adminisztrációs felületet biztosít a rendszer. Ennek segítségével megvalósulhat a tárolt adatok menedzsmentje: ROI-k menedzsmentje, külső adatforrások meghatározása, automatikus frissítéssel kapcsolatos és egyéb beállítások, saját adatbázisban tárolt adatok menedzsmentje, statisztikák megjelenítése stb.

A felhasználók számára egy térképszolgáltatáson alapuló webes felület biztosítja a keresési lehetőségeket. Kiválaszthatják és súlyozhatják a keresésükben szerepet játszó szempontokat és ezeknek megfelelően generálhatnak hőtérképeket. A generált térképen bármelyik pontra kattintva részletes információt igényelhetnek az illető területről. A ROI-k hagyományos megjelenítését és a hagyományos keresést (pl. ROI-k/POI-k kiválasztása) is támogatja az alkalmazás, klaszterezési lehetőséget is biztosítva a jobb áttekinthetőség érdekében.

A fentebb leírt RegionRank alkalmazást mutatja be röviden a dolgozat. Első része az alkalmazás megvalósításához felhasznált módszerek, elméleti fogalmak rövid leírását tartalmazza. Második részében a felhasznált fejlesztési eszközöket, technológiákat, mintákat és fejlesztési módszereket mutatja be. A harmadik rész a projekt konkrét leírása, vázlatosan ismerteti a követelményeket, használati eseteket, a projekt architektúráját és tervét, megvalósításának fontosabb részleteit, valamint tartalmaz egy rövid esettanulmányt, amely egyben egy nagyon vázlatos felhasználói dokumentációnak is tekinthető. Végül az eredmények összefoglalása után néhány továbbfejlesztési lehetőség felsorolása következik.

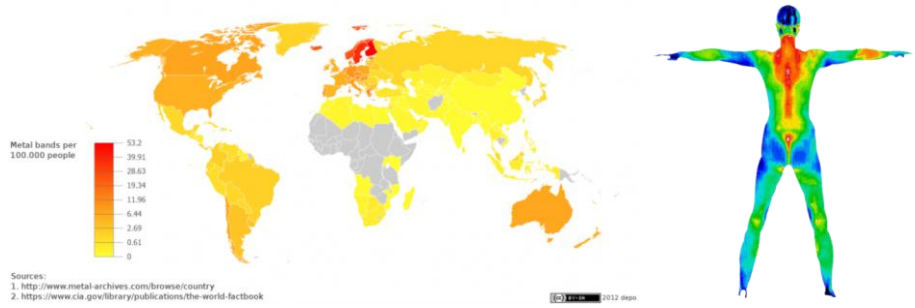
Az alkalmazás alapötlete az EIT ICT Labs által szervezett finnországi Cloud Computing Summer School rendezvény keretein belül került bemutatásra, Szilágyi Péter és társai által, ahol az Aalto Center for Entrepreneurship által bírált Business Model Design Competition versenykategóriában első helyezést ért el. A jelenlegi projekt alapterve Szilágyi Péter és dr. Simon Károly további egyeztetései során alakult ki, és a kolozsvári Codespring Kft. vállalta, hogy támogatja a prototípus fejlesztését. A dolgozat szerzői a Codespring Kft.-nál eltöltött szakmai gyakorlat során sajátították el a prototípus elkészítésében szerepet játszó technológiák és módszerek alapjait, a fejlesztési folyamat a cég megbízott szakembereinek szakmai irányításával, a cég infrastruktúráját igénybe véve történt. A RegionRank prototípus létrejöttéért köszönet illeti Szilágyi Pétert és csapatát, akik az eredeti ötlet gazdái, a szakmai irányítókat, valamint a Codespring Kft.-t.

1. Módszerek, elméleti alapfogalmak

A RegionRank alkalmazás üzleti logikáért felelős komponensei a funkcionalitások biztosításának érdekében számos háttérműveletet és számítást végeznek. A felhasznált módszerek és algoritmusok közül kiemelhetőek a hőtérkép megjelenítéséhez használt módszerek, illetve a megjelenítés optimalizálására alkalmazott klaszterezési algoritmusok.

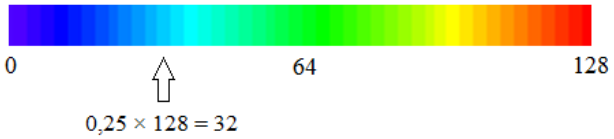
1.1. Hőtérkép

A hőtérkép egy olyan grafikus ábrázolása egy adathalmaznak, amelyen belül az adatok közötti eltéréseket különböző színek vizualizálják. Számos alkalmazási területe létezik, használják földrajzi szintkülönbségek szemléltetésére, statisztikai adatok (pl. népsűrűség) megjelenítésére, az orvostudomány területén stb.



*3. **ábra:** példák hőtérkép alapú ábrázolásra – rock együttesek „sűrűsége" földrajzi régiók szerint, hőtérkép orvosi alkalmazásban (források: http://metalluminati.com ; http://www.sigmahellas.gr)*

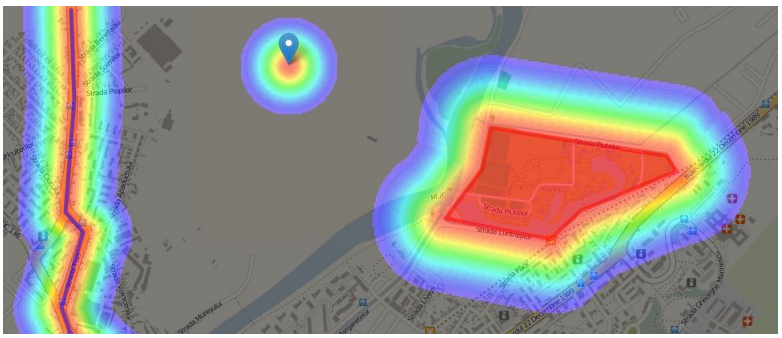
A RegionRank projekt esetében a hőtérkép célja megmutatni a felhasználónak a kiválasztott kategóriáknak megfelelő ROI-k összesített hatását. A hőtérkép szempontjából minden ROI-t négy fontos tulajdonság jellemez: a ROI-nak megfelelő alakzat típusa (POI-t reprezentáló földrajzi pont, adott területet reprezentáló sokszög, vagy törött vonal – pl. útszakaszok reprezentálására), a pozíciója (a ROI-nak megfelelő alakzatot meghatározó pontok koordinátái), a hatósugara (milyen körzetben fejti ki hatását), illetve az értéke (a hatás erőssége). Ezeknek a tulajdonságoknak az ismeretében felépíthető a hőtérkép, amely szemlélteti a ROI-k környezetükre kiváltott együttes hatását.



*4.**ábra**: példa színtérképre, amely a 0,25 értékhez a 32-es színt felelteti meg*

Egy adott koordinátájú pont színét meghatározza a pontra ható ROI-k hatásainak összessége. Egy adott ROI hatása függ a pont és ROI közötti távolságtól. Az adatok alapján kiszámított hatást leíró értéknek egy színtérkép alapján megfeleltethető egy szín.

A ROI-k esetében a különböző típusú alakzatok hatásai különböző módon érvényesülnek, ahogyan ezt az 5. ábra szemlélteti.



*5.**ábra**: ROI-k környezetükre gyakorolt hatása. Balról jobbra: törött vonal által reprezentált ROI (pl. útszakasz vagy folyó), földrajzi pont által reprezentált ROI (POI – pl. étterem, mozi stb), sokszög által reprezentált ROI (pl. park, adott bűnözési mutatóval jellemezhető városnegyed stb)*

A RegionRank projekten belül a térképszolgáltatást a Leaflet biztosítja, amely egy speciális URL segítségével kapja meg a térképet alkotó képeket egy Tile Server-től. Az URL-en keresztül a szerver kap három paramétert, a kép pozícióját meghatározó *x* és *y* paramétereket és a nagyítási szintet meghatározó *z* paramétert. Ezek függvényében visszaküldi a térkép megfelelő részét, egy 256×256 pixel felbontású, png formátumú kép (*tile*) formájában. A szerveren 19 különböző nagyítási szintnek megfelelően tárolva van a világtérkép, az egyes nagyítási szintek megjelenítéséhez szükséges részletességgel, a szinteknek megfelelően felbontva, kis képek formájában. Az egyes nagyítási szinteknek megfelelő felosztást a 1. táblázat szemlélteti.

n	Felosztás	Darabszám
0	1 *tile* (lefedi az egész világtérképet)	1
1	2 x 2 *tile*	4
2	4 x 4 *tile*	16
3	8 x 8 *tile*	64
n	2^n x 2^n *tile*	2^{2n}
16	2^{16} x 2^{16} *tile*	4 294 967 296
17	2^{17} x 2^{17} *tile*	17 179 869 184
18	2^{18} x 2^{18} *tile*	68 719 476 736

1. ***táblázat**: nagyítási szinteknek megfelelő térkép-felosztás és a szükséges tile-ok száma*

Egy ugyanilyen felosztáson alapuló saját Tile Server-t megvalósítva egy új réteg adható hozzá a térképhez. A konkrét megvalósítás bővebben a dolgozat harmadik részében kerül bemutatásra.

