MIB II.-92/2007/2012.

SZAKDOLGOZAT

KÁKONYI ROLAND

2012

Pécsi Tudományegyetem
Pollack Mihály Műszaki és Informatikai Kar
Mérnök Informatikus Szak

SZAKDOLGOZAT

Sugárzásmérési adatok megjelenítése térképen

Készítette: Kákonyi Roland
Témavezető: Zidarics Zoltán
Konzulens: Bertók Attila
Pécs

2012
PÉCSI TUDOMÁNYEGYETEM

POLLACK MIHÁLY MŰSZAKI
ÉS INFORMATIKAI KAR
Mérnök Informatikus Szak

Szakdolgozat száma:

MIB II.-92/2007/2012.

SZAKDOLGOZAT FELADAT

KÁKONYI ROLAND
hallgató részére

A záróvizsgát megelőzően szakdolgozatot kell benyújtania, amelynek témáját és feladatait az alábbiak szerint határozom meg:

Téma: Sugárzásmérési adatok megjelenítése térképen

Feladat:
- Mikrokontrolleres adatgyűjtőből származó adathalmaz tárolása
- Adatgyűjtő paraméterezése
- Mért koordinátákhoz tartozó sugárzási érték pontonkénti megjelenítése
- Kvázi ekvivalens pontokra görbe illesztés
- Az így megkapott görbék alapján súlypont (sugárzás forrása) meghatározása

A szakdolgozat készítéséért felelős tanszék: Rendszer és Szoftvertechnológia Tanszék

Külső konzulens: Bertók Attila
munkahelye: Webstar Csoport Kft.

Témavezető: Zidarics Zoltán
munkahelye: PTE-PMMK Automatizálási Tanszék

Pécs, 2011. szeptember 26.

Dr. Szakonyi Lajos
op. szakvezető

HALLGATÓI NYILATKOZAT

Alulírott szigorló hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye. A felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Egyéb jelentősebb segítséget nem vettem igénybe.
Az elkészült szakdolgozatban talált eredményeket a főiskola, a feladatot kiíró intézmény saját céljaira térítés nélkül felhasználhatja.

Pécs, 2012. június 6.

.....

hallgató aláírása

Köszönetnyilvánítás

Ezúton szeretném megragadni az alkalmat arra, hogy köszönetemet és tiszteletemet fejezzem ki mindenkinek, aki a diplomamunkám elkészítéséhez nagyban hozzájárult.

Szeretném kinyilvánítani köszönetemet Zidarics Zoltánnak, aki szakértelmével, végtelen hosszú türelmével nagyban hozzájárult ahhoz, hogy ez a dolgozat létrejöjjön.

Végül, de nem utolsó sorban szeretném kifejezni megbecsülésemet, szívből jövő szeretetemet és köszönetemet családtagjaimnak és a barátnőmnek, akik szeretetükkel, segítségükkel és bíztatásukkal mindvégig támaszt nyújtottak PTE-s tanulmányaim során. Köszönöm!

1. Feladat specifikáció

A szakdolgozatom célja egy olyan alkalmazás fejlesztése, amely képes feldolgozni, relációs adatbázisban tárolni és megjeleníteni mikrokontrolleres adatgyűjtőből származó sugárzásmérési adatokat. Ezeken túl képes a korábban feldolgozott és tárolt adatokat a megjelenítéshez visszatölteni. A megjelenítést webes felületen keresztül egy térkép segítségével valósítsa meg, ezáltal platform független legyen. A feldolgozás során az alkalmazásnak képesnek kell lennie a kvázi ekvivalens pontok alapján, a sugárzás valószínűsíthető súlypontját, forrását meghatároznia, majd megjelölnie azt a webes felületen található térképen.

Célkitűzésem a megvalósított megoldás bemutatása az alkalmazás működésének részletes bemutatásán valamint a használt matematikai módszerek és algoritmusok ismertetésén keresztül.

A szakdolgozatom további célja a felhasznált technológiák és a fejlesztést segítő eszközök, módszertanok bemutatása. Rengeteg hasznos, a fejlesztést hatékonyabbá és gyorsabbá tevő eszköz és technológia létezik, ezek közül is bemutatok néhányat a teljesség igénye nélkül.

2. Felhasznált eszközök, technológiák és a fejlesztői környezet áttekintése

2.1. Az alkalmazás technológiai alapjai

Az alkalmazás kódja 2 fő részre osztható, kliens és szerver[1] oldali kódra.
A szerver oldali kódja a PHP[2] programozási nyelv 5.2.17-es verziójában készült, amit egy Apache HTTP Szerver[3] szolgál ki a kapcsolódó klienseknek.
A kliens oldali kód JavaScript nyelven íródott, ezt a felhasználó által használt web böngésző dolgozza fel.
Az alkalmazás megjelenítéséhez HTML5[4] jelölő nyelvet használtam.
A felhasználói élmény növeléséhez kettő izgalmas JavaScript plugin-t használtam,
a jQuery[5] 1.7-es és a jQuery-impromptu[6] 3.2-es verzióját. Ezek segítségével az alkalmazás teljes funkcionalitása kihasználható az oldal újratöltése nélkül, így DHTML oldalt alkotva.
Az adatok tárolását egy SQLite relációs adatbázis támogatja.

2.2. Általánosan a fejlesztői környezetről

A fejlesztői környezetet egy Windows 7 operációs rendszert futtató számítógépen építettem fel. A kódolást a NetBeans[7] integrált fejlesztői környezet 7.1.2-es verziójában végeztem. Az adatbázis kezelésére, az SQLite Manager nevű Firefox kiegészítőt használtam - ezt az eszközt a későbbiekben részletesen bemutatom. A verziókövetésre a git[8] nevű verziókövető keretrendszert használtam. A verziókövető rendszereket és a git-et részletesebben a 2.4-es fejezetben mutatom be.

2.3. Az adatbázis motor

Az adatok hatékony tárolásához és visszakereséséhez elengedhetetlen egy adatbázis motor, amely végrehajtja az írási, olvasási valamint a keresési műveleteket a tárolt adatokon.

2.3.1. SQLite bemutatása

Az SQLite[9] egy beágyazott (embedded) relációs adatbázis kezelő, azaz az őt használó szoftverhez linkelve lehet használni, nem egy külön SQL szerver. Nincs felhasználó adatbázisa. Ettől eltekintve nagyrészt megvalósítja az SQL-92[10] szabványt. Olyan esetekben érdemes megfontolni az alkalmazását, amikor a helyi rendszeren szeretnénk az adatainkat tárolni. Az SQLite-ot neves szoftverek és eszközök használják: MAC OS, Solaris 10, Skype, iPhone, Firefox. A fejlesztők szlogenje szerint használjuk ott ezt a könyvtárat, ahol egyébként a fopen()[11] parancsot használnánk. Gondoljunk arra, hogy egy hordozható eszközhöz írt szoftvernek általában pont ilyen adatbázis kezelőre van szüksége.
Adatbázisok esetén az ACID[12] mozaikszó az Atomicity (atomicitás), Consistency (konzisztencia), Isolation (izoláció), és Durability (tartósság) rövidítése. Ezek az adatbázis kezelő rendszer tranzakció feldolgozó képességeinek alapelemei. Enélkül az adatbázis integritása nem garantálható, így a tranzakció kezelés támogatott ebben a környezetben is.

Részlegesen megvalósítja a triggereket és a legtöbb komplex/összetett lekérdezést. Az SQLite szokatlan típuskezelést használ az SQL adatbázis-kezelőhöz: egy adattípus nem a táblaoszlopaihoz, hanem egyedi értékekhez van hozzárendelve, más szóval dinamikus típuskezelést használ, gyengén típusos adatkezelés mellett. Amennyiben string típusú adat beilleszthető integer oszlopba, akkor a SQLite először a stringet integerre konvertálja, ha az oszlop preferált típusa integer. Ez nagyobb rugalmasságot ad az oszlopoknak, ami hasznos lehet dinamikus típuskezelésű script-nyelvekben való alkalmazás esetén, azonban ez a technika nem vihető át más SQL adatbázis-kezelőkbe. Az SQLite képtelen atipikus adatbázisokban található szigorúan típusos oszlopok kezelésére.

Ugyanazt az adatbázist több processz és szál használhatja egyidejűleg problémamentesen. Az olvasási kérelmek kiszolgálása párhuzamosan történik. Az írási kérelmek végrehajtása akkor történik meg, amikor nincs folyamatban más kérelem kiszolgálása, egyébként az írási kérelem sikertelen lesz és hibakóddal tér vissza, illetve lehetőség van egy beállítható várakozási időelteltével a kérelem ismétlésére. Ez a konkurens hozzáférési állapot megváltoztatható ideiglenes táblák használata esetén.

Az SQLite számos programnyelvből használható, így BASIC, C, C++, Common Lisp, Java,C#, Visual Basic .NET, Delphi, Curl, Lua, Tcl,R, PHP, Perl, Ruby, Objective-C (Mac OS Xen), Python, newLisp, Haskell, OCaml, Smalltalk és Scheme nyelvekhez rendelkezik illesztő felülettel.

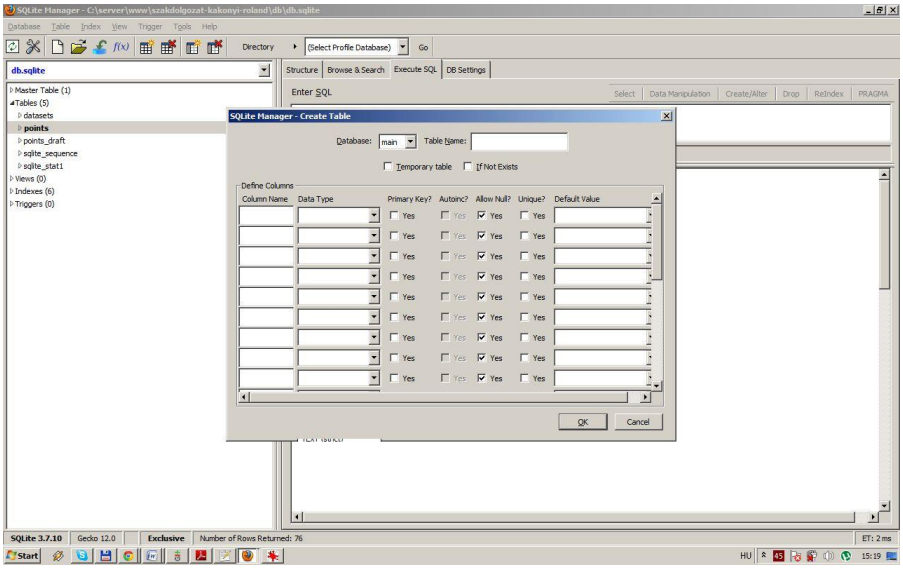
Érdekesség és eltérés a MySQL adatbázis motorhoz képest, hogy egy *autoincrement* mező létrehozásához elég csupán elsődleges kulcsnak és *INTEGER* típusúnak állítani a kívánt mezőt, ezek után ez a mező automatikusan növekszik minden új rekordban.

2.3.2. Választás indoklása

Egy olyan adatbázis motorra volt szükség, amely kis erőforrásigénnyel rendelkezik, de képes a megfelelő szolgáltatások biztosítására. A választás az SQLite adatbázis motorra esett. A használt php interpreterhez elérhető legmagasabb verziószámú, 2.8.17-es SQLite adatbázis motort választottam. Az SQLite választása mellett döntött továbbá a hordozhatósága, egyszerű kezelhetősége, külön szolgáltatástól, szervertől való függetlensége. Továbbá a szakdolgozatom keretein belül fejlesztett alkalmazás teljesítmény és a képesség igényeinek teljesen megfelel.

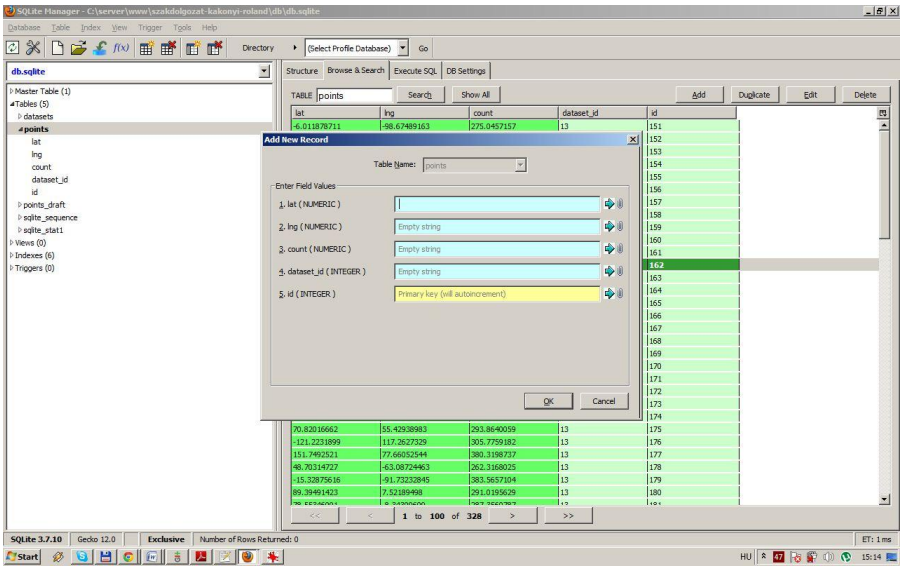
2.3.3. SQLite Manager, mint adatbázis kezelő felület

A szakdolgozatban bemutatott alkalmazás fejlesztése során a használt adatbázis létrehozására, módosítására, monitorozására és ellenőrzésére az SQLite Manager nevű ingyenes Firefox bővítmény 0.7.7-es változatát használtam [13][14]. Teljes körű támogatást nyújt az SQLite adatbázisok létrehozására és kezelésére.
A legfontosabb felületei a következőek:



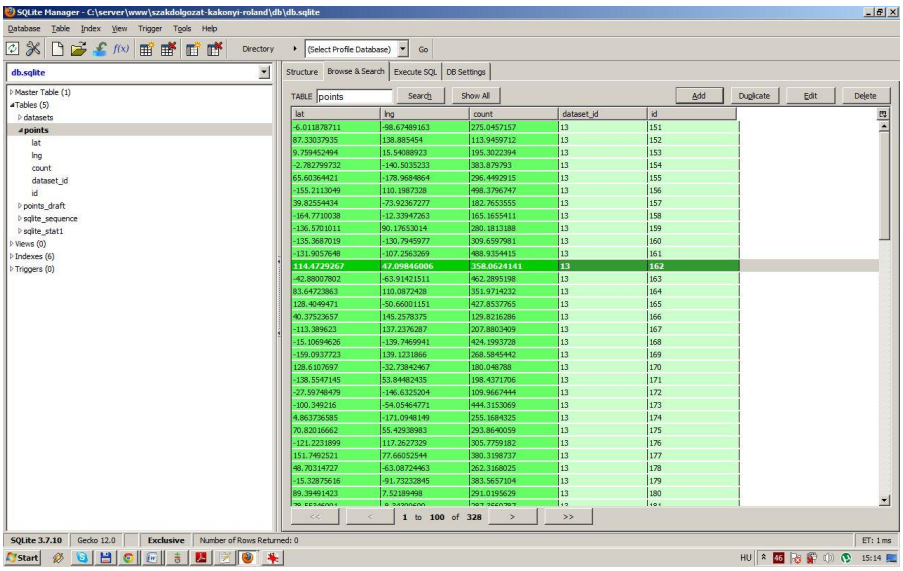
1. ábra Tábla létrehozása

Az 1. ábrán látható az alkalmazás tábla létrehozás nézete. Itt létrehozhatunk új táblákat a kívánt mezőkkel és azok tulajdonságaival.



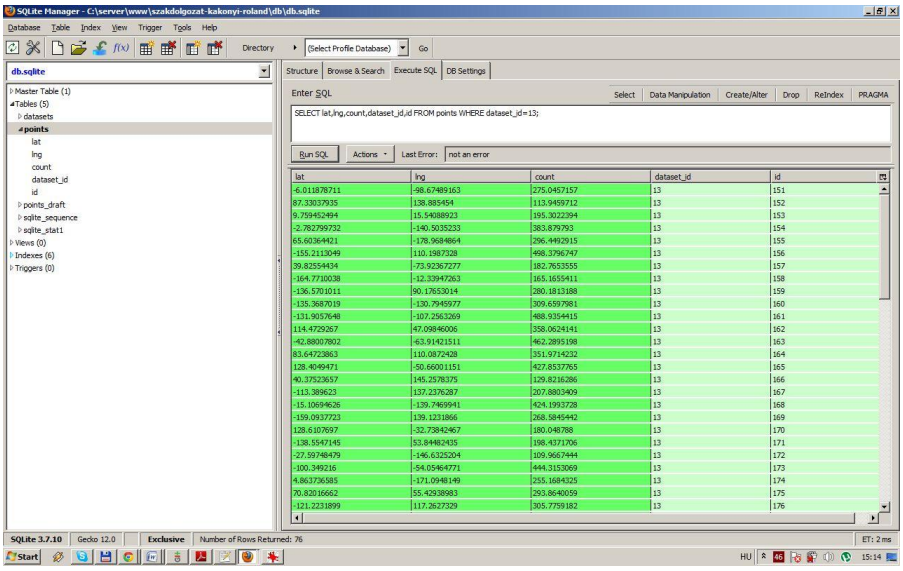
2. ábra Rekord hozzáadása egy táblához

A 2. ábrán egy rekordot adhatunk hozzá az adatbázis egy táblájához. Hozzáadás előtt az eszköz ellenőrzi a tábla rekordjainak mezőire vonatkozó feltételek meglétét.



3. ábra Tábla adatainak böngészése

A 3. ábrán látható felületen a kiválasztott tábla adatait böngészhetjük, szűrhetjük vagy módosíthatjuk egyéni feltételek alapján.



4. ábra Lekérdezés futtatása és annak eredménye

Az SQLite Manager lekérdezés felületét a 4. ábra mutatja. Itt futtathatunk egyéni lekérdezéseket, amelyek eredménye táblázatosan jelenik meg. Hiba esetén egy felugró ablakban értesíti a felhasználót.

2.4. Verziókövető rendszerek

Egy alkalmazás fejlesztése során rengeteg eszköz, technológia és módszertan segítheti a hatékony munkát. Ezek közül az egyik legfontosabb a verziókövető rendszerek használata.

2.4.1. Verziókövető rendszerek általánosan

A verziókövető rendszerek [15] arra szolgálnak, hogy a programkódunk minden változatát elmentsék és segítsék a korábbi verziókra való visszatérést és a csoportmunkát. Láthatjuk ki melyik fájlt módosította, mikor, miért, a változtatások pedig könnyen összevethetőek egymással. A szakdolgozatom során egyedül csináltam az alkalmazást, mégis jó hasznát vettem a verziókövetésnek, mert így két gépen is tudtam haladni a fejlesztéssel, attól függően, hogy épp hol voltam, valamint utólag is nyomon tudom követni, mikor mit csináltam.

Csoportmunka során pedig elengedhetetlen a verziókövetés, mert egy remek segítség abban, hogy ne kelljen egymásra várni, hanem egyszerre lehessen dolgozni különböző részeken, sőt, akár ugyan azon a fájlon is.

A verziókövetéshez szükséges egy szerver, ami a verziókezelésbe vont fájlokat tárolja. Első alkalommal feltöltünk egy induló változatot és utána minden egyes újbóli feltöltésnél a rendszer azt fogja eltárolni, hogy melyik fájlok változtak és miben. Ezzel a módszerrel biztosítja azt, hogy lépésenként láthassuk, hogy hogyan változtak a fájlok és bármelyik verzióra vissza tudjunk térni.

A verziókövető rendszerek általában kliens-szerver alapúak: a fejlesztők a szerveren tárolt ún. *repository*-ból jutnak hozzá a fájlokhoz és a változtatásaikat is először a *repository*-ba küldik be (ez a *commit* művelet), a többi fejlesztő már a *repostory*-ból jut hozzá a változtatásokhoz.

2.4.2. Verziókövető rendszerek típusai

Két nagy csoportra bonthatjuk a verziókövetőket: elosztott és központosított.

A központosított verziókövetők a régebbiek, ma már kevésbé népszerűek, itt ugyanis minden változás csak egy helyen, a szerveren van tárolva. Ilyen például a *Subversion*. [16]

Az elosztott verziókövető azt jelenti, hogy bár van egy központosított szerver, minden felhasználónál megtalálható a központi szerveren található *repository* tulajdonképpeni teljes archívuma (biztonsági mentése). Egy üzemzavar vagy meghibásodás esetén ezek bármelyike visszatölthető a szerverre, hogy lecserélje a központi szerveren levő példányt vagy pótolja hiányát.

Az elosztott verziókövető rendszer további jellemzője, hogy nem igényel állandó, megbízható internet-kapcsolatot a központi szerverrel, és emiatt sokkal kevesebbszer fordul a szerverhez, mint nem elosztott társai. A legtöbb művelet a helyi *repository*-ra hat, így jóval kisebb hálózati forgalmat generál és az átlagos válaszideje is kisebb. A két legnépszerűbb és legelterjedtebb elosztott verziókövető a *Git[8]* és a *Mercurial[17]*.

2.4.3. Verziókövető rendszerek alapfogalmai

Az alábbiakban a legfontosabb alapfogalmakat és műveleteket ismertetem: [15]

- **Repository** (*tároló*): egy távoli szerver, ezen találhatóak a fájlok
- **Revision** (*változat*): a szerveren található fájlok egy adott változata
- **Commit** (*elkönyvelés*): a helyi változtatások elkönyvelése a továbbításhoz
- **HEAD revision** (*fő változat*): a szerveren található legújabb állapot
- **BASE revision** (*alapváltozat*): a munkakönyvtárban található legfrissebb állapot
- **Differences** (*eltérésmutatás*): két különböző állapot összevetésének naplója
- **Merge** (*összefűzés*): két különböző változat egybefűzése (előfordul, hogy nem sikerül tökéletesen, ilyenkor összetűzés (*conflict*) alakul ki)
- **Conflict** (*összetűzés*): tökéletlen összefűzés (*merge*) során kialakuló helyzet
- **Resolve** (*feloldás*): az összetűzés (*conflict*) megoldása (ilyenkor a munkatárs könyvelés előtt átnézi a konfliktusba került módosításokat és végrehajt egy sikeres összefűzést)
- **Checkout**: a távoli szerver egy adott állapotának lemásolása a gépeden található munkakönyvtárba
- **Working copy** (*munkakönyvtár*): a fájlokat tartalmazó mappa a számítógépeken, ennek tartalma kerül könyvelésre a szerverre
- **Trunk** (*törzs*): a fő fejlesztési ág
- **Branch** (*ág*): a fő ággal és egyéb ágakkal párhuzamosan fejlesztett ágak
- **Tag** (*megjelölt változat*): egy lezárt fejlesztési ág vagy kiadás mappája

2.4.4. A verziókövető rendszer kiválasztása

A fenti ismertetésből már látszik, hogy mindenképp érdemes elosztott rendszert használni, még akkor is, ha csak egyedül fejlesztünk valamit. Én személy szerint használtam már mind a három fentebb említett verziókövetőt, és a szubjektív véleményem alapján a *Git* használata mellett döntöttem. A *Subversion* azért esett ki a választásból, mert nem elosztott és hálózati kapcsolat nélkül nem használható. A *Git* és a *Mercurial* között az döntött, hogy *Git*-et áthatóbban ismertem meg, valamint gyorsabbnak és gyorsabban tanulhatóbbnak mutatkozott, valamint ennek a kezelése számomra szimpatikusabb és egyszerűbbnek tűnik, mint a *Mercurial* kezelése.

2.4.5. Git-hez használatos eszközök bemutatása

A Git használatához a Windows-ra elérhető támogató eszközöket tartalmazó programcsomagot telepítettem.[8]

2.4.6. Git bash

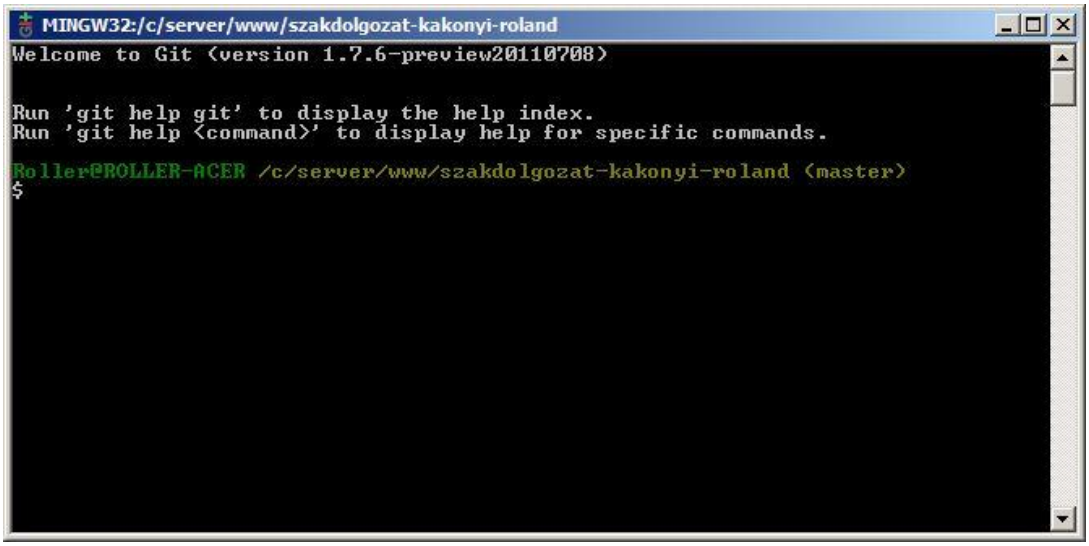
A Git bash-t a 6. ábra mutatja, parancssori eszköz, használata a Linux alapú rendszerekből ismert *BASH*[18] parancsértelmező működése szerint történik.

A Git bash-t használhatjuk a Git Gui vagy a Gitk grafikus felületek elindításához a "git gui" vagy a "gitk" parancsok begépelésével és futtatásával, vagy parancssorból használhatjuk a már említett grafikus felülettel rendelkező segédprogramokban is elérhető funkciókat. A parancsokat a „git” parancsszó után begépelve érhetjük el.

A Git bash-ben használható parancsokat a „git help” parancs futtatásával listázhatjuk, ezek rövid ismertetése az 5. számú táblázatban található.

branch	A fejlesztési ágak listázását, létrehozását és törlését valósítja meg
checkout	Fejlesztési ágak, branchek közötti váltásra szolgál
clone	Lemásol egy repository-t egy új könyvtárba
commit	Változások elkönyvelése a verziókövezéshez
diff	Megmutatja a különbségeket adott commitok vagy branchek között
fetch	Letölti csak a változások listáját a távoli repositoryból
grep	Kilistázza a megadott mintára illeszkedő sorokat
init	Új repositoryt hoz létre vagy újrainicializál egyet
log	Megjeleníti a commit-ok történetét
merge	Összevon egy vagy több fejlesztői ágat
pull	Letölt és összefűz egy fejlesztői ágat a helyi változatával
push	Elküldi az elkönyvelt változásokat a távoli tárolónak
reset	A jelenlegiről visszaáll egy korábbi verzióra
status	Megmutatja a jelenlegi állapot változott, de nem commit-olt fájljainak listáját
tag	Pillanatnyi állapotról mentést, pillanatképet, úgynevezett címkét készít

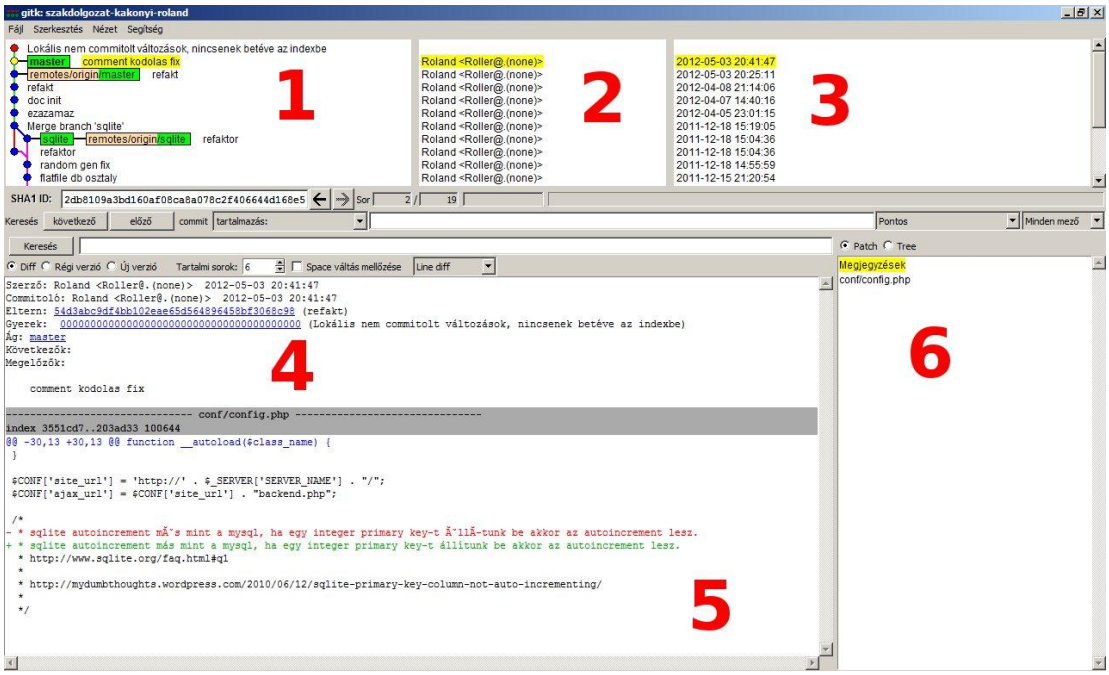
5. táblázat Git bash parancsok



6. ábra Git bash működés közben

2.4.7. Gitk

A fejlesztési történet vizualizálására szolgál. A verziókövetés során eltárolt változások között böngészhetünk, kereshetünk vagy adott esetben visszatérhetünk a kívánt verzióra.

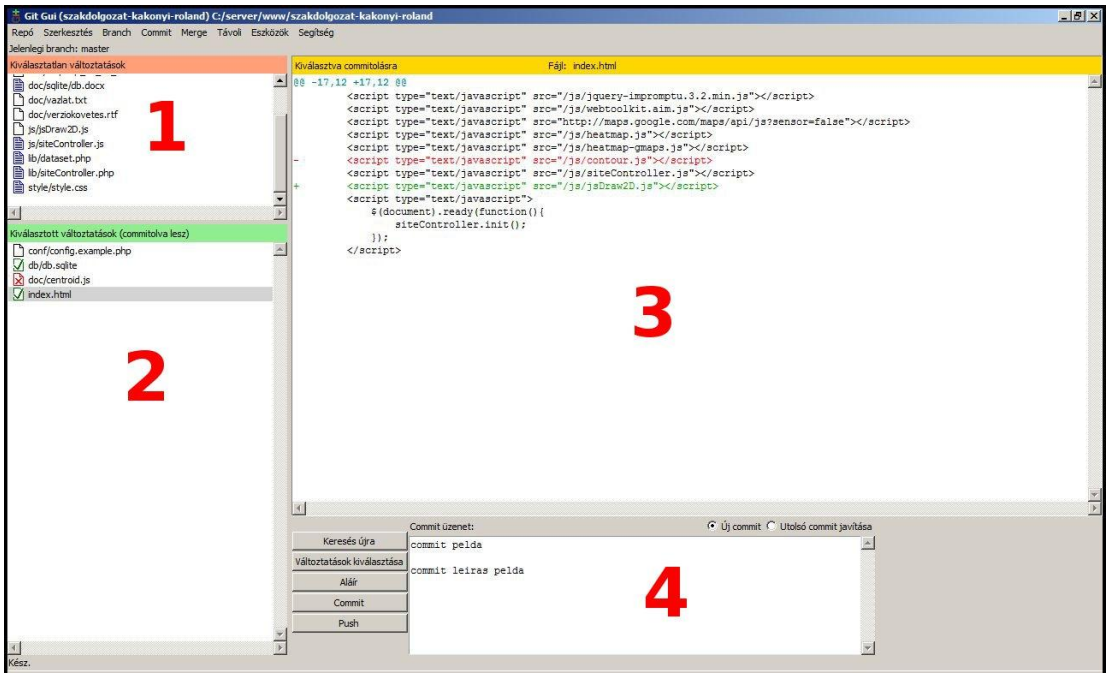


7. ábra Gitk működés közben

A Gitk működését a 16. ábra szemlélteti. A felület részei a következők:

1. Verzió fa: elkönyvelt változások listája fejlesztési áganként
2. Az adott változást beküldő felhasználó neve
3. Az adott változás beküldési ideje
4. A változás részletes adatai, fejlesztési ág, szülő commit, gyerek commit
5. Az adott változásban érintett fájlok tartalma
6. Az adott változásban érintett fájlok listája

2.4.8. Git Gui



8. ábra Git Gui működés közben

A Git Gui működését a 17. ábra szemlélteti. A felület részei a következők:

1. A legutóbbi commit óta változott fájlok listája
2. A jelenlegi commit-ba beválasztott fájlok listája
3. Az 1. vagy a 2. felületen kiválasztott fájlban történt változások
4. A jelenlegi commit-hoz tartozó üzenet

2.4.9. Verziókövető használata és előnyei a fejlesztés során

Ahhoz, hogy használni tudjuk a verziókövetőt, szükségünk van egy központi szerverre, ahol létrehozhatjuk a repository-t (tárolót). Három népszerű, ingyenes tároló megoldást ismerek, ezek közül választottam ki a fejlesztés során használandót. A *GitHUB* [19] és a *Bitbucket* [20] nagy népszerűségnek örvend, én mégis a *Google code* [21] mellett döntöttem egyszerű, gyors és biztonságos használata miatt. A *Google code* támogatja a Git-et és a Mercurial-t, tehát ebből a szempontból is megfelelő számomra.

Regisztráció után létre tudunk hozni egy repository-t, majd a saját gépünkre le kell *clone*-oznunk (vagyis egy lokális másolatot csinálunk). Innentől kezdve a Git Shell, a Gitk és a Git Gui segítségével végezhetjük a verziókövetést.

Ezek segítségével több lokális ágat (branch-et) tarthatunk fenn, amelyek teljesen függetlenek egymástól. Ezen fejlesztési vonalak létrehozása, egyesítése és törlése csupán másodpercekbe kerül.

Ez azt jelenti, hogy olyan dolgokat tehetünk, mint például:

- Létrehozhatunk egy új ágat, hogy kipróbálhassunk egy új ötletet, néhány változtatás (commit) után visszaválthatunk oda, ahonnan nyitottuk az új ágat, összefésülhetjük az azóta fejlesztett dolgokkal, majd visszaváltunk a kísérletező helyre, és beolvasztjuk.
- Fenntarthatunk egy ágat, amiben mindig csak az éles rendszerbe kerülő dolgok vannak (ezt szokás *master* ágnak hívni), egy másikat, amibe a tesztelésre kerülő munkákat olvasztjuk be, és több kisebbet a mindennapi feladatokra.
- Minden újabb feladatra, amelyen dolgozunk, új ágat hozhatunk létre, hogy gond nélkül váltogathassunk közöttük. Ezek után mindegyiket törölhetjük, amikor az adott dolog beolvasztásra kerül a főágban.
- Létrehozhatunk egy új ágat kísérletezésre; ha esetleg nem jött be a kísérletezés, egyszerűen kitörölhetjük, lemondva az abban végzett munkáról. Ezt az egészet senki más nem látja (akkor sem, ha közben más ágakat feltöltöttünk).
- Amikor egy távoli tárolóba (repositroy-ba) feltöltjük a változásokat, akkor nem kell minden ágat továbbítani, elég csak azt, amin épp dolgoztunk.

A projekteken belül általában vannak olyan fájlok, amiket nem szükséges követni. Ilyenek például a lokális konfigurációs fájlok, amik minden fejlesztőnél mások (pl. adatbázis-kapcsolat, elérési utak, stb.), vagy az ideiglenes létrehozott fájlok. Egy *.gitignore* nevű fájlban beállíthatjuk, hogy ezek a fájlok ne legyenek követve, így mielőtt commit-olnánk, nem is látszódnak, hogy ezek változtak.
Az én szakdolgozatom esetében ilyen nem követett állomány a *config.php*, ami az adatbázis-elérés konfigurációja és az elérési utak beállításait tartalmazza. Helyette egy *config.example* fájl van követve, amiben mindig jelzem a változásokat.

A szakdolgozatomat tartalmazó tároló megtalálható a http://code.google.com/p/szakdolgozat-kakonyi-roland/ címen.

3. Az alkalmazás tervezése

3.1. A tervezés és a kódolás során alkalmazott módszertanok

Az alkalmazás létrehozásának legfontosabb eleme a tervezés, itt a specifikációban megfogalmazott elvárásoknak megfelelő alkalmazás elméleti alapjait kell lefektetni. A gördülékeny fejlesztéshez és hatékony működéshez elengedhetetlenek a tervezés és a kódolás során alkalmazandó különböző programtervezési és programozási módszertanok ismerete.

3.1.1. Az Objektum Orientált Programozás alapjai

Az objektum-orientált programozás[22] (röviden OOP) a természetes gondolkodást, cselekvést közelítő programozási mód, amely a programozási nyelvek tervezésének természetes fejlődése következtében alakult ki. Az így létrejött nyelv sokkal strukturáltabb, sokkal modulárisabb és absztraktabb, mint egy hagyományos nyelv. Egy OOP nyelvet három fontos dolog jellemez. Ezek a következők:

Az *egységbezárás* (*encapsulation*) azt takarja, hogy az adatstruktúrákat és az adott struktúrájú adatokat kezelő függvényeket kombináljuk; azokat egy egységként kezeljük, és elzárjuk őket a külvilág elől. Az így kapott egységeket *objektumoknak* nevezzük. Az objektumoknak megfelelő tárolási egységek típusát a PHP-ban *osztály*nak (*class*) nevezzük.

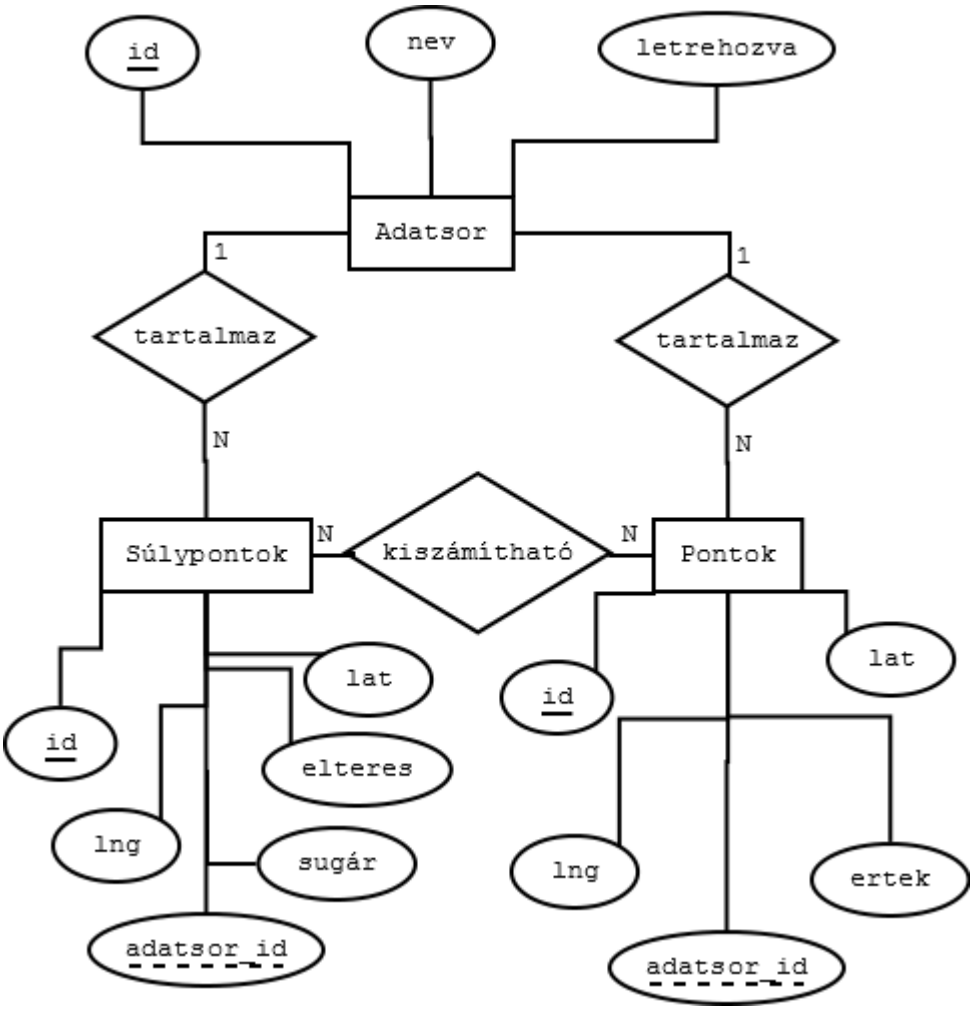
Az *öröklés* (*inheritance*) azt jelenti, hogy adott, meglévő osztályokból levezetett újabb osztályok öröklik a definiálásukhoz használt alaposztályok már létező adatstruktúráit és függvényeit. Ugyanakkor újabb tulajdonságokat is definiálhatnak, vagy régieket újraértelmezhetnek. Így egy osztályhierarchiához jutunk.

A *többrétűség* (*polymorphism*) alatt azt értjük, hogy egy adott tevékenység (metódus) azonosítója közös lehet egy adott osztályhierarchián belül, ugyanakkor a hierarchia minden egyes osztályában a tevékenységet végrehajtó függvény megvalósítása az adott osztályra nézve specifikus lehet. Az ún. *virtuális függvények* lehetővé teszik, hogy egy adott metódus konkrét végrehajtási módja csak a program futása során derüljön ki.

Ezek a tulajdonságok együtt azt eredményezik, hogy programkódjaink sokkal strukturáltabbá, könnyebben bővíthetővé, könnyebben karbantarthatóvá válnak, mintha hagyományos, nem OOP technikával írnánk őket.

3.1.2. E-K diagram

Először az E-K, egyedkapcsolati diagram megtervezése a legfontosabb (10. ábra). Ennek segítségével már könnyebben átlátható az egész adatbázis, amire az alkalmazás ráépül.

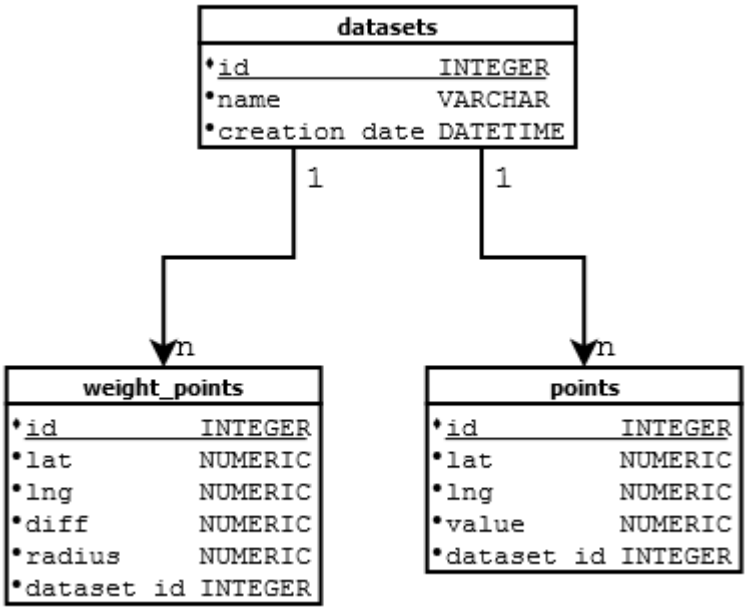


9. ábra E-K diagram

3.2. Az adatbázis szerkezete

Az adatbázis szerkezete lehetővé teszi több adatsor pontjainak és azok súlypontjainak tárolását és visszakeresését.

3.2.1. Adattáblák fizikai terve



10. ábra Adattáblák fizikai terve

3.2.2. Adattáblák bemutatása

A „datasets” tábla tárolja az adatsorok csoportosításához és azonosításához szükséges adatokat. Az azonosításhoz az adatsor mentésekor megadott nevet a „name” mezőben valamint a mentés idejét a „creation_date” mezőben tárolja.

Mezőnév	id	name	creation_date
Típus	integer	varchar	datetime
Alapértelmezett érték	nincs	nincs	datetime()
Lehet null	nem	nem	nem
Index	elsődleges kulcs	nincs	csökkenő
Auto increment	igen	nem	nem

11. táblázat „datasets” tábla szerkezete

A „points” tábla az adatsorokhoz tartozó pontok különböző tulajdonságait tárolja. A szélességet a „lat”, a hosszúságot az „lng”, a ponthoz mért sugárzási értéket pedig a „value” mező tartalmazza. Ezeken kívül az adott ponthoz tartozó adatsor azonosítója szerepel a táblában a „dataset_id” mezőben.

Mezőnév	id	lat	lng	value	dataset_id
Típus	integer	numeric	numeric	numeric	integer
Alapértelmezett érték	nincs	nincs	nincs	nincs	nincs
Lehet null	nem	nem	nem	nem	nem
Index	elsődleges kulcs	nincs	nincs	nincs	nincs
Auto increment	igen	nem	nem	nem	nem

12. táblázat „points” tábla szerkezete

Mezőnév	id	lat	lng	diff	radius	dataset_id
Típus	integer	numeric	numeric	numeric	numeric	integer
Alapértelmezett érték	nincs	nincs	nincs	nincs	nincs	nincs
Lehet null	nem	nem	nem	nem	nem	nem
Index	elsődleges kulcs	nincs	nincs	nincs	nincs	nincs
Auto increment	igen	nem	nem	nem	nem	nem

13. táblázat „weight_points” tábla szerkezete

3.2.3. Kapcsolatok az adatbázis táblái között

Az adatbázis táblái – egyedei közötti viszonyt jelölik a kapcsolatok, ezzel kiegészülve az E-K diagram, szemlélteti az adatbázis teljes struktúráját.

Adatsorok-pontok kapcsolat: Az „adatsorok” tábla minden egyes rekordja egy a többhöz kapcsolatban áll a „pontok” tábla bejegyzéseivel. Ez a kapcsolat biztosítja, hogy az egy adathalmazhoz tartozó pontokat be tudjuk azonosítani és csoportosítani. A kapcsolatot az „adatsorok” tábla „id” mezője és a „pontok” tábla „dataset_id” mezőinek egyezése határozza meg.

Adatsorok-súlypontok kapcsolat: Az „adatsorok” tábla rekordjai egy a többhöz kapcsolatban áll a „súlypontok” tábla bejegyzéseivel is. Ez a kapcsolat biztosítja, hogy az egy adathalmazhoz tartozó pontokból kiszámított súlypontokat be tudjuk azonosítani és csoportosítani. A kapcsolatot az „adatsorok” tábla „id” mezője és a „súlypontok” tábla „dataset_id” mezőinek egyezése határozza meg.

Pontok-súlypontok kapcsolat: A „súlypontok” tábla rekordjai egy a többhöz kapcsolatban állnak a „pontok” tábla bejegyzéseivel. Ez a kapcsolat biztosítja, hogy az egy adathalmazhoz tartozó pontokat és az azok feldolgozásából kiszámított súlypontot össze tudjuk rendelni. A kapcsolatot az „pontok” tábla „dataset_id” mezője és a „súlypontok” tábla „dataset_id” mezőinek egyezése, tehát az azonos adatsorhoz való kapcsolat jellemzi. Ez az adatszerkezet képes lenne tárolni több a többhöz kapcsolatban is a pontok-súlypontok kapcsolatait.

3.3. Kapcsolat az alkalmazás és az adatbázis között

Meg kell teremtenünk az alkalmazás és az adatbázis közötti kapcsolatot, erre a PHP, az egységes adatbázis kapcsolatok kezelésére létrehozott kiterjesztését, a PHP PDO kiterjesztés lehetőségeit használtam ki.

3.3.1. Mi az a PDO?

A PHP Data Objects[23] egy PHP 5 kiterjesztés neve, mely C kód formájában oldja meg az adatbázis absztrakció problémáját, olyan szolgáltatásokat nyújtva, mint a tranzakciók kezelése, előkészített SQL[10] parancsok futtatása, paraméterezése, automatikus típus szerinti *escapelés* (az SQL Injection[24] támadások ellen) és így tovább. Ez az egységesített interfész lehetővé teszi például, hogy rendszerünket SQLite alapokról MySQL[25] alapokra vigyük át egyetlen sor átírásával.
A PDO nem más, mint egy objektum az adatbázis kapcsolatok, lekérdezések kényelmes, hatékony, átlátható kezelésére.

3.3.2. Használata, legfontosabb metódusai

Használata egyszerű, gyorsan tanulható, a kódot újrahasznosíthatóvá, hordozhatóvá teszi [26].

Kapcsolat inicializálása:

```
$db = new PDO('mysql:host=localhost;dbname=database', 'user', 'password');
```

A kapcsolat létrehozásakor példányosított objektum metódusain keresztül érjük el az adatbázisunkat.

A legfontosabb metódusok:

Egyszerű lekérdezés:

A *query* metódussal egyszerűen futtathatunk lekérdezéseket.

```
$db->query(" SELECT * FROM `table` ");
```

Tranzakciók:

A tranzakciók lényege, hogy egy blokkban vagy mindegyik SQL utasítás végrehajtódik, vagy pedig egyik sem. A legjobban ezt egy banki átutalásos példával lehet szemléltetni, ahol az egyik lekérdezés leveszi "A" ügyfél számlájáról a pénzt, a második pedig rárakja "B" számlájára. Ez esetben azt szeretnénk, hogy ha nem sikerül az első lépés, akkor a második se hajtódjon végre. Ezt a következő példában mutatom be:

Példakód:

```
try {
        $db = new PDO('mysql:host=localhost;dbname=database', 'user', 'password');
        $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        $db->beginTransaction();
        $db->exec(" UPDATE `accounts` SET `account_value`=`account_value`-100 WHERE
`account_id`='1' ");
        $db->exec(" UPDATE `accounts` SET `account_value`=`account_value`+100 WHERE
`account_id`='2' ");
        $db->commit();
} catch (PDOException $e) {
        $db->rollBack();
}
```

A példában használt metódusok:

- *setAttribute*: Az adatbázis kezelő objektum tulajdonságait tudjuk vele beállítani különböző, a PDO objektumban tárolt konstansokkal. Ez esetben az lett beállítva, hogy - az alapértelmezettől eltérően - kivételt dobjon hiba esetén.
- *beginTransaction*: Jelzi, a tranzakció kezdetét.
- *commit*: A tranzakció lefuttatása.
- *rollBack*: A tranzakció visszavonása.

Előkészített lekérdezés:

Ennek előnye akkor mutatkozik, amikor egy lekérdezést többször kell futtatnunk különböző paraméterekkel. Ilyenkor a *prepare* metódussal előkészítjük a lekérdezést, a paraméterek helyére egy-egy tetszőleges nevet írva, majd az *execute* metódusnak megadjuk a paraméter név-érték párokat asszociatív tömbként.

Példakód:

```
$pre = $db->prepare(' INSERT INTO "attrs" ("id", "name") VALUES (:id, :name) ');
$pre->execute(array(':id' => '1', ':name' => '12'));
$pre->execute(array(':id' => '2', ':name' => '43'));
```

Kapcsolat lezárása:

```
$db = NULL;
```

3.3.3. Miért választottam?

A választásomat a PDO objektum orientáltsága, ezáltal gyors, könnyed kezelése, valamint hasonló képességű konkurens megoldás hiánya indokolja. Működési feltétele csupán a PHP PDO kiterjesztésének és a kiválasztott adatbázis motort támogató PHP kiterjesztés, esetemben SQLite támogatás, a php_sqlite kiterjesztés telepítése és engedélyezése.

4. Az alkalmazás megoldásainak bemutatása

4.1. Mikrokontrolleres adatgyűjtőből származó adathalmaz tárolása

Az adatok feltöltéséhez az adatsor nevét és a feltöltendő adatokat tartalmazó fájlt kell megadnunk. A betölteni kívánt adatokat az alkalmazás CSV[27] formátumból képes beolvasni és eltárolni. A CSV formátum jellegzetessége, hogy az összetartozó mezőket vesszővel (,), a sorokat sortöréssel választva tárolja. Ehhez igazodva dolgozza át az alkalmazás a beolvasott sorokat az adatbázisba való beszúráshoz. A feltöltött fájl sorait a PHP beépített *fgetcsv*[28] függvényével olvassuk ki. A feldolgozandó fájlt csak ideiglenesen tároljuk, a tartalma a későbbiekben az adatbázisból visszanyerhető.

Az adatsor nevének beszúrása után a pontok adatait (feltöltött fájl sorait) egyesével átalakítva szúrja be az adatbázisba az adatsor azonosítójával együtt - ügyelve a mezők szövegből lebegőpontos értékké való átalakítására. Beszúrás előtt ellenőrizzük az összes érték meglétét (szélesség, hosszúság, érték), ezek nélkül a pont adatai nem értelmezhetőek.

4.1.1. Adatbázis réteg

Az adatbázis és az alkalmazás közötti kapcsolatot az alkalmazás adatbázis rétege teremti meg. Erre a célra egy külön osztályt hoztam létre, a *Db* osztályt.

Ez az osztály szolgál arra, hogy az egy lekéréssel történő alkalmazás műveletek ugyanazon az adatbázis kapcsolaton keresztül képesek legyenek kommunikálni az adatbázissal, így teljesítményben és az adatbázis elérésének gyorsításában előrelépést jelent.

Itt azt használjuk ki, hogy a PDO kapcsolat az alkalmazáson belül addig él amíg a PDO objektumot tartalmazó változó értéke érvényes.

Ezt kombinálva PHP osztályok statikus változóinak élettartamával – az osztály beállított statikus változójának értéke megmarad, amíg a PHP script fut – egy bárhonnan elérhető, adatbázis kapcsolatot nyújtó statikus osztály metódust írhatunk az alkalmazásunkhoz.

A fentieket megvalósító PHP osztály kódja a következőképpen néz ki:

```
class Db {
        protected static $_oDb;
        public static function getInstance() {
                if (!self::$_oDb instanceof PDO) {
                global $CONF;
                try {
                        self::$_oDb = new PDO("sqlite:" . $CONF['db_settings']['path'] .
                $CONF['db_settings']['filename']);
                } catch (PDOException $e) {
                        die();
                }
        }
        return self::$_oDb;
        }
}
```

A fenti kódban láthatjuk, hogyan is működik a korábban vázolt adatbázis kapcsolati objektum előállítás és lekérés.

A *Db::getInstance()* metódus a meghívás után ellenőrzi, hogy az osztály *$_oDb* változója egy PDO objektum tárol-e, ha igen akkor visszaadja ezt az objektumot, ha nem akkor létrehozza a kívánt objektumot a *conf/config.php* fájlban beállított értékek alapján, eltárolja az *$_oDb* osztályváltozóban majd a korábbihoz hasonlóan visszaadja azt. Ha a kapcsolat létrehozása során hiba lép fel, a PHP script futását leállítjuk.

4.1.2. Adatok mentése az alkalmazásba

14. ábra Adatok feltöltése réteg

Adatsor beszúrása:

Az adatsort azonosító adatokat mentjük le először, mivel erre hivatkozva tudjuk eltárolni az összetartozó pontokat. A „dataset" táblába való beszúrás végrehajtása után a létrehozott adatsor azonosítóját (*id*-ját) lekérjük a PDO objektum *lastinsertId()* metódusával, majd eltároljuk az osztály *$iId* változójában, a pontok adatainak beszúrásánál elengedhetetlen lesz ez az adat.

A következő kódrészlet a *dataset* osztály *import()* metódus egy részlete.

```
try {
        $oSth = Db::getInstance()->prepare("INSERT INTO datasets (name,creation_date) VALUES
(:name,:creation_date)");
        $oSth->bindParam(':name', $this->sName);
        $oSth->execute();
        $this->iId = Db::getInstance()->lastInsertId();
}
catch (PDOException $e) {
        die();
}
```

Adatsorhoz tartozó pontok beszúrása:

A pontokat egyesével szúrjuk be a „points” táblába, amint beolvastunk egy sort a pont adataival a feltöltött fájlból.

Az alábbi részlet a *dataset* osztály *_savePoints()* metódusából származik.

```
if (($handle = fopen($sSource_file, "r")) !== FALSE) {
	try {
		$sQuery = "INSERT INTO points (lat,lng,count,dataset_id)
		VALUES (:lat,:lng,:count,:dataset_id);";
		$oSth = db::getInstance()->prepare($sQuery);
		$oSth->bindParam(':dataset_id', $this->iId);
		$oSth->bindParam(':lat', $fLat);
		$oSth->bindParam(':lng', $fLng);
		$oSth->bindParam(':count', $fCount);
	}
	catch (PDOException $oE) {
		die();
	}
	while (($aRow = fgetcsv($handle, $iMax_line_length, $sDelimiter, $sEnclosure)) !== FALSE) {
		try {
			$fLat = floatval($aRow[0]);
			$fLng = floatval($aRow[1]);
			$fCount = floatval($aRow[2]);
			if (isset($fLat) && isset($fLng) && isset($fCount)) {
				$oSth->execute();
			}
		}
		catch (PDOException $oE) {
			@fclose($handle);
			die();
		}
	}
	fclose($handle);
}
```

Ahogy azt az előző két kódrészletből láthatjuk, az adatbázisba való beszúrás során kihasználtam a 4.1.1 fejezetben bemutatott adatbázis objektum létrehozását és lekérését valamint a PHP PDO objektumainak képességeit, mint az előkészített lekérdezés és a hibakezelést kivételekkel.

4.1.3. Adatok betöltése az alkalmazásból

Mentett adatsorok listázása az adatbázisból:

Ahhoz hogy a mentett adatokat be tudjuk tölteni, a felhasználónak lehetőséget kell adnunk arra, hogy egy listából kiválaszthassa a kívánt adatsort. A kiválasztáshoz megjelenítjük a listában a mentett adatsor nevét, mentési idejét és a hozzá tartozó pontok számát.

Az alábbi kódrészlet a *dataset* osztály *getDatasetList()* metódus része, egyetlen paramétere pedig a lista megtekintendő oldalszáma, aminek alapértéke 1.

A listázáshoz az adatokat az alábbi kódrészlet szolgáltatja:

```
try {
	$oSth = Db::getInstance()->query('SELECT count(*) FROM datasets ORDER BY creation_date DESC');
	$oSth->execute();
	$iCount = intval($oSth->fetchColumn(0));
	if ($iCount) {
		$iOffset = intval(($iPage - 1) * 5);
		$oSth = Db::getInstance()->query('SELECT d.id as id,d.name as name,d.creation_date as
	creation_date,count(p.id) as count FROM datasets d JOIN points p ON d.id=p.dataset_id GROUP BY d.id
	ORDER BY creation_date DESC LIMIT 5 OFFSET ' . $iOffset);
		$oSth->execute();
		$aDatasets = $oSth->fetchAll(PDO::FETCH_ASSOC);
	}
	else {
		$aDatasets = array();
		$iOffset = 0;
	}
}
catch (PDOException $oE) {
	die();
}
return array('success' => true, 'datasets' => $aDatasets, 'count' => $iCount, 'offset' => $iOffset);
```

15. ábra Mentett adatsorok listája

Adatsorhoz tartozó pontok betöltése a megjelenítéshez:

A fent említett listában, a betölteni kívánt adatsor sorában a „betöltés” gombra kattintva érhetjük el az adott adatsor pontjainak megjelenítését. A betöltés során megerősíthetjük vagy megadhatjuk újra (módosíthatjuk), a súlypont meghatározáshoz szükséges „maximálisan megengedett érték eltérés” paramétert. Amennyiben új értéket adunk meg a feldolgozás megtörténik, azonban ha nem változtatjuk meg a paramétert, akkor az adatbázisból betöltjük a már kiszámított súlypont koordinátáit. A feldolgozás menetét a 4.4. pontban részletesen ismertetem.

Első lépésként lekérdezzük az adatsorhoz tartozó adatokat, valamint a mentett súlypont adatait:

```
try {
        $oSth = Db::getInstance()->prepare("SELECT d.id,d.name,strftime('%Y. %m. %d.
%H:%M:%S',d.creation_date) as creation_date, w.diff as param, w.lat, w.lng FROM datasets d LEFT JOIN
weight_points w ON d.id=w.dataset_id WHERE d.id=:id");
        $oSth->bindParam(':id', $this->iId);
        $oSth->execute();
        $aDataset = $oSth->fetchAll(PDO::FETCH_ASSOC);
        $aDataset = $aDataset[0];
```

Majd lekérjük a pontok halmazát:

```
$oSth = Db::getInstance()->prepare('SELECT p.lat,p.lng,p.value FROM datasets d JOIN points p ON
p.dataset_id=d.id WHERE d.id=:id ORDER BY p.id');
$oSth->bindParam(':id', $this->iId);
$oSth->execute();
$aPoints = $oSth->fetchAll(PDO::FETCH_ASSOC);
if (!count($aPoints)) {
        die();
}
```

Ha találhatóak pontok, akkor átalakítjuk az egyes értékeit a megfelelő típusúra, eközben végzünk egy maximumkeresést, ez az érték a megjelenítéshez szükséges. Erről bővebben a 4.3. pontban.

```
$iMax = 0;
foreach ($aPoints as $aRow) {
        $iValue = floatval($aRow['value']);
        if ($iValue > $iMax) {
                $iMax = $iValue;
        }
        $aPoints[] = array(
                'lat' => floatval($aRow['lat']),
                'lng' => floatval($aRow['lng']),
                'count' => $iValue
        );
}
```

Szükségünk van a pontok közül a dél-nyugati irányban a legtávolabb helyezkedő, valamint az észak-keleti irányban a legtávolabb helyezkedőre. Ezek is szintén a megjelenítéshez szükségesek. A megjelenítésről szóló, 4.3. pontban taglalom ezek fontosságát.

Az adatsor pontjait és a határpontokat lekérdező kódrészlet:

```
                $oSth = Db::getInstance()->prepare("SELECT p.lat,p.lng FROM datasets d JOIN points p ON
p.dataset_id=d.id WHERE d.id=:id ORDER BY p.lat+p.lng ASC LIMIT 1");
                $oSth->bindParam(':id', $this->iId);
                $oSth->execute();
                $aBoundSw = $oSth->fetchAll(PDO::FETCH_ASSOC);
                $oSth = Db::getInstance()->prepare("SELECT p.lat,p.lng FROM datasets d JOIN points p ON
p.dataset_id=d.id WHERE d.id=:id ORDER BY p.lat+p.lng DESC LIMIT 1");
                $oSth->bindParam(':id', $this->iId);
                $oSth->execute();
                $aBoundNe = $oSth->fetchAll(PDO::FETCH_ASSOC);
        }
}
catch (PDOException $oE) {
        die();
}
return array('success' => true, 'dataset' => $aDataset, 'points' => $aPoints, 'max' => $iMax, 'sw' => $aBoundSw[0],
'ne' => $aBoundNe[0]);
```

4.2. Adatgyűjtő paraméterezése

Az alkalmazás képes az adatgyűjtő konfigurálásához előállítani egy paraméterfájlt. A paraméterfájl CSV formátumban kerül kigyártásra a megadott adatok, paraméterek alapján. A funkció a felületen a „Paraméterfájl generálás” menüpontban érhető el. A gomb megnyomására a következő felület jelenik meg:

16. ábra Paraméter generálás felület

A „Generál” gomb megnyomása után a paramétereket CSV formátumba a következő kód alakítja át és írja egy fájlba.

```
$sCsvFileName = $CONF['var_path'] . "parameterek.csv";
@unlink($sCsvFileName);
$handle = fopen($sCsvFileName, 'w');
fputcsv($handle, array_keys($aValues));
fputcsv($handle, array_values($aValues));
fclose($handle);

$sUrl = $CONF['ajax_url'] . "?method=getParamFile";
die(json_encode(array('success' => true, 'url' => $sUrl)));
```

A generált fájl szerkezete a következő:

- első sor: feltöltött paraméterek nevei
- második sor: feltöltött paraméterek értékei

A paraméterek nevei és értékei oszloponként összetartoznak.

paraméter1 név	paraméter2 név	paraméter3 név
paraméter1 érték	paraméter2 érték	paraméter3 érték

17. táblázat Sablon a paraméterfájlra

A kigenerált fájlt a legyártott URL-en érjük el és automatikusan megnyitjuk amint elkészült a paraméterfájl.

A legyártott URL-en az alábbi kód szolgálja ki a kigyártott tartalmat:

```
$sCsvFileName = $CONF['var_path'] . "parameterek.csv";
header('Content-type: text/csv');
header("Content-Disposition: attachment; filename=parameterek.csv");
header("Pragma: no-cache");
header("Expires: 0");
header('Content-length: ' . filesize($sCsvFileName));
readfile($sCsvFileName);
exit();
```

A fájlt a CSV formátumnak megfelelő „text/csv” tartalom típussal szolgáljuk ki így biztosítva, hogy a felhasználó a számítógépén a megfelelő alapértelmezett programmal tudja megnyitni. Valamint biztosítjuk, hogy a felhasználó böngészője ne mentse gyorsítótárba a fájlt, ezáltal mindig a legfrissebb tartalom kerül kiszolgálásra.

4.3. Mért koordinátákhoz tartozó sugárzási érték pontonkénti megjelenítése

A térkép megjelenítéséhez a Google Maps API V3[29] alkalmazást használtam, a heatmap.js[30] nevű hő térkép megjelenítő javascript programkönyvtárral kiegészítve.

4.3.1. A megjelenítésről általánosan

Az alkalmazásnak a megjelenítéshez aktív internetkapcsolatra van szüksége a Google Maps térkép miatt. Ez az alkalmazás egyetlen külső függősége. Ezáltal az API komoly változása esetén az alkalmazás megjelenítési funkciója működésképtelenné válhat.

4.3.2. Google Maps API használata

A Google Maps JavaScript API V3 egy online térképalkalmazás interfész, ami a rengeteg térképkezelési és megjelenítési eszközt biztosít. A szakdolgozatomban a jelölő, kör objektum és nézőpont beállító funkciókat használtam.

A többféle nézőpont beállító funkció közül a határokkal beállított nézőpont meghatározást használtam. Ebben az esetben a térképnek dél-nyugati valamint az észak-keleti pontjait adjuk meg – amelyek nem egyezhetnek meg – és így állítja be a nézőpontot.

Példakód a nézőpont beállítására:

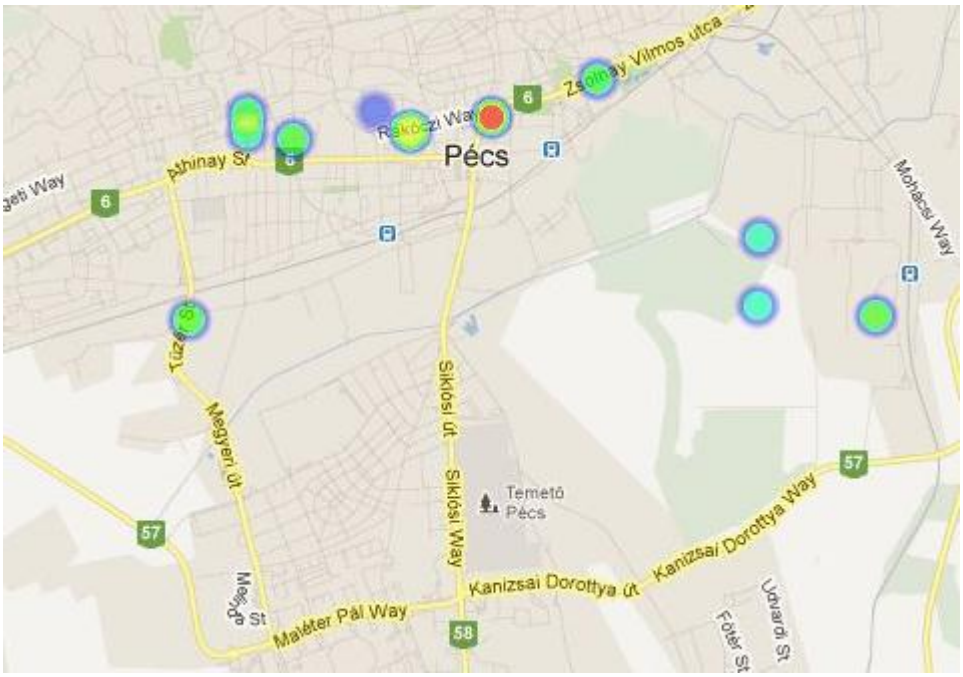
```
siteController.heatmapOverlay.map.fitBounds(
        new google.maps.LatLngBounds(
                new google.maps.LatLng(sw.lat, sw.lng),new google.maps.LatLng(ne.lat, ne.lng)
        )
);
```

Ezt a nézőpont beállítási módot akkor használjuk, ha a felhasználó nem kéri a sugárzás forrásának meghatározását, ellenkező esetben a 4.5.3 fejezetben leírtak szerint jár el az alkalmazás.

4.3.3. Heatmap.js használata

A heatmap.js egy HTML5 alapú hő térkép megjelenítésére alkalmas JavaScript programkönyvtár. A hő térképet a Google Maps térképet tároló *div* elem fölé pozícionált *canvas* elemre rajzol.

A heatmap.js programkönyvtárnak megadva a pontok szélességi, hosszúsági és mért érték adatait, az alkalmazás megjeleníti a térképen a mért értéknek megfelelő színnel. A színt relatívan választja ki a megjelenített adatsor összes pontjának mért érték viszonyai alapján. A halmaz legnagyobb értéke lesz piros színű, a középérték zöld és a legkisebb érték kék. A köztes értékeket ezen színek árnyalataival jeleníti meg.



18. ábra Pontok megjelenítése példa

A pontok megjelenítési szín skálája testre szabható a hő térkép objektum létrehozásakor a különböző érték lépcsők megadásával a konfigurációs objektum „gradient” mezőjében. Használt maximális érték-érték arány és szín párok a hő térkép objektum konfigurációjában:

```
gradient:{
        0.45: "rgb(0,0,255)",
        0.50: "rgb(0,128,255)",
        0.55: "rgb(0,255,255)",
        0.60: "rgb(0,255,128)",
        0.65: "rgb(0,255,196)",
        0.70: "rgb(0,255,0)",
        0.75: "rgb(64,255,0)",
        0.85: "rgb(128,255,0)",
        0.95: "rgb(255,255,0)",
        1.0: "rgb(255,0,0)"
}
```

4.4. Kvázi ekvivalens pontokra görbe illesztés

Adatsor feltöltés után vagy adatsor betöltésekor lehetőségünk van becsült sugárzás forrás meghatározására, ehhez a 19. ábrán látható felületen kell megadnunk lebegőpontos értékként a *maximálisan megengedett érték eltérés*t. Ez az érték jelentősen meghatározza a súlypont számításának eredményét.

19. ábra paraméter megadás súlypont számításához

Az adatsorhoz tartozó pontokat a *$aPoints* tömb tartalmazza. Ezen a ponthalmazon iterálunk végig és csoportosítjuk össze a pontokat a *$aEqualPoints* tömbbe. Az összerendelés akkor jön létre, ha a két aktuálisan vizsgált ponthoz a mért értékek különbségének abszolút értéke kisebb, mint a megadott *$Param* paraméter.

A következő kódrészlet valósítja meg a pontok érték szerinti összerendelését:

```
$aEqualPoints = array();
for ($i = 0; $i < count($aPoints); $i++) {
        for ($k = $i; $k < count($aPoints); $k++) {
                if ($i == $k) {
                        $aEqualPoints[$aPoints[$i]['id']][] = $aPoints[$k];
                        continue;
                }
                $diff = abs(floatval($aPoints[$k]['value']) - floatval($aPoints[$i]['value']));
                if ($diff <= $Param) {
                        $aEqualPoints[$aPoints[$i]['id']][] = $aPoints[$k];
                        $aEqualPoints[$aPoints[$k]['id']][] = $aPoints[$i];
                }
        }
}
```

Ha összetartozó pontokat találunk, akkor mindkét ponthoz létrehozzuk a kapcsolatot, így közel felére csökkenthető az iterációk száma és ez által gyorsabb a feldolgozás.

A pontokat saját magukkal sem hasonlítjuk össze, csupán hozzáadjuk a kapcsolatait tartalmazó tömbbe.

Az alábbi táblázatok szemléltetik a különbséget a fent megvalósított megoldás csökkentett iteráció számú megoldás és a teljes iteráció között 5 elemű tömb bejárása esetén:

$i\$k	1	2	3	4	5
1	x	x	x	x	x
2		x	x	x	x
3			x	x	x
4				x	x
5					x

20. táblázat csökkentett számú iteráció által vizsgált esetek

$i\$k	1	2	3	4	5
1	x	x	x	x	x
2	x	x	x	x	x
3	x	x	x	x	x
4	x	x	x	x	x
5	x	x	x	x	x

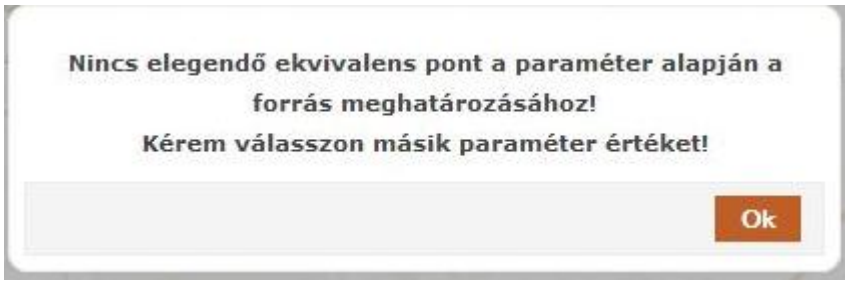
21. táblázat teljes iteráció által vizsgált esetek

A 20. és a 21. táblázatokban jól látható, hogy a csökkentett számú iteráció optimálisabb megoldás sebesség szempontjából mivel 5 elemű halmaz esetén 25 iteráció helyett, amely a teljes iteráció esetén történne, csupán 15 iterációra van szükség.

Következő lépésként meg kell keresni a legtöbb összetartozó értékkel rendelkező összecsoportosított halmazt.

```
$maxArrCount = 0;
foreach ($aEqualPoints as $i => $arr) {
        $count = count($arr);
        if ($count > $maxArrCount) {
                $maxArrCount = $count;
                $maxArrIdx = $i;
        }
}
```

Abban az esetben, ha nincs legalább három összetartozó értékünk, akkor nem tudjuk folytatni a súlypont keresését, mert a kör nem határozható meg az ismert adatokból.
Ilyenkor hibajelzéssel válaszol a program és megkéri a felhasználót új paraméter érték megadására. Ebben az esetben a 19. ábrán mutatott űrlapot kínáljuk fel a felhasználónak.



22. ábra Ekvivalens pontok keresés hibaüzenet

4.5. Az így megkapott görbék alapján súlypont (sugárzás forrása) meghatározása

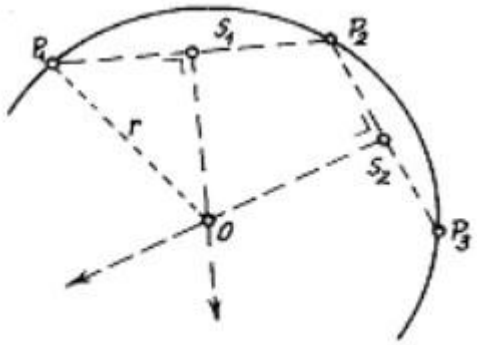
4.5.1. Kör középpontjának (sugárzás forrásának) keresése pontok alapján

A 4.4. fejezetben kinyert pontok alapján kell meghatároznunk a kör középpontját.

Egy kört definiál a körív 3 adott pontja, ennek kötelező feltétele a pontosan 3 pont megléte, valamint azok nem eshetnek egy egyenesre.

A pontok legyenek *P1, P2* és *P3*.

A *P1P2* és *P2P3* egyenesek oldalfelező merőlegeseinek közös metszéspontja meghatározza a kör középpontját *O*. Ezt reprezentálja a 23. ábra.



23. ábra Kör középpontjának szerkesztése 3 kerületi pontból

A következő kódrészlet valósítja meg a kör középpontjának meghatározását:

```
$tmp = $cx * $cx + $cy * $cy;
$bc = ($bx * $bx + $by * $by - $tmp) / 2;
$cd = ($tmp - $dx * $dx - $dy * $dy) / 2;

$det = ($bx - $cx) * ($cy - $dy) - ($cx - $dx) * ($by - $cy);
$det = 1 / $det;

$lat = ($bc * ($cy - $dy) - $cd * ($by - $cy)) * $det;
$lng = (($bx - $cx) * $cd - ($cx - $dx) * $bc) * $det;
```

Ahol a P1, P2 és P3 pontok koordinátáit a P1=($bx,$by), P2=($cx,$cy) és a P3=($dx,$dy) változók reprezentálják.

4.5.2. A kör középpontjától legtávolabb elhelyezkedő mért koordináta

A kör középpontjától legtávolabb elhelyezkedő mért koordinátát az alábbi SQL lekérdezéssel nyerhetjük ki az adatbázisból:

```
SELECT lat,lng
FROM points
WHERE dataset_id={adatsor azonosito}
ORDER BY (lat-{középpont szélesség})*(lat-{középpont szélesség}) +
(lng-{középpont hosszúság})*(lng-{középpont hosszúság}) DESC
LIMIT 1
```

A fenti lekérdezés az adatsorhoz tartozó pontok szélességi és hosszúsági értékekei alapján kiszámolja a kiszámolt középponthoz mért távolságot, itt a viszonyítás miatt kell a távolság, pontos adatot nem szolgáltat a valódi távolságról.
A legtávolabbi pont lesz a lekérdezés eredményének egyetlen sora.

4.5.3. Megfelelő nézőponthoz szükséges adatok kiszámolása

A program előállítja a már kiszámított kör középpontja és az fent kiszámolt legtávolabbi pont koordinátái alapján a két pont közötti pontos távolságot a Haversine formula [31] segítségével, amelyre az általános képlet a következő:

$$\Delta\text{lat} = \text{lat2} - \text{lat1}$$

$$\Delta\text{long} = \text{long2} - \text{long1}$$

$$\text{a} = \sin^2\left(\frac{\Delta\text{lat}}{2}\right) + \cos(\text{lat1}) \cdot \cos(\text{lat2}) \cdot \sin^2\left(\frac{\Delta\text{long}}{2}\right)$$

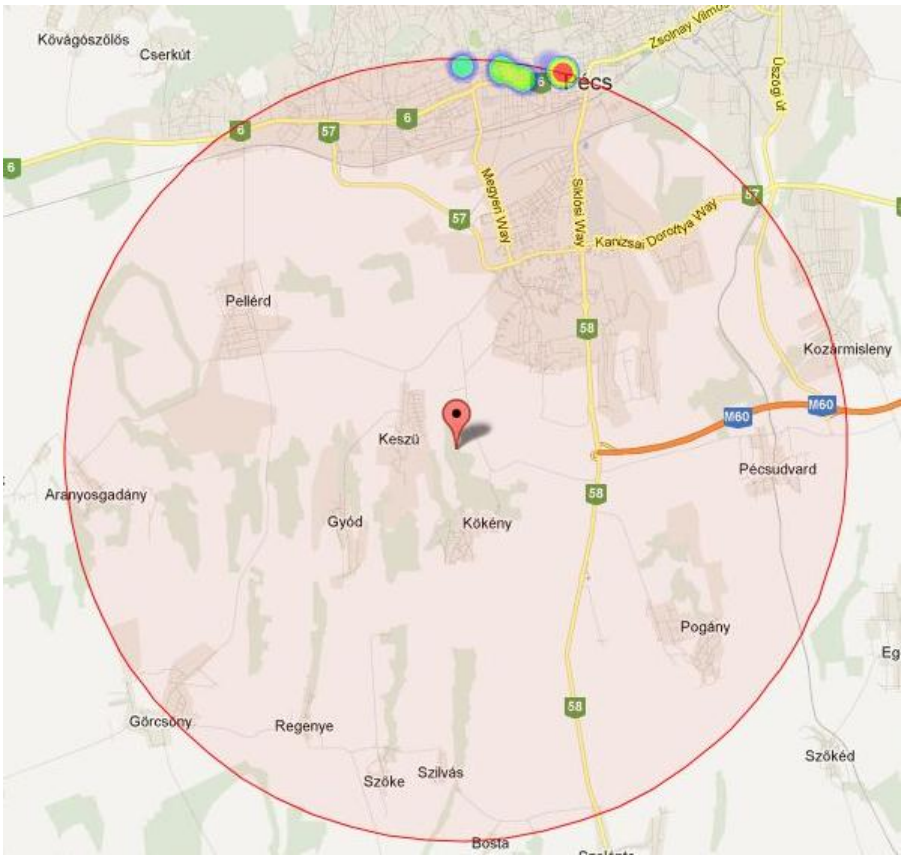
$$\text{c} = 2 \cdot \text{atan2}\left(\sqrt{\text{a}}, \sqrt{1-\text{a}}\right)$$

$$\text{d} = \text{R} \cdot \text{c}$$

24. egyenlet Haversine formula

Ahol R=6371, a föld sugara kilométerben értve.
Az így kapott *d* távolságot kilométerben kapjuk meg.

Erre a távolságra a térkép megfelelő nézőpontjának beállításához van szükség.
A kívánt nézőpontot egy láthatatlan kör segítségével állíthatjuk be. Ennek a körnek a középpontja a kiszámított sugárzás forrás koordinátája, sugara pedig a Haversine formulával kalkulált távolság. A nézőpontnak pedig a kör teljes területét mutatnia kell, így az összes megjelenítendő pontunk és a sugárzás forrása is látható lesz a térképen.



25. ábra Nézőpont beállító segédkör vizualizálása

A sugárzás forrásának jelzéséhez az alábbi kóddal egy jelölőt helyezünk a térképre:

```
siteController.center = new google.maps.Marker({
        position: new google.maps.LatLng(center.lat,center.lng),
        map: siteController.heatmapOverlay.map,
        title:'Becsült forrás helye',
        "clickable": true
});
```

A 25. ábrán látható nézőpontot és kör objektumot az alábbi kód segítségével állíthatjuk be:

```
siteController.circle=new google.maps.Circle({
        animation: google.maps.Animation.BOUNCE,
        center: siteController.center.getPosition(),
        clickable: true,
        fillColor: '#f00',
        fillOpacity: 0.05,
        map: siteController.heatmapOverlay.map,
        radius: ret.radius,
        strokeColor: '#f00',
        strokeWeight: 1,
        strokeOpacity: 1.0,
        visible:false
});
```

Nézőpont beállítása az alábbi paranccsal történik:

```
siteController.heatmapOverlay.map.fitBounds(siteController.circle.getBounds());
```

Ez a parancs biztosítja a térkép pozíciójának és nagyításának helyes beállítását.

5. Összefoglalás

A szakdolgozatom eredménye egy önálló alkalmazás, amely a feladatkiírásban szereplő elvárásoknak megfelel, követelményeit sikeresen teljesíti.

Képes a mért adatok tárolására és visszakeresésére, ezekhez átlátható felhasználói felületet nyújt.

A szakdolgozat készítésének idejében még el nem készült mikrokontrolleres adatgyűjtő prototípus paraméterezésére is lehetőséget nyújt az alkalmazás. Ezen funkció a prototípuson való további tesztelése szükséges a tökéletes kompatibilitás elérése miatt.

A ponthalmazok megjelenítését a térképen a pont koordinátái és érték tulajdonság alapján időjárási térképekhez hasonló hő térkép szerű vizualizációval.

A tárolt adatokat képes feldolgozni, kvázi ekvivalens mért értékkel rendelkező pontokat összerendezni. Az ekvivalencia megállapításához a felhasználónak meg kell adnia az maximálisan megengedett érték eltérést. Ez a paraméter lényegesen meghatározza a feldolgozás kimenetelét.

Az ekvivalensnek értékelt ponthalmaz alapján képes a sugárzás forrását megközelítő koordináta meghatározására és ennek jelölésére a térképen. A forrás koordinátáinak megállapításához a koordináta geometria módszereit használja.

A feladatkiíráson túl az alkalmazás képes dinamikusan irányítani a térkép nézőpontját a megjelenítendő ponthalmazok és adatok alapján.

<u>Továbbfejlesztési lehetőségek:</u>

- pontok megjelenítésének teljesítmény optimalizálása
- pontok megjelenítésében előforduló véletlen hibák javítása
- ekvivalens pontok keresése algoritmus pontosítása, bővítése

6. Mellékletek

Forráskód:

Az alkalmazás forráskódja elérhető a *https://code.google.com/p/szakdolgozat-kakonyi-roland/* címen.

Az alkalmazás működéséhez szükséges telepítendő szolgáltatások, szoftverek:

- Apache HTTP Server 2.2.17
- PHP 5.2.17

Az alkalmazás működéséhez szükséges PHP kiterjesztések listája:

- php_pdo
- php_pdo_sqlite
- php_sqlite

Apache virtualhost példafájl:

```
NameVirtualHost szakdolgozat.local:80

<VirtualHost szakdolgozat.local:80>
  ServerName szakdolgozat.local
  DocumentRoot c:/server/www/szakdolgozat-kakonyi-roland
  DirectoryIndex index.php index.html
  ServerAdmin rolandkakonyi@gmail.com
</VirtualHost>
```

Adatbázis szerkezetet létrehozó SQL parancsok:

```
CREATE TABLE "datasets" ("id" INTEGER PRIMARY KEY  NOT NULL ,"name" VARCHAR NOT NULL
,"creation_date" DATETIME DEFAULT (datetime()) );

CREATE TABLE "points" ("lat" NUMERIC NOT NULL ,"lng" NUMERIC NOT NULL ,"value" NUMERIC NOT NULL
,"dataset_id" INTEGER NOT NULL ,"id" INTEGER PRIMARY KEY  NOT NULL );

CREATE TABLE "weight_points" ("lat" NUMERIC NOT NULL ,"lng" NUMERIC NOT NULL ,"diff" NUMERIC NOT
NULL ,"dataset_id" INTEGER NOT NULL ,"id" INTEGER PRIMARY KEY  NOT NULL ,"radius" NUMERIC NOT
NULL );

CREATE INDEX "d_creation_date" ON "datasets" ("creation_date" DESC);
```

Példafájl tartalma teszt feltöltéshez:

Szélesség	Hosszúság	Mért érték
46.072549	18.213741	175.0457157
46.073849	18.224741	113.9459712
46.072149	18.217641	195.3022394
46.072649	18.227841	183.879793
46.061449	18.208741	196.4492915
46.072649	18.227741	198.3796747
46.073749	18.213741	182.7653555
46.062349	18.257541	165.1655411
46.073449	18.234741	280.1813188
46.061649	18.267741	198.3796747
46.075749	18.243741	182.7653555
46.066349	18.257641	165.1655411
46.074449	18.474741	180.1813188

A táblázat tartalma elérhető a forráskód mellett, az alkalmazás főkönyvtárában a *test* könyvtár *test.csv* fájljában.

7. Irodalomjegyzék

[1] HOW WEB SERVERS WORK - CLIENTS AND SERVERS [Online] http://computer.howstuffworks.com/web-server4.htm (látogatva: 2011. december)

[2] PHP [Online] http://php.net/ (látogatva: 2011. december)

[3] APACHE HTTP SZERVER [Online] http://httpd.apache.org/ (látogatva: 2011. december)

[4] HTML5 [Online] http://html5.org/ (látogatva: 2011. december)

[5] JQUERY [Online] http://jquery.com/ (látogatva: 2011. december)

[6] JQUERY IMPROMPTU [Online] http://trentrichardson.com/Impromptu/ (látogatva: 2011. december)

[7] NETBEANS [Online] http://netbeans.org/ (látogatva: 2011. december)

[8] GIT [Online] http://git-scm.com/ (látogatva: 2011. december)

[9] SQLITE [Online] http://www.sqlite.org/ (látogatva: 2011. december)

[10] ISO/IEC 9075:1992, DATABASE LANGUAGE SQL [Online] http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt (látogatva: 2011. december)

[11] PHP FUNCTION FOPEN [Online] http://php.net/manual/en/function.fopen.php (látogatva: 2011. december)

[12] ATOMIC CONSISTENT ISOLATED DURABLE [Online] http://c2.com/cgi/wiki?AtomicConsistentIsolatedDurable (látogatva: 2011. december)

[13] SQLITE-MANAGER FIREFOX ADD-ON [Online] http://code.google.com/p/sqlite-manager/ (látogatva: 2012. január)

[14] SQLITE-MANAGER FIREFOX ADD-ON [Online] https://addons.mozilla.org/hu/firefox/addon/sqlite-manager/ (látogatva: 2012. január)

[15] VERZIÓKÖVETÉS [Online] http://logout.hu/cikk/verziokovetes/a_verziokovetes_lenyege.html (látogatva: 2011. december)

[16] VERSION CONTROL WITH SUBVERSION [Online] http://svnbook.red-bean.com/ (látogatva: 2011. december)

[17] MERCURIAL [Online] http://mercurial.selenic.com/ (látogatva: 2011. december)

[18] BASH - GNU BOURNE-AGAIN SHELL [Online] http://linux.die.net/man/1/bash (látogatva: 2011. december)
[19] GITHUB [Online] https://github.com/ (látogatva: 2011. december)
[20] ATLASSIAN BITBUCKET [Online] https://bitbucket.org/
(látogatva: 2011. december)
[21] GOOGLE CODE [Online] http://code.google.com/ (látogatva: 2011. december)
[22] ANGSTER, ERZSÉBET: AZ OBJEKTUMORIENTÁLT TERVEZÉS ÉS PROGRAMOZÁS ALAPJAI (magánkiadás 1999; ISBN 9636508186)
[23] PHP DATA OBJECTS [Online] http://hu.php.net/pdo (látogatva: 2011. december)
[24] WEBOLDALAK BIZTONSÁGA 1: SQL INJECTION [Online]
http://pezia.hu/content/2009/03/08/weboldalak_biztons%C3%A1ga_1_sql_injection
(látogatva: 2011. december)
[25] MYSQL [Online] http://www.mysql.com/ (látogatva: 2011. december)
[26] INTRODUCTION TO PHP PDO [Online]
http://www.phpro.org/tutorials/Introduction-to-PHP-PDO.html
(látogatva: 2011. december)
[27] COMMON FORMAT AND MIME TYPE FOR COMMA-SEPARATED VALUES (CSV) FILES [Online] http://www.ietf.org/rfc/rfc4180.txt
(látogatva: 2011. december)
[28] PHP FUNCTION FGETCSV [Online]
http://php.net/manual/en/function.fgetcsv.php (látogatva: 2011. november)
[29] GOOGLE MAPS API V3 [Online]
https://developers.google.com/maps/documentation/javascript/reference
(látogatva: 2011. november)
[30] HEATMAP.JS BY PATRICK WIED [Online]
http://www.patrick-wied.at/static/heatmapjs/ (látogatva: 2011. november)
[31] U. S. CENSUS BUREAU GEOGRAPHIC INFORMATION SYSTEMS FAQ, WHAT IS THE BEST WAY TO CALCULATE THE DISTANCE BETWEEN 2 POINTS? [Online]
http://www.movable-type.co.uk/scripts/GIS-FAQ-5.1.html (látogatva: 2012. május)