Rendszerterv

1. Funkcionális terv

1.1. Feladat leírása:

A feladat egy GPS-képes eszközökön futó alkalmazás, illetve ennek szerver oldali párjának létrehozása. A program a szerveren tárolt adatbázis alapján küldene információt a felhasználóknak. Egy felhasználó egy GPS-t ismerő eszköz lehet, de ezen belül nincs megkötés, tehát elvileg lehet ez mobiltelefon, PDA, laptop, stb. A felhasználó elküldi GPS-en keresztül megszerzett pozícióját a szervernek, mely egy térképrészletet küld vissza a felhasználónak, természetesen a kapott adatoktól függően (például egy teljes Magyarország térképet feltételezve a felhasználónak egyszerre maximum egy kisebb városnyi térképre lehet szüksége, ennél nagyobb, áttekintőbb térkép csak ritkán szükséges). A rendszer alapvetően többfelhasználós, vagyis nem csak egy felhasználó kommunikál a szerverrel, hanem egyszerre akár több is. Így lehetséges olyan megoldás is, hogy a felhasználók csoportokba sorolódnak, és egy csoporton belüli felhasználók látják egymás pozícióját a kapott térképen. Lehetséges továbbá olyan statikus „felhasználók" jelenléte is, akik nem igazi felhasználói a rendszernek, csak egyetlenegyszer kerül be a pozíciójuk a rendszerbe, majd ezt később kívánságra bármely felhasználó láthatja (a térképi esetben ez például jelenthet éttermeket, mozikat, bármilyen fix helyzetű, a felhasználó számára fontos helyet).

1.2. Rendszer célja, motivációja:

A feladat az önálló labor VIII. szemeszterének alapját képezi. Ennek megfelelően nem azok a követelmények vonatkoznak rá, mint egy a való életnek készülő projekt esetében. A cél itt az, hogy egy diák esetében meg lehessen ítélni, hogy egy (két) félév alatt mit képes önállóan alkotni. Már a tárgy neve is mutatja, hogy ez egy önálló munka lesz, amibe természetesen a konzulensek munkája is beletartozik, de mégis igazából egy ember teljesítményét kell ez alapján értékelni. Emiatt maga a projekt sem lehet olyan komplikált és mindenre kiterjedő, mint egy való életnek szóló munka. Ez elsősorban abban mutatkozik meg, hogy sok dolgot el lehet és el is kell hanyagolni, mert mindenre nem juthat idő. Emiatt fontos, hogy ki tudjuk emelni azokat a sarokpontjait egy projektnek, amit egy ember is el tud végezni ennyi idő alatt, de mégis egy egész áll belőle össze, ráadásul még értékelhető is kell, hogy legyen.
A projektet nem az üzleti szférának tervezzük és kivitelezzük. Nem kell foglalkozni egyelőre gazdaságossági és marketing elemekkel, a külcsín is mellékes még ezen a szinten. De emellett olyan területekkel is csak érintőlegesen foglalkozunk, mint például a rendszer integrálhatósága, az emberi felhasználásból fakadó problémák, vagy az üzemeltetés problémái. A hangsúly abba az irányba tolódik el, hogy láthatóvá váljon a hallgató felkészültsége azokból a technológiákból és paradigmákból, melyek az alapjait képezik egy ilyen rendszernek. Mindazonáltal érdemes figyelembe venni azt a tényezőt, hogy az elkészült projektet kis ráfordítással át lehessen vinni a való életbe is, hiszen mégiscsak ezzel fogunk foglalkozni a későbbiek folyamán, így nem árt ilyen téren sem a tapasztalatszerzés. Viszont fontos megtalálni a megfelelő arányt, hogy egy kicsit konkrétabb legyek, nem szabad eltölteni két hetet az amúgy is szűk időből a grafikai látványterv kidolgozására, téve mindezt például a tesztelési terv rovására.

1.3. Szereplők és igényeik:

A rendszert alapvetően két szereplőre lehet lebontani, viszont közülük csak egyikük igazi felhasználója a programnak, mivel a kliens szerver architektúrát feltételezve csak a kliens valódi felhasználó. A szerver oldal ugyanis el van rejtve a felhasználók elől abban az értelemben, hogy csak a fejlesztő fér hozzá, a valódi felhasználók nem. Ők egy kliens oldali alkalmazást futtatnak a megfelelő eszközükön. Mégis meg kell azonban különböztetni egy másik típusú felhasználót. Ők azok, akiket a feladatleírásban statikus „felhasználónak" neveztem, vagyis akik csak információt (pozícióadatokat) szolgáltatnak bizonyos, a valódi felhasználók érdeklődésére számot tartó helyekről. Nyilvánvalóan az ő igényeiknek is meg kell tudni felelni, mert egy esetleges valós életbeli hasonló projektben ők jelenthetik a bevétel egy jelentős részét, hiszen ez tulajdonképpen egy hirdetési felület lehet számukra.