1.2. Klaszteranalízis és POI/ROI klaszterezés

A klaszterezés [1] folyamán egy adathalmaz elemeinek „természetes csoportosulásait” azonosítjuk be, az elemeket csoportokba soroljuk valamilyen hasonlósági mérték alapján. Egy-egy csoportot általában egy prototípus vektor, a klaszterközéppont határoz meg. Az eljárásra nagyon sok esetben szükségünk lehet és ennek megfelelően a módszerek, klaszterezési algoritmusok száma is nagy. Rendelkezésünkre állnak klasszikusnak számító, közismert módszerek és nagyon sok új megközelítést, például természetből inspirált

számítástechnikai modelleket (pl. genetikus algoritmusokat) is alkalmaznak az említett feladat megoldására.

A klaszterezési algoritmusokat többféleképpen kategorizálhatjuk. Beszélhetünk például hierarchikus és nem hierarchikus módszerekről (annak függvényében, hogy csak egy adott csoportosítást, vagy egy teljes csoport-hierarchiát keresünk), dinamikus módszerekről (amikor a keresett csoportok száma nem képezi az algoritmus bemenetét), fuzzy módszerekről (amikor egy elem több klaszterhez is tartozhat, egy-egy adott „hozzátartozási" értéknek megfelelően) stb. A hierarchikus módszereket tovább oszthatjuk aszerint, hogy felosztáson, vagy összevonáson alapulnak. Az előbbiek első lépésben egy nagy klaszterbe sorolják az összes elemet, majd lépésenként ezt a klasztert próbálják meg kisebb részekre bontani. Az utóbbiak úgy tekintik, hogy kezdetben minden elem egy-egy külön klasztert képvisel, majd lépésenként olvasztják egybe ezeket a klasztereket, mindaddig, amíg eljutnak egy kívánt felosztásig.

POI-k megjelenítésének esetében is hasznos és ennek megfelelően gyakori a klaszterezési algoritmusok alkalmazása. Több ezer POI megjelenítése ugyanazon a felületen rontaná az átláthatóságot. Klaszterezési algoritmusok segítségével ezek csoportokba rendezhetőek, így optimalizálható a megjelenítés.

A RegionRank projekt esetében ugyanez a klaszterezési algoritmusok használatának motivációja, csak a klaszterezendő halmaz elemei nem egyszerű POI-k, hanem tetszőleges alakzatok által reprezentált régiók, ROI-k (Region Of Interest). Az összehasonlítás alapjául a régiók közötti földrajzi távolság szolgál.

A RegionRank klaszterező komponense a Strategy tervezési mintának megfelelően van megvalósítva, több különböző klaszterezési algoritmust biztosít, és ezeknek cseréje vagy új módszer bevezetése egyszerűen megvalósítható. Jelenleg a projekt két klaszterezési módszert támogat: a *k*-means algoritmust, valamint egy rács-alapú módszert. A fejlesztés során alkalmazva volt még a DBScan sűrűség alapú klaszterezési algoritmus, valamint a *k*-means egy hierarchikus változata, de a prototípus jelenlegi verziója ezeket nem tartalmazza. A későbbi verziókba lehetséges, hogy visszakerül a hierarchikus módszer javított változata, illetve egy dinamikus genetikus klaszterezési algoritmus kipróbálása is szerepel a tervek között.

1.2.1. A *k*-means algoritmus

A klaszterelemzési módszerek közül talán a legnépszerűbb a *k*-means algoritmus [1], amelynek alapverzióján kívül számos más változata (pl. hierarchikus is) létezik. Az alap algoritmus lépései a következők:

1. Véletlenszerűen kiválaszt *k* darab klasztert és meghatározza ezek középpontját.
2. Minden pontot a hozzá legközelebb álló klaszterbe helyez.
3. Újraszámolja a klaszterek középpontját.

4. A 2. és 3. lépéseket ismétli a leállási feltétel teljesüléséig.

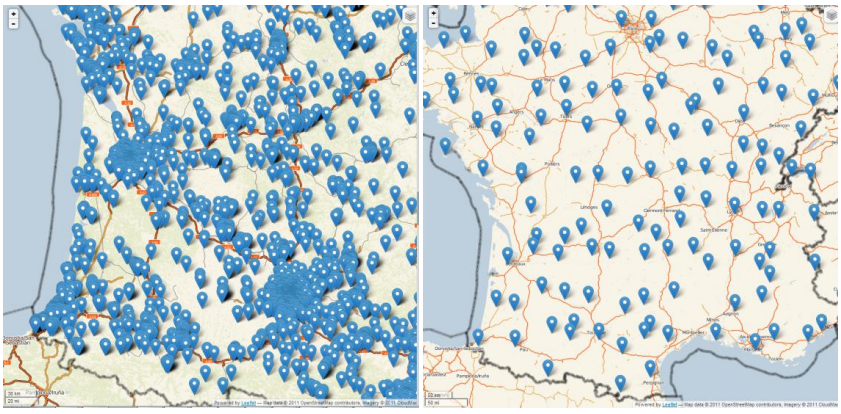
Az algoritmus legnagyobb előnye az egyszerűsége és a sebessége. Ennek ellenére a RegionRank prototípus jelenlegi változatában csak másodlagos algoritmusként szolgál. Ennek oka egyrészt, hogy az algoritmus megköveteli a klaszterek számának ismeretét és ez a változó adatmennyiség miatt megjelenítési problémákhoz vezethet. Másrészt hatékonyság szempontjából is problémák jelentkezhetnek a valós idejű frissítések következtében igényelt újra-klaszterezések esetében.

1.2.2. Rács-alapú klaszterezés

A POI megjelenítő és kereső szolgáltatások esetében gyakran alkalmaznak rács-alapú klaszterezési algoritmusokat [2], amelyek egy más szempontból közelítik meg a feladatot. A keresési teret „dobozokra” osztják fel, ezek egy tartalmazási hierarchiát alkothatnak. A klaszterezendő pontokat ezekbe a „dobozokba” csoportosítják az algoritmusok.

POI-k klaszterezésének esetében az ilyen módszerek gyakorlatilag egy téglalapokból vagy négyzetekből álló rács-szerkezetből indulnak ki, a térkép aktuális nagyításnak megfelelő látható felületét osztják fel. A marker-eket a cellákba csoportosítják, így nem vizsgálják a pontok közti távolságokat, hanem egyszerűen olyan csoportokba rendezik azokat, amelyek a megjelenítés szempontjából optimálisabbak. Ilyen algoritmus áll például a GoogleMaps MarkerClusterer eszköze mögött is. Természetesen ezeknek az algoritmusoknak is megvannak a hátrányai, például előfordulhat, hogy egymáshoz nagyon közeli, de a cellák határainak közelében elhelyezkedő pontok különböző klaszterekbe kerülnek stb. Másrészről a módszer nagyon hatékony, és a megjelenítés szempontjából is megfelelő.

A kipróbált algoritmusok közül a RegionRank esetében is ez a megközelítés bizonyult a legjobbnak, mind a megjelenítés, mind a hatékonyság szempontjából.



6. **ábra**: *az Active Fonera telefontársaság antennáinak megjelenítése a RegionRank segítségével - klaszterezés nélkül, illetve klaszterezve (felhasznált adatforrás: http://poiplaza.com)*

2. Felhasznált technológiák és eszközök

A RegionRank Java technológiákra épül, alapja a Spring vállalati keretrendszer [3], a MongoDB [4] dokumentum alapú NoSQL adatbázist használja adattárolásra, a Vaadin keretrendszert alkalmazza a webes felületek létrehozására, a Leaflet JavaScript könyvtárat a térképszolgáltatás elérésére, a Google Guava – Event Bus könyvtárat a komponensek közötti eseményalapú kommunikáció megvalósítására. A felsoroltakon kívül további API-kat és keretrendszereket alkalmaz a szolgáltatási és kommunikációs rétegeken belül, az adatok validálására stb.

2.1. A Spring keretrendszer

A Spring[3] közkedvelt keretrendszer, amelyet Java vállalati (enterprise) alkalmazások fejlesztése során használnak.

A keretrendszer a következő alapvető részekből áll:

- Core: a core csomag képezi az egész keretrendszer alapját, ez tartalmazza az Inversion of Control (*IoC*) konténert, illetve lehetőséget biztosít a komponensek közötti függőségek meghatározására (*dependency injection*). Ez a csomag bármilyen Java-ban írt alkalmazás által használható.
- AOP: a Spring tartalmaz egy saját keretrendszert, amely az aspektusorientált programozási paradigmát támogatja. Több megkötés jellemzi az AspectJ nyelvhez képest, ugyanakkor több szempontból előnyösebb, egyszerűbb és kényelmesebb, hatékonyabb fejlesztést tesz lehetővé. Megjegyzendő, hogy AspectJ is használható Spring alkalmazásokon belül.
- DAO: egy JDBC fölötti absztrakciós réteget biztosít a programozók számára.
- ORM: az ORM csomag egy integrációs réteget biztosít a népszerű objektum-relációs keretrendszerek számára, támogatja például a JPA specifikációt implementáló keretrendszerek használatát (Hibernate, EclipseLink, TopLink stb).
- JEE: támogatja a Java szerver-oldali komponensmodelljének, az EJB-nek a használatát, valamint további Java specifikációknak megfelelő technológiák alkalmazását, mint például JMX (Java Management Extensions), JCA (Java Connector Architecture), JMS (Java Message Service) stb.
- Spring Web: ez a csomag tartalmazza a web alkalmazásokhoz szükséges funkcionalitásokat, mint például az IoC konténer inicializálása servlet listenerek és web-orientált alkalmazás-kontextus segítségével, webes MVC keretrendszerek használata (JSF, Struts stb) stb.

A felsorolt alapvető csomagokon kívül a Spring további keretrendszereket is biztosít, többek között támogatja NoSQL adatbázisrendszerek elérését (pl. Spring DATA – MongoDB), mobilkészülékekre fejlesztett webes alkalmazások készítését (Spring

[3] http://www.springsource.org/

Mobile), szociális hálózatokkal kapcsolatos szolgáltatásokat nyújtó rendszerekkel a kommunikációt (Spring Social) stb.

A RegionRank projekten belül a Spring Core biztosítja a komponensek menedzsmentjét, valamint az IoC konténert, amely a komponensek közötti szoros függőségek feloldásának szempontjából fontos. A már lefordított Java osztályok bájtkódjának fordítási idejű módosítása, különböző funkcionalitások (pl. az @Autowired annotációval ellátott komponensek konfigurálása) beillesztésének érdekében, az aspektusorientált paradigmának megfelelően, a Spring AOP segítségével történik. Néhány EJB specifikus funkcionalitás (pl. példányosítás utáni inicializálási műveletek) a Spring JEE által biztosított. Az adathozzáférési réteg a további Spring modulokat használó Spring DATA – MongoDB-n alapszik.

2.2. MongoDB

A MongoDB[4] a 10gen[5] által fejlesztett, dokumentumorientált, NoSQL adatbázis-kezelő rendszer. A relációs adatbázisokkal ellentétben, a MongoDB az előre meghatározott szerkezetű táblák helyett, kollekciókat (*collection*) és dokumentumokat (*documents*) használ. Az adatokat BSON[6] formátumban tárolja, ezáltal a dokumentumok szerkezete egy adott kollekción belül is különbözhet. Az egyik nagyon fontos tulajdonsága, amiért a térképszolgáltatásokat nyújtó alkalmazások esetében közkedvelt, hogy támogatja a 2d indexelést (*geospatial index*). Ezáltal nagyon gyorsan lehet keresni földrajzi koordinátákhoz kötött adatok esetében.

A RegionRank projekt esetében az előbb említett tulajdonság fontos szerepet játszott az adatbázis-kezelő rendszer kiválasztásában. További pozitívum, hogy a MongoDB támogatja az osztott módon történő adattárolást, így nagymennyiségű adat esetén is hatékony tárolási lehetőséget biztosít, egy a fejlesztők számára transzparens (de szabályozható) módon.

2.3. Google Guava – EventBus

Az EventBus[7], a Google által írt, Google Guava[8] részét képező könyvtár, melynek célja egy "feliratkozás-értesítés" (*publish-subscribe*) típusú folyamatközi kommunikációra (*interprocess communication*) alkalmas mechanizmus megvalósítása, amely lehetővé teszi, hogy az egyes komponensek értesítsék egymást egy bizonyos eseményről, anélkül, hogy szoros kapcsolat lenne közöttük (anélkül, hogy „tudnának egymásról"). Lévén, hogy a RegionRank több modulból épül fel, hasznos egy ilyen megoldás használata, annak érdekében, hogy a modulok közötti fölösleges szoros függőségek kiszűrhetőek legyenek.

[4] http://www.mongodb.org/
[5] http://www.10gen.com/
[6] http://bsonspec.org/
[7] http://code.google.com/p/guava-libraries/wiki/EventBusExplained
[8] http://code.google.com/p/guava-libraries/

2.4. Vaadin

A Vaadin[9] [5] a Vaadin LTD által (egykori nevén: IT Mill) fejlesztett nyílt forráskódú Java alapú web-alkalmazás keretrendszer, amely modern, interaktív és gyors válaszidőt garantáló oldalak fejlesztését segíti Java-ban. Más JavaScript és böngésző kiegészítőkre alapozó megoldásokkal szemben, a Vaadin egy teljes komponenskészletet bocsájt a rendelkezésre, amellyel hatékonyan fel lehet építeni egy teljes web alkalmazás felhasználói felületét. Amennyiben a keretrendszer által biztosított komponenskészlet mégsem bizonyulna elegendőnek, a fejlesztők által könnyen bővíthető HTML5[10], JavaScript és GWT[11] segítségével. A kliens (böngésző) és szerver oldal közti kommunikáció Ajax[12] alapú, JSON[13] objektumok segítségével valósul meg.

A fentebb felsorolt tulajdonságoknak köszönhető, hogy a RegionRank projekt fejlesztése a Vaadin keretrendszer felhasználásával történik.

2.5. Térképszolgáltatások

A RegionRank projekt fejlesztése során több térképszolgáltatás is alkalmazva volt. A ROI klaszterező módszerek kifejlesztését célzó RedDog alprojekt esetében először a GoogleMaps volt használva, majd ennek korlátozásai miatt a csapat áttért OpenLayers használatára. Végül a RegionRank jelenlegi változatában a Leaflet alkalmazása mellett született döntés, annak számos előnye miatt.

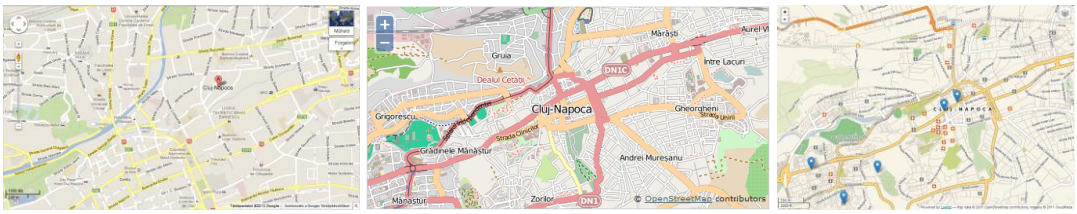
A GoogleMaps[14] a Google által fejlesztett ingyenes térképszolgáltatás, amelyhez a Google egy API-t[15] is biztosít a fejlesztők számára. Ez az API fejlett és sokoldalú, azonban az OpenLayers-el ellentétben a használata korlátozott és csak a fizetős ügyfelek használhatják minden funkcióját.

Az Open Layers[16] az Open Source Geospatial Foundation (*OSGeo*)[17] által fejlesztett JavaScript függvénykönyvtár, amely térképadatok megjelenítésére szolgál. Szolgáltatásait tekintve, nagyban hasonlít a Google Maps API[18]-ra, azzal a különbséggel, hogy az Open Layers szolgáltatásai teljesen ingyenesek.

A Leaflet[19] Vladimir Agafonkin[20] ukrán informatikus és egy közreműködő fejlesztői csapat által fejlesztett nyílt forráskódú JavaScript könyvtár, melyet olyan ismert alkalmazások

[9] https://vaadin.com/
[10] http://www.whatwg.org/specs/web-apps/current-work/multipage/
[11] https://developers.google.com/web-toolkit/
[12] http://www.w3schools.com/ajax/
[13] http://www.json.org/
[14] https://maps.google.com/
[15] https://developers.google.com/maps/
[16] http://openlayers.org/
[17] http://www.osgeo.org/
[18] https://developers.google.com/maps/
[19] http://leafletjs.com/index.html
[20] http://agafonkin.com/en/

használnak, mint például a Flickr[21], a Foursquare[22], a Meetup[23] vagy az OpenStreetMap (*OSM*)[24]. A fejlesztői egy olyan könyvtárat szerettek volna létrehozni, amely egyszerű, de a nagy többség igényeit kielégíti és külső (*third-party*) modulok segítségével könnyen bővíthető. Amennyiben valaki új funkcionalitások beépítését igényelné, a csapat rendelkezésre bocsájt egy oldalt[25], ahol a fejlesztők kifejthetik véleményüket, illetve megszavazhatják egyes funkcionalitások beépítésének lehetőségét.



*8. **ábra**: Kolozsvár központja a GoogleMaps, OpenLayers és Leaflet térképeken*

A RegionRank projekt megírása során a Leaflet használata mellett szólt, hogy egy jól dokumentált API-val rendelkezik, olyan nagyvállalatok közreműködésével fejlesztik, mint a CloudMade[26] vagy az Open Street Map, külső modulok segítségével egyszerűen bővíthető, illetve a HTML5-nek és CSS3-nak köszönhetően az asztali gépektől a mobil platformokig, mindenen könnyen használható.

2.6. További technológiák

A RegionRank projekt esetében a naplózás az SLF4J[27] és a log4j[28] keretrendszerek segítségével történik. Az SLF4J egy általános, úgynevezett „facade" naplózási keretrendszer, amely lehetőséget ad több naplózási keretrendszer (a RegionRank esetében log4j) alkalmazására.

A szoftver biztonsági mechanizmusáért és a jogosultságok kezeléséért az Apache Shiro[29] keretrendszer felelős. A Shiro végzi a felhasználók és adminisztrátorok hitelesítését, a munkamenetek kezelését, a jogosultságok ellenőrzését. Kezelni tud különböző adatforrásokból (például LDAP, Active Directory, JDBC) érkező, felhasználókkal kapcsolatos adatokat, és bővíthető saját adatforrásokat kezelő modulokkal.

Az adatbázisban tárolt adatok helyességének érdekében, a mentés előtt szükséges ezek ellenőrzése. Erre a célra a RegionRank a JSR-303 szabványra épülő Bean Validation keretrendszert használja.

[21] http://www.flickr.com/
[22] https://foursquare.com/
[23] http://www.meetup.com/find/
[24] http://www.openstreetmap.org/
[25] http://leaflet.uservoice.com/forums/150880-ideas-and-suggestions-for-leaflet
[26] http://cloudmade.com/
[27] http://www.slf4j.org/
[28] http://logging.apache.org/log4j/
[29] http://shiro.apache.org/

A publikus szolgáltatások megvalósítása REST alapú web szolgáltatások segítségével történik, a JAX-RS[33] (*Java API for RESTful Services*) API-ra épülő Jersey[34] keretrendszert alkalmazva. A Jersey szolgáltatja a Leaflet által használt hőtérkép felépítéséhez szükséges *tile*-okat, emellett az adatok importálásában és exportálásában van szerepe. Az adatok hordozható formátumba (*JSON/XML*) alakítását a JAX-B[35] API alapú Jackson[36] keretrendszer oldja meg.

2.7. Felhasznált eszközök

A RegionRank prototípusa egy összetett szoftverrendszer, amelynek fejlesztése során kulcsfontosságú volt a megfelelő fejlesztői eszközök használata és ezek helyes kiválasztása, valamint a használatukhoz szükséges infrastruktúra. A fejlesztési folyamat támogatására a csapat projektmenedzsment, verziókövető, hibakövető, build automatizáló és kódelemző eszközöket használt. Ezeken kívül természetesen szükségesek voltak további alapvető eszközök: megfelelő fejlesztői és tervezői környezet, GUI tervező eszköz, web-szerver stb.

Amikor több fejlesztő egyszerre dolgozik ugyanazon a projekten, elengedhetetlen a verziókövető szoftverek használata. Ezek a rendszerek lehetőséget biztosítanak a forráskód változásainak nyomon követésére, archiválására, különböző kiadások kezelésére, jogosultságkezelésre, fejlesztési ágak, illetve címkék kezelésére. Két nagy csoportba lehet besorolni a verziókövetőket: a kliens-szerver modellre épülő központosított verziókövető rendszerek (pl. CVS, SVN) és az osztott verziókövetők (pl. Git, Mercurial).

A RegionRank projekt megírásakor a választás az osztott verziókövetőkre esett, azon belül is a Mercurialra[37], amely egy Pythonban írt ingyenes osztott verziókövető rendszer. Számos előnye van a központosított verziókövetőkhöz képest, ezek indokolták a választást. A fontosabb különbségek:

- A "változtatási halmazok" fogalom bevezetésével az összefésülési (*merge*) műveletek hatékonyabbá válnak.
- Kisebb változtatásokat is fel lehet tölteni a lokális tárolóba, a verziókövetés előnyeit élvezve, anélkül, hogy a többi fejlesztőnek bármilyen hátránya származna ebből.
- A lokális tárolóknak köszönhetően hálózati problémák esetén is használható a rendszer. Kisebb adatforgalmat generál, ezáltal gyorsabb is.

Minden projekt esetében kritikus követelmény az idő és erőforrások megfelelő beosztása. A projektmenedzsment szoftverek ezt a feladatot segítik, olyan hasznos funkciókat biztosítva, mint például egy adott munkamenet létrehozása, illetve elérhetővé tétele a csapat számára, az egyes feladatok kiosztása és a haladás nyomon követése. Fórumokat biztosítanak a felmerülő kérdések megtárgyalására, wiki-t a különböző dokumentumok tárolására,

[33] https://jax-rs-spec.java.net/
[34] https://jersey.java.net/
[35] http://jaxb.java.net
[36] http://jackson.codehaus.org/
[37] http://mercurial.selenic.com/

megosztására és megosztott módon történő szerkesztésére. Az integrált hibajelentő és hibakövető rendszerek hatékonyabbá teszik a hibajavítást. A modern projektmenedzsment szoftverek a felsoroltakon kívül számos más funkcionalitást is biztosítanak: e-mail értesítések a határidőkről, változásokról, hírfolyam stb.

A RegionRank fejlesztése során használt BitBucket[38] egy webes projektmenedzsment és hibakövető szoftver, amely egyben verziókövető rendszert is biztosít (a Mercurial és Git osztott verziókövetőket támogatja).

Összetettebb projektek esetében, ahol több komponensből épül fel a szoftver és a modulok közötti függőségi gráf bonyolult, elengedhetetlen valamilyen build és függőség-kezelő rendszer használata. A különböző programozási nyelvekhez és környezetekhez számos ilyen eszköz létezik. Ilyenek például az Apache Ant[39], Apache Maven[40] és Apache Ivy[41] (Java), a NAnt és NMake (.Net), a Meique (C, C++), a Cabal (Haskell) stb.

A RegionRank projekt esetében a döntés az Apache Maven rendszerre esett, amely a build menedzsment mellett beépülő modulok és a függősségek kezelésével is foglalkozik. A Maven konvencióknak köszönhetően a build folyamatot leíró állomány (pom.xml) sokkal kompaktabb és átláthatóbb, mint például az Ant esetében.

Egy szoftver esetében a kód minősége nagyon fontos szerepet játszik és ebbe a lehető legoptimálisabb megoldások kiválasztása mellett beleértendő a kód átláthatósága és érthetősége is, a vonatkozó szabályok betartása, a konvenciók követése. Számos eszköz áll a rendelkezésre, amelynek feladata, hogy elemezze a kódot és figyelmeztessen az esetleges hibákra, és rossz megoldásokra. Ilyen eszközök például a Squale, a CodePro AnalytiX, a Checkstyle, a Kalistick, vagy a RegionRank projekten belül használt Sonar[42].

A Sonar a SonarSource által fejlesztett, Java-ban írt platform-független, ingyenes kódminőség ellenőrző szoftver, amely jelenleg több mint húsz programozási nyelvet támogat, és hét területet fed le az ellenőrzés szempontjából: duplikátum ellenőrzés, teszt-lefedettség, komplexitás, potenciális hibák, megfelelő dokumentáltság, architektúra és design, illetve kódolási szabályok, konvenciók.

A RegionRank fejlesztése során számos további fejlesztési eszköz használatára szükség volt. A fejlesztői környezet szerepét a NetBeans IDE[43] töltötte be, web-szervernek a Jetty[44] és az Apache Tomcat[45] volt használva. A felhasználói felület tervezésére a Balsamiq[46] tervezőeszköz szolgáltatott megoldást.

[38] https://bitbucket.org/
[39] http://ant.apache.org/
[40] http://maven.apache.org/
[41] http://ant.apache.org/ivy/
[42] http://www.sonarsource.org/
[43] https://netbeans.org/
[44] http://www.eclipse.org/jetty/
[45] http://tomcat.apache.org/
[46] http://www.balsamiq.com/

A NetBeans egy Java nyelven írt ingyenes integrált fejlesztői környezet, amelyet 1996 óta fejlesztenek. Elsődlegesen Java alkalmazások fejlesztésére használják, de támogatja a PHP, C, C++, JavaScript nyelveket is.

Az Apache Tomcat Java nyelvben írt, Apache Licenc 2.0 alatt terjesztett web szerver, amely egyaránt implementálja a Java servlet és JSP technológiákat. Jelenleg a 7.x verziónál tart és támogatja a Servlet 3.0, JSP 2.2, valamint az EL 2.2 specifikációkat.

A Jetty egy Java nyelvben írt, Apache Licenc 2.0 alatt terjesztett web szerver és servlet konténer. 2008 óta a hetes verzió a HTTP protokollon kívül támogatja a WebSocket es SPDY protokollokat is.

Nemcsak az alkalmazás architektúráját fontos jól megtervezni, hanem a felhasználói felületet is, amelynek jól átgondoltnak, letisztultnak, átláthatónak, szoftver-ergonómiai szempontból megfelelőnek kell lennie. Ebben szolgálhat segítségünkre a Balsamiq, amellyel könnyen megtervezhető a felhasználói felületet, anélkül, hogy a fölösleges részletek, színek és design elemek elvonnák figyelmünket a funkcionalitásokról és a szoftver-ergonómiai szempontokról.

A RegionRank rendszerben felhasznált ROI klaszterező modul prototípusa egy külön projekt keretében készült el, ahol néhány más eszköz is használva volt. A projektmenedzsment rendszer szerepét ebben az esetben a RedMine töltötte be, a verziókövetés SVN segítségével történt, fejlesztői környezetként az Eclipse IDE volt használva.

3. A RegionRank projekt

A következőkben a dolgozat röviden ismerteti a RegionRank projekttel kapcsolatos követelményeket, vázlatosan bemutatja a szoftver használati eseteit, a rendszeren belüli központi entitásokat és azok kapcsolatait, és röviden tárgyalja a megvalósítás fontosabb elemeit.

3.1. Követelmények

Az alábbiakban bemutatott követelményspecifikáció vázlatos, csak a funkcionális követelmények rövid összefoglalója, amely nem tér ki a különböző nem funkcionális követelményekre, és a felsorolt követelmények részletes tárgyalását sem tartalmazza. A teljes követelményspecifikáció a projekt dokumentációjának részét képezi.

3.1.1. RegionRank Server

A szerver feladata a RegionRank Admin UI és RegionRank Client UI, illetve a RegionRank API alrendszerek kiszolgálása, kapcsolatteremtés az adatbázissal, illetve az alkalmazáslogika megvalósítása. A szerver fontosabb funkcionalitásai:

- A modellobjektumok lekérése, mentése és törlése az adatbázisból. Ezek az objektumok a rendszer központi entitásait reprezentálják. Lehetnek régiók, felhasználók, ROI-k, kategóriák, ROI halmazok és tulajdonságok. Megjegyzés: a projekt kontextusában a régiók azok a területek (pl. városok), amelyekre a rendszer keresési lehetőséget biztosít, ROI adatbázisokat tárol.
- Speciális adatbázis lekérdezések megvalósítása (pl. adott régióhoz tartozó, adott kategóriájú ROI-k lekérése).
- Különböző adatforrásokból származó bemeneti adatok feldolgozása (XML, CSV stb).
- Felhasználók beléptetése, kiléptetése, jogosultságkezelés.
- ROI-k klaszterezése.

3.1.2. RegionRank Admin UI

Az Admin UI alrendszer felelős az adminisztrációs felület megvalósításáért. Fontosabb funkcionalitásai:

- Belépési felület biztosítása az adminisztrátorok számára.
- Régiók, ROI-k, kategóriák listázása, hozzáadási, szerkesztési, törlési felületek biztosítása. Régiók és ROI-k hozzáadásánál egy Leaflet térképen alapuló szerkesztő biztosítása, amelyen megrajzolható az adott régió vagy ROI.
- Feltöltési felület biztosítása hordozható, strukturált fájlformátumok számára.

3.1.3. RegionRank Client UI

A Client UI alrendszer szerepe a kliensalkalmazás felhasználói felületének megvalósítása. Fontosabb funkcionalitásai:

- Regisztrációs felület biztosítása új felhasználók számára.
- Belépési felület biztosítása a felhasználói fiókkal rendelkező felhasználók számára.
- Lekérdezési felület biztosítása, amelyen a felhasználók kiválaszthatják a keresésben szerepet játszó kategóriákat és beállíthatják azok fontosságát.
- Hőtérkép megjelenítése a Leaflet API segítségével

3.1.4. RegionRank WidgetSet

A saját komponenseket foglalja magába. Fontosabb funkcionalitásai:

- Egy térkép komponens megvalósítása, amely megjelenít egy hőtérképet.
- Egy térkép komponens, amely lehetőséget biztosít régiók és ROI-k szerkesztésére.

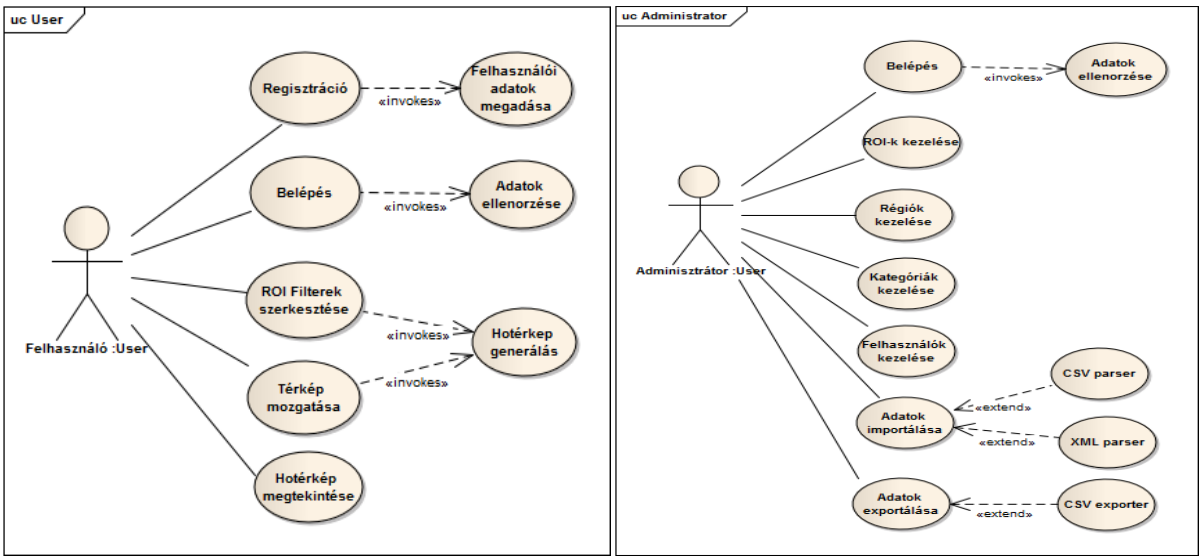
3.1.5. RegionRank API

Az API felelős a nyilvános szolgáltatások biztosításáért:

- Hőtérkép generálása és tile-ok szolgáltatása a Leaflet API számára.
- Az adatbázis tartalmának kimentése XML állományba.

3.2. Használati esetek

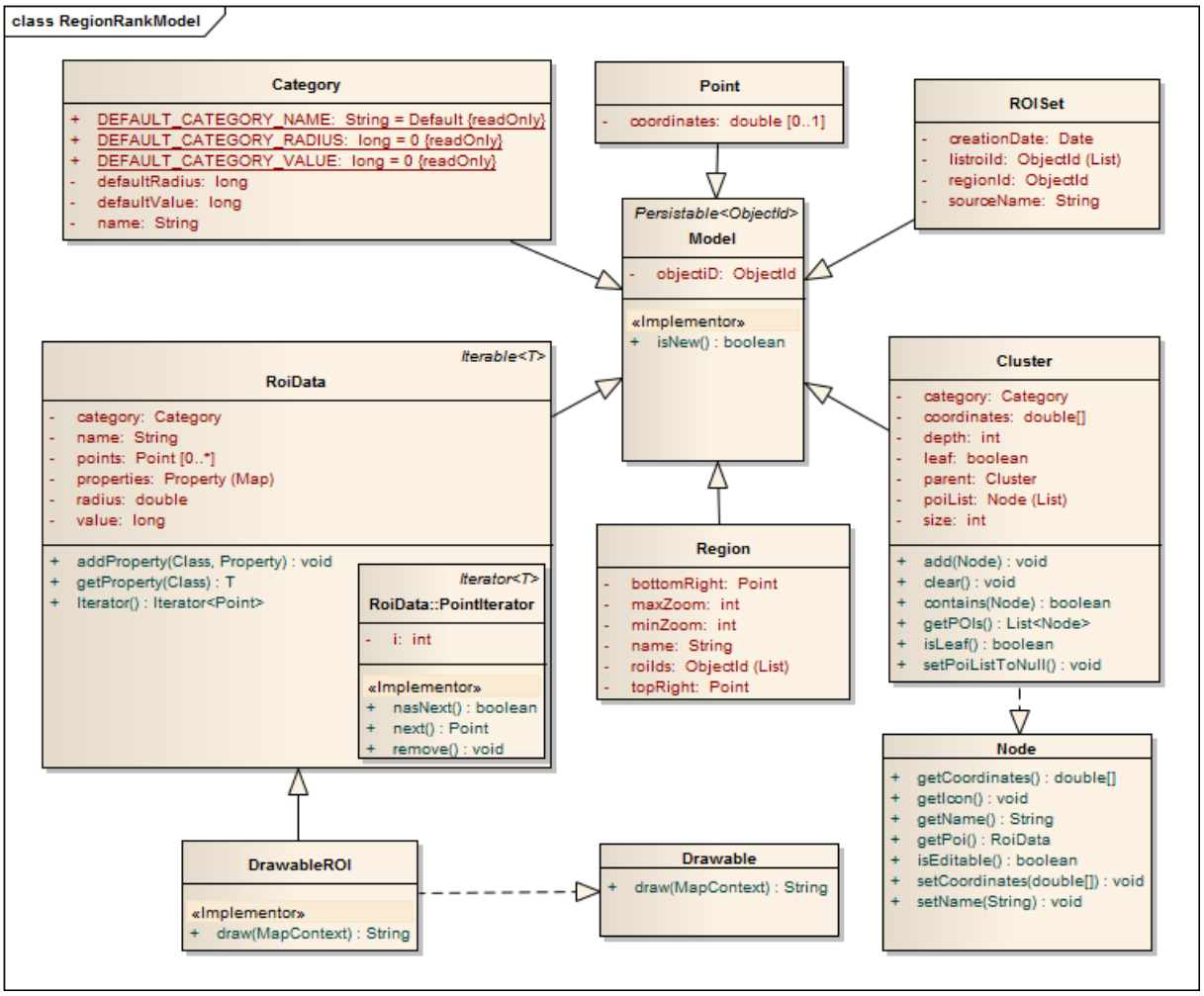
A 9. ábrán két leegyszerűsített *use-case* diagram szemlélteti a rendszer fontosabb használati eseteit a felhasználók és adminisztrátorok szemszögéből.



9. ***ábra:*** *Felhasználók és adminisztrátorok számára elérhető funkcionalitások összefoglalója egyszerűsített* ***use case*** *diagramok segítségével.*

3.3. Környezeti elemzés

A *com.regionrank.model* csomagban találhatóak a szoftver központi entitásait reprezentáló Java Bean-ek, ahogyan ezt a 10. ábra szemlélteti.



*10. **ábra**: A modell osztályok vázlatos diagramja (a konstruktorok, getter és setter metódusok, valamint az Object ősosztály újradefiniált metódusai nem jelennek meg a diagramon)*

A modell osztályok közös tulajdonsága, hogy mindannyian privát adattagokkal rendelkeznek, publikus getter és setter metódusaik vannak és megvalósítják a *java.io.Serializable* interfészt. Konkrétabban az entitások közös ősosztályaként szolgáló *Model* osztály megvalósítja a *Persistable* interfészt, amely a *Serializable* kiterjesztése.

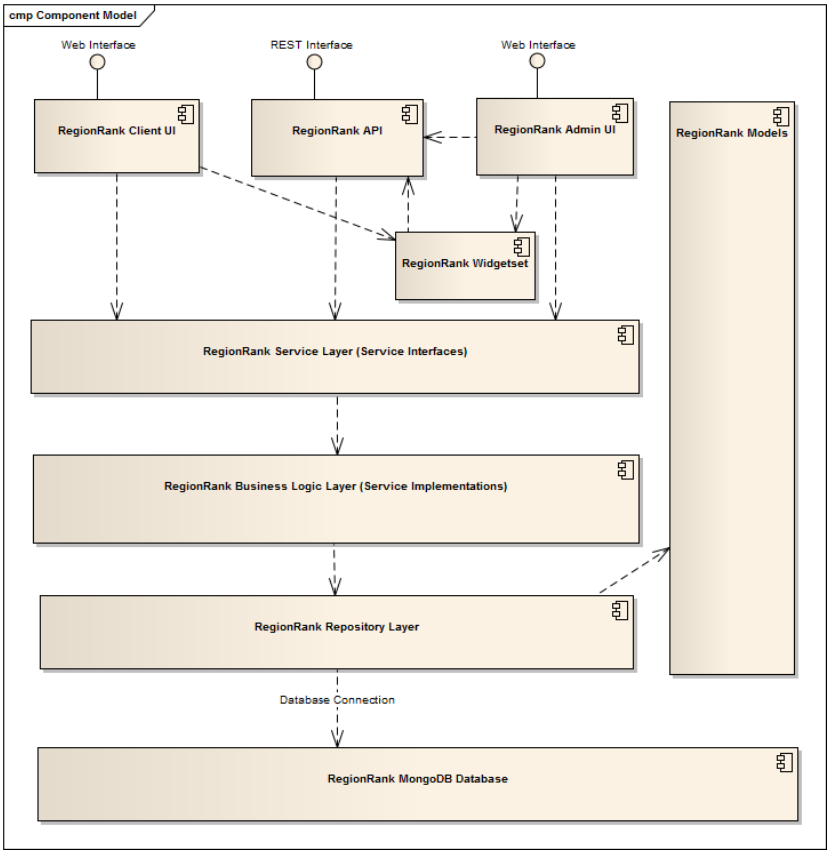
A *RoiData* modell osztály példányai tárolják a ROI-kra vonatkozó információkat. A ROI rendelkezik egy névvel (*name*), kategóriával (*category*), egy ponthalmazzal (*points*), egy hatósugárral (*radius*), egy számértékkel (*value*) és egy tetszőleges számú, tetszőleges típusú (fordítási időben meghatározható) meta-információt hordozó adattaggal (*properties*).

A *ROISet* az egyszerre importált ROI-k könnyű kezelését hivatott megoldani. A *Cluster* és *Node* osztályok a ROI-k klaszterezéséhez szükségesek. A *Node* interfész biztosítja a lehetőséget a térképen megjelenítendő ROI-k/klaszterek egységes módon történő kezelésére.

Az objektumok adatbázisba történő mentésének a módjáról a Spring DATA-MONGODB keretrendszerből importált annotációk gondoskodnak. Minden adatbázisba mentendő típus @Document annotációval van ellátva, így az osztályok függetlenek az adatbázistól, de a *Repository* réteg megkapja a perzisztenciával kapcsolatos műveletek elvégzéséhez szükséges információkat. A modellek a *javax.xml.bind* csomagból származó annotációkkal is el vannak látva, így egyszerűen importálhatóak XML állományokból és exportálhatóak XML állományokba. Az adatok helyességéért a JSR-303 szabványnak megfelelő Bean Validation API-t használ az alkalmazás.

3.4. Architektúra

Az 11. ábra a RegionRank fontosabb alrendszereit és azok kapcsolatait szemlélteti.



*11. **ábra**: A RegionRank architektúrája*

A RegionRank az adatokat egy dokumentumorientált NoSQL adatbázisban tárolja, amely flexibilisen kezeli az adatkészletet, így egyszerűen változtathatóak az adatformátumok.

A *RegionRank Model* tartalmazza a szoftver központi entitásait reprezentáló osztályokat. Ezeket minden alrendszer használja, így mindegyik függ ettől az alrendszertől.

A *RegionRank Repository Layer* feladata az entitások adatbázisba írása, illetve adatbázisból történő kiolvasása, a már bevitt adatok módosítása, esetleg törlése.

A *RegionRank Service Layer-en* keresztül elérhetőek a rendszer szolgáltatásai a többi alrendszer számára. A szolgáltatások megvalósításához szükséges műveleteket a *RegionRank Business Logic Layer* komponensei végzik. Ez az alrendszer a *Repository* rétegtől kapja a műveletek elvégzéséhez szükséges adatokat, és az eredményeket a *Service* rétegen keresztül továbbítja a klienseknek.

A *RegionRank Client UI* alrendszer a szoftver felhasználóit hivatott kiszolgálni. Megjeleníti a grafikus felhasználói felületet, feldolgozza a felhasználó kéréseit, továbbítja ezeket a *Service Layer*-nek, a visszakapott adatokat a felhasználó által értelmezhető formában megjeleníti. A „külvilággal" a Web Interface-en keresztül, HTTP protokoll alapján kommunikál.

A *RegionRank Admin UI* feladata a rendszer karbantartóinak lehetőséget adni a rendszer adatokkal való feltöltésére, adminisztrálására, konfigurálására. A *Client UI*-hoz hasonlóan grafikus felhasználói felületet biztosít és weben keresztül kommunikál a felhasználójával.

A *RegionRank API* lehetőséget ad arra, hogy a szoftver funkcionalitásait más alkalmazás is elérje. Jól meghatározott beérkező kérésekre az API megfelelő választ küld. Az API-n keresztül kérhetünk térképre vonatkozó információt, importálhatunk vagy exportálhatunk adatokat, ha rendelkezünk megfelelő jogosultsággal az illető műveletekhez.

3.5. Megvalósítás

A RegionRank projekt megvalósítása során számos tervezési minta, keretrendszer, eszköz és módszer használatára volt szükség. A tervezés és fejlesztés során nagy hangsúlyt kapott a bővíthetőség és továbbfejleszthetőség, a széles körben használt, hosszú távú és megfelelő terméktámogatással rendelkező keretrendszerek részesültek előnyben.

3.5.1. RegionRank Backend

A *com.regionrank.backend.model* csomagban található osztályok függetlenítik az alkalmazás által használt adatbázis-kezelő rendszer tárolási mechanizmusát és formátumát az alkalmazásban használt reprezentációtól. Ennek előnye, hogy az alkalmazás nagy részében változatlanul kezelhetőek a modellek, az adatbázis-kezelő rendszer esetleges cseréje esetén is. A modellekben nincs megvalósítva bonyolult üzleti logika, de validátorokat tartalmaznak. Ezek végzik az adatok helyességének ellenőrzését a mentés előtt. Minden entitás rendelkezik továbbá egy egyedi azonosító mezővel.

A *com.regionrank.backend.repository* csomagban interfészek találhatóak, amelyek a *ModelRepository*-ból származnak. Ezek feladata a modellobjektumokkal kapcsolatos adathozzáférési műveletek meghatározása. A *Repository* réteg a Spring keretrendszer Spring DATA – MONGODB komponensén alapszik. Ez közvetlen kapcsolatban áll a MongoDB

adatbázissal, így az egyedüli adatbázis-kezelő rendszertől függő komponens. A *ModelRepository*-ból származó *Repository* osztályok számára biztosítja a gyakran használt adatbázis metódusokat (pl. *findAll()*, *find(modelid)*, *save(model)* stb). Ha saját lekérdezések definiálása is szükséges, könnyen hozzáadható az interfészhez egy új metódus fejléce, a Spring DATA-MONGODB a metódus szabványos nevéből és paramétereiből felépíti az adatbázis-lekérdezést és visszatéríti ennek eredményét.

Mivel a *Repository* réteg adatbázis-kezelő rendszertől függ, az alkalmazás üzleti logikája egy másik alrendszeren belül van megvalósítva. Az alkalmazás specifikus funkcionalitásait leíró interfészek, illetve az ezeket megvalósító osztályok a *com.backend.service* csomagban találhatóak. A *Service* komponensek felelősek a felhasználók jogosultságainak ellenőrzéséért, a *Repository* rétegtől kapott adatok összegzéséért, feldolgozásáért és értelmezéséért stb. A réteg a Dependency Injection tervezési mintán alapszik, így a szolgáltatások konkrét megvalósításai könnyen cserélhetőek, egyedüli követelmény, hogy megvalósítsák a szolgáltatásnak megfelelő interfészt. A Dependency Injection megvalósítását támogatja a Spring keretrendszer, a függőségek annotációk segítségével konfigurálhatóak.

3.5.2. RegionRank User Interface (UI)

A RegionRank felhasználói felülete két nagy modulra osztható: az átlag felhasználók kiszolgálásáért felelős *Client UI*, és a rendszer karbantartói számára további funkcionalitásokat biztosító *Admin UI*. Mindkét modul megvalósítása a Vaadin keretrendszeren alapszik.

Mind a felhasználói, mind az adminisztrátori felület web böngésző segítségével érhető el, a szerver oldali Java osztályok felhasználásával a Vaadin keretrendszer létrehozza a megfelelő HTML, CSS és JavaScript komponenseket.

A kommunikáció a kliens oldali és szerver oldali komponensek között Ajax alapú, JSON formátumú objektumok segítségével értesítik egymást a kommunikálni kívánó komponensek a változtatásokról.

3.5.3. RegionRank Widgetset

A felhasználói felület tartalmaz egy *Widgetset* nevű modult, amelyben a Vaadin által nem biztosított, saját fejlesztésű komponensek találhatóak. Ez a modul biztosítja a megfelelő komponenst a térképszolgáltatással történő kommunikációhoz és egy speciális, a rendszer karbantartóinak szánt *ShapeEditor* komponenst, amellyel a térképen bármilyen formájú régiót könnyen szerkeszthetünk.

A *Widgetset* tartalmazza a Leaflet függvénykönyvtárat, amely megkönnyíti a térképen végzett műveletek megvalósítását és nagyon egyszerűen bővíthető saját rétegekkel (*layer*) is.

A hőtérkép generálás a RegionRank esetében is egy külön térkép rétegre történik, a *RegionRank API* segítségével.

3.5.4. RegionRank API

A *RegionRank API* feladata azoknak a szolgáltatásoknak a biztosítása, amelyek RESTful modellen alapulnak. A rendszerbe jelenleg beépített ilyen szolgáltatások a *TileService* és a *DataService*.

A *TileService* a hőtérkép generálásáért felel. A hőtérképet a gyors betöltés és egyszerű gyorsítótárazás (*cache*-elés) érdekében 256×256 pixel nagyságú PNG kiterjesztésű képekben adja vissza. Megfelelő hardver infrastruktúra segítségével több szerveren is futtatható, nagyméretű cache memóriával, így tehermentesíteni lehet a *Backendet*.

A *DataService* az adatbázisban levő adatok exportálására és külső adatforrásokban elérhető adatok importálására szolgál és ugyancsak REST API-n keresztül érhető el.

Az API komponenseinek fejlesztése a JAX-RS szabványra épülő Jersey keretrendszer segítségével történt. Az XML alapú importálást és exportálást, illetve kommunikációt a JAX-B szabványra épülő Jackson keretrendszer támogatja. Java objektumokat XML formátumra tud alakítani (*marshalling*), és a szabványnak megfelelő XML alapján Java objektumokat tud létrehozni (*unmarshalling*)

3.5.5. Hőtérkép

A RegionRank projekten belül a hőtérkép generálását a TileService REST szolgáltatás biztosítja. Egy adott tile kérése a http://localhost:8080/api/tiles/get/?x={x}&y={y}&z={z} URL-en keresztül történik, amely magába foglalja a három szükséges paramétert (koordináták és nagyítási szint). Ezek segítségével a BoundingBox osztály tile2boundingBox(int x, int y, int zoom) metódusa kiszámolja a megfelelő tile északi, déli, keleti és nyugati határait. A határok ismeretében lekérdezhetőek az adott tile-ra esetlegesen ható ROI-k.

A térkép pontjait kiszínező algoritmus a 256×256 pixeles *tile* minden pontjára kiszámolja a pontra ható ROI-k hatásainak intenzitását, összegzi ezeket az értékeket és az eredmény alapján színezi ki a pontot.

A ROI-k hatásai az adott pontra azok ponttól mért távolságaitól függnek. A számításokat elvégző modul a pont és ROI között mért távolságot egy 0 és 1 közötti számmá skálázza, majd ez alapján meghatározza a pontnak megfelelő színt egy színtérkép alapján.

A modul a Strategy tervezési mintára [6] épül, így a konkrét számításokat elvégző algoritmus könnyen cserélhető. Különböző módszerek alkalmazhatóak a hatás meghatározására. A RegionRank jelenlegi változata két implementációt biztosít:

1. A pont ROI-tól mért távolságát egy lineáris képlet segítségével skálázza:

```
Distance = (radius – distanceOf(Point(i,j), roi)) / radius,
```

ahol *radius* a ROI hatósugara, *distanceOf(a, b)* a függvény, amely visszatéríti az *a* és *b* közötti távolságot, *Point(i, j)* az *(i, j)* pozícióban levő pixel alapján visszatéríti az adott pont koordinátáit.

2. A pont távolságát egy Gauss-függvény skálázza 0 és 1 közé:

$$Distance = e^{-\alpha * d * d},$$

$$d = (radius - distanceOf(Point(i,j), roi)) / radius,$$

ahol *radius* a ROI hatósugara, *distanceOf(a, b)* a függvény, amely visszatéríti az *a* és *b* közötti távolságot, *Point(i, j)* az *(i, j)* pozícióban levő pixel alapján visszatéríti az adott pont koordinátáit és α egy skálázási paraméter (az exponenciális függvény értéktartományának átskálázására szolgál).

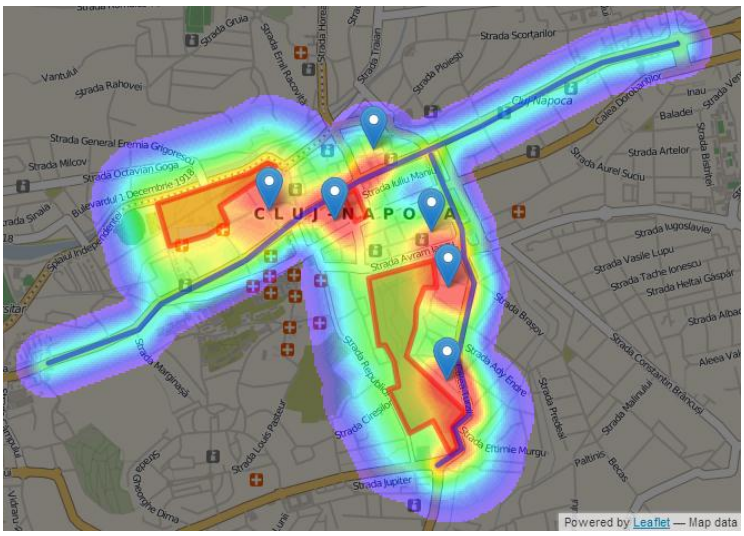
A műveleteket elvégző modulnak egy adott pont esetében az arra a pontra potenciálisan ható (adott körzetben található) minden ROI-ra ki kell számolnia annak az adott ponttól mért távolságát. Ha ez a távolság kisebb, mint a ROI hatósugara, akkor a megfelelő távolságfüggvény használatával megkapja a pontban a ROI hatását és a hatásnak megfelelően kiszínezi a pontot.

Egyszerű, pontok által reprezentált ROI-k esetében a pont távolsága a ROI-tól a két koordinátapár földrajzi távolsága lesz. Törött vonal alakú ROI-nál a pont és a ROI közötti távolságot a következő távolságok minimuma adja:

- a pont távolsága a ROI töréspontjaitól;
- a pont távolsága a ROI szakaszaitól, ha a pont vetülete rajta van az adott szakaszon.

Sokszög alakú ROI esetében, az algoritmus ellenőrzi, hogy a pont benne van-e a ROI által reprezentált régióban. Ha igen, akkor a ROI hatása maximális ebben a pontban. Ellenkező esetben a törött vonal esetében használt módszer alapján számol.

Ezek a szolgáltatások a RegionRank projekten belüli *Geo* osztály metódusain keresztül érhetőek el.



*12.**ábra**: egy területre ható ROI-k együttes hatása által meghatározott hőtérkép (az adathalmaz 6 pont által reprezentált, és 2-2 törött vonal, illetve sokszög által reprezentált ROI-t tartalmaz).*

3.6. Használati esettanulmány

A RegionRank működésének szemléltetésére tekintsük a következő példát: egy turista átautózik Kolozsváron és tenne egy egy órás pihenőt, sétával egybekötve, miközben ebédelne is. Olyan városrész érdekli tehát, amely főúthoz közeli, van a közelben étterem és zöldövezet.

*13.**ábra**: Adminisztrációs felületen található ROI szerkesztő*

Az alkalmazásnak rendelkezésére áll az adathalmaz, amely tartalmazza a megfelelő kategóriákat. A kategóriákon belüli minden ROI esetében tárolva vannak a hozzátartozó alakzatot meghatározó pontok, a hatósugara, az érték, amely megmutatja hatásának erősségét, illetve specifikus tulajdonságai. A felhasználói felületen belül megjeleníthető a ROI-k listája, ahogyan azt az 14. ábra is szemlélteti, és szűrési, illetve rendezési lehetőségek is biztosítottak.

NAME	POINT	CATEGORY	RADIUS	VALUE
Bulgakov	{(46.77,23.59)}	Szórakozóhely	0.3	60
Calea Turzi	{(46.77,23.60)(46.77,23.60)(46.77,23.60)(46.77,23.60)(46.76,23.60)(46.76,23.60)(46.76,23.60)(46.76,23.60)(46.76,23.60)}	Ut	0.2	40
Klausen	{(46.77,23.59)}	Szórakozóhely	0.25	50
Kozponti Park	{(46.77,23.59)(46.77,23.59)(46.77,23.59)(46.77,23.59)(46.77,23.59)}	Park	0.4	60
La Liga	{(46.76,23.59)}	Szórakozóhely	0.25	50
Mesele Vesele	{(46.77,23.58)}	Szórakozóhely	0.2	40
Motilor/December 1	{(46.76,23.57)(46.76,23.57)(46.76,23.57)(46.77,23.58)(46.77,23.59)(46.77,23.59)(46.77,23.60)(46.78,23.61)(46.78,23.61)(46.78,23.61)}	Ut	0.2	40
Park	{(46.77,23.57)(46.77,23.58)(46.77,23.58)(46.77,23.58)(46.77,23.58)(46.77,23.58)(46.77,23.58)(46.77,23.58)(46.77,23.58)(46.77,23.57)}	Park	0.4	80
Park	{(46.77,23.60)(46.77,23.60)(46.77,23.60)(46.76,23.59)(46.76,23.59)(46.76,23.59)(46.76,23.60)(46.76,23.60)(46.76,23.60)(46.76,23.60)(46.76,23.60)(46.76,23.59)(46.76,23.59)(46.76,23.59)(46.76,23.59)(46.76,23.59)(46.76,23.59)(46.77,23.59)(46.77,23.60)}	Park	0.4	60
Quo Vadis	{(46.77,23.58)}	Szórakozóhely	0.2	40
Shanghai	{(46.76,23.60)}	Szórakozóhely	0.2	35
Tokyo Sushi	{(46.76,23.58)}	Szórakozóhely	0.2	40
Tramvay	{(46.77,23.59)}	Szórakozóhely	0.2	55
Viking	{(46.76,23.58)}	Szórakozóhely	0.2	40

*14. **ábra**:a ROI-k táblázata*

A kliens felület lehetőséget ad az adathalmaz hagyományos módon történő megjelenítésére, valamint hőtérkép megjelenítésére is, ahogyan azt az 15. ábra szemlélteti. Szintén adottak szűrési lehetőségek, a felhasználó egy ROI-t kiválasztva megtekintheti az ahhoz kapcsolódó információkat, továbbá klaszterezési lehetőséget is biztosít a rendszer.

*15. **ábra**:a ROI-k a térképen és a megfelelő hőtérkép*

A felhasználó a felületen kiválaszthatja azokat a ROI kategóriákat, amelyek számára a keresés szempontjából fontosak és súlyozhatja azok fontosságát. A keresés eredményét egy hőtérkép formájában kapja meg, az általa meghatározott feltételek szerint. A térképnek azok a részei kell magasabb értéket mutassanak, ahol megtalálhatóak a kívánt szolgáltatások.

Az 15. ábrán látható az említett példának megfelelő keresés eredménye. A felhasználó kritériumainak megfelelő területek meleg színekkel, a kevésbé megfelelőek hideg színekkel vannak reprezentálva (az alapértelmezett színskála a felhasználó igényeinek megfelelően változtatható).

Következtetések és továbbfejlesztési lehetőségek

A RegionRank projekt keretein belül sikerült megvalósítani egy új típusú, térkép alapú internetes keresési szolgáltatást. A fejlesztés alatt sikerült kiterjeszteni a klasszikusnak számító POI fogalmat és bevezetni az általánosabb ROI fogalmat. A keresések alapjául földrajzi pontok által reprezentált helyszínek mellett tetszőleges alakzatok által reprezentált régiók is szolgálhatnak. Az alkalmazás támogatja az elterjedtebb adatformátumok importálását és exportálását, több klaszterezési módszert és több különböző térképszolgáltatást.

Sikerült egy új megközelítés szerint, könnyen értelmezhető módon, hőtérkép formájában megjeleníteni a keresési eredményeket, egy bárki számára elérhető, kényelmesen használható webes felületen. Egy összetettebb, több kritérium szerint történő keresés esetében nem szükséges külön elvégezni az egyes kritériumoknak megfelelő kereséseket, majd összevetni és értelmezni azok eredményeit, mivel ezt az alkalmazás a hatások összesítése által megoldja. Ezen kívül a kritériumok súlyozhatóak, így a keresés teljes egészében a felhasználói igényeinek megfelelően testreszabható.

A RegionRank projekt jelen állapotában az alapötletet megvalósító demó verzió. A projekt tervezése és fejlesztése során számos új funkcionalitás szükségessége felmerült:

- Mivel az alkalmazás sok felhasználót hivatott kiszolgálni, felmerül a gyorsítótár (*cache*) használatának lehetősége, amely tehermentesítené a szervert.
- A felhasználók keresési eredményének optimalizálása érdekében hasznos lehet egy értékelési rendszer megvalósítása. A felhasználók rangsorolhatják a ROI-kat, emellett lehetőséget kaphatnak keresés után visszajelzések küldésére, amelyek tudatában az alkalmazás képes lesz a későbbiekben pontosabb válaszokat generálni.
- A későbbiekben elképzelhető a RegionRank alkalmazás felhő alapú (*cloud-based*) szolgáltatássá alakítása.
- Az alkalmazás bővíthető olyan modulokkal, amelyek bizonyos interneten található publikus adatforrásokat automatikusan bejárva adott sablonok alapján értelmezik és feldolgozzák az elérhető adatokat.

Hivatkozások

[1] Guojun Gan, Chaoqun Ma, Jianhong Wu, *Data Clustering: Theory, Algorithms, and Applications*, SIAM, Society for Industrial and Applied Mathematics, 2007.

[2] Jiawei Han, Micheline Kamber & Jian Pei, *Data Mining: Concepts and Techniques 3th Edition*, Morgan Kaufmann, 2011.

[3] Rod Johnson , Juergen Hoeller és társai, *Spring Framework Reference Documentation*, 2004-2012.

[4] ***, *MongoDB Documentation*, 2013, (a 10gen, Inc. kiadásában).

[5] Marko Grönroos, *Book of Vaadin: Vaadin 7 Edition*, 2013, (a Vaadin LTD. kiadásában).

[6] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software,* 1994.