1.3.1. Valódi felhasználók:

A legfontosabb, amit egy valódi felhasználóról megjegyezhetünk, hogy mobil eszközt fog használni, ennek minden előnyével és hátrányával együtt. Legfontosabb paramétere ezeknek, hogy viszonylag kis sávszélességen tudnak csak kapcsolatot teremteni más eszközzel. Az elsődleges korlátunk tehát a sávszélesség lesz. Mivel emberek fogják használni a kliensünket, ezért fontos, hogy a kezelése emberléptékkel számolva gyors legyen. Ez azt jelenti, hogy nem szívesen várunk hosszú, 5-10 másodperceket egy utasítás végrehajtására. Viszont arra szinte mindenki hajlandó, hogy egy alkalmazás futása során egyszer, de csak egyszer a futás elején várjon egy hosszabb időtartamot. Ezt az időtartamot kell kihasználnunk minden nagyobb méretű adat átvitelére, ez után már törekedni kell, hogy minimalizáljuk az adatok áramlását a kliens és a szerver között, előnyben részesítve a kliensben végzett műveleteket.
Azt is meg kell jegyezni, hogy ezek az eszközök általában kis számítási kapacitással rendelkeznek, ez alól azért a laptopok kivételek, viszont a GPS-szel ellátott eszközök között kisebb a részarányuk, és fejleszteni mindig a legrosszabb esetre kell. Természetesen ez csak viszonylagos, de a szerverhez képest mindenképp jelentősnek tekinthető a különbség. Emiatt meg kell fontolni például, hogy egy műveletet elvégeztessünk a klienssel, vagy küldjük el kérésként a szervernek, majd az csak a választ küldje vissza.
A fenti két megfontolás miatt egyelőre a tervezés ezen fázisában még nem zárható ki egyik módszer sem a kliens és a szerver közötti feladatmegosztással kapcsolatban. Tehát a tervezés és főleg a komponensek esetén figyelembe kell venni azt az esetet, amikor a kliens csak kérést küld a szervernek, és onnan válasz érkezik (gyakorlatilag távoli eljáráshívás vagy webservices), és azt az esetet is, amikor a kliensre nagyobb mennyiségű adatot egyszerre letöltünk, és később ezen végzünk különböző műveleteket. Az első megoldás nyilván jobb, mivel a nagyobb teljesítményű szerverre bízza az adattárolást és a feldolgozást is, viszont a sávszélesség szűk keresztmetszete miatt a második eset is szóba kerülhet.
A harmadik nem elhanyagolható momentum, hogy az ilyen eszközök megjelenítőképessége szintén korlátozott (természetesen a laptopok itt is kivételek). Ez számunkra előny, mivel nem kell komoly grafikai teljesítmény a megjelenítéshez illetve kisebb adatmennyiség átvitele is elegendő a kijelző maximális kihasználásakor is. Abból a szempontból viszont hátrány, hogy a grafikus felület tervezésekor vissza kell fogni magunkat, egy kis felületre nem zsúfolhatunk be túl sok információt.
Összességében elmondható, hogy egy kompaktul megtervezett felhasználói felületet kell a kliensnek készíteni, melyen könnyen és értelmesen lehet navigálni és a különböző funkciókat kihasználni.

1.3.2. „Hirdetők”:

Ezekkel a felhasználókkal kevesebb a „probléma”, hiszen az általuk támasztott követelmények lényegesen egyszerűbbek, mint az előző esetben. Felhasználói felületként a legegyszerűbben egy webes felület képzelhető el, melyen keresztül be tudnak jelentkezni a rendszerbe, és be tudják vinni az adataikat. Jelen esetben, mivel nem komplett rendszert tervezek, el lehet tekinteni olyan megoldások megvalósításától, mint a számlázási adatok gyűjtése és egyéb, a való élettel kapcsolatos adminisztratív problémák kezelése.

1.3.3. Szerver oldal:

A szerver oldalon nem beszélhetünk igazi felhasználóról. A fejlesztő elkészíti az alkalmazást, majd innentől kezdve csak akkor kell hozzányúlni, ha szükséges. Természetesen egy felhasználói felület azért nem haszontalan, ezen keresztül a karbantartást könnyebben meg lehet oldani. Ebben azonban a követelmények sokkal inkább a funkcionalitást követelik meg, mint sem a megjelenést.

1.4. Komponensek, erőforrásaik, interfészeik:

1.4.1. Kliens oldali komponensek listája

1.4.1.a Kapcsolatfelépítés: ez a komponens valószínűleg egy előre megírt és tesztelt komponens lesz, mivel ez egy igen gyakran használt része a különböző programoknak. Ami egyéni, azok a kapcsolatfelvételkor elküldött adatok lennének. Nyilvánvalóan első lépésben el kell küldeni a megszerzett GPS koordinátákat, de ezen kívül az eszközön való megfelelő futtatás érdekében olyan adatokat is el kellene küldeni, mint például az eszköz felbontóképessége, maximális memóriája, stb.

1.4.1.b. Megjelenítés: mivel Java alapon létezik a MapViewer alkalmazás, ezért valószínűleg ennek egy tárolt változatát fogom a megjelenítésre használni. A komponens bemenete egy adatbázis lekérdezés eredményéből származó térképformátumú fájl lesz, amely majd a szervertől érkezik.

1.4.1.c. Kontroller: ez a komponens kezelné a különböző felhasználói interakciókat, és ezeknek megfelelően cselekedne. Ilyen interakciók lennének például térképen való mozgás négy irányban, megjelenítési rétegek változtatása (utak, domborzat, stb.), többfelhasználós rendszerben a többi felhasználó megtekintése, legrövidebb útkeresés hozzájuk, stb.

1.4.1.d. GPS: valószínűleg ezt is előre megírt változatban fogom átvenni, mivel ennek a technológiai háttere nem témája ennek a feladatnak, így ebben nem is szeretnék egyelőre elmélyedni. Mindazonáltal arra azért figyelni kell, hogy az így lekérdezett adatok kompatibilisek legyenek az Spatial koordinátarendszereinek valamelyikével, hogy az együttműködés megoldható legyen.

1.4.2. Szerver oldali komponensek listája

1.4.2.a. Kapcsolatfelépítés: ez a komponens valószínűleg egy előre megírt és tesztelt komponens lesz, mivel ez egy igen gyakran használt része a különböző programoknak. Az egyediség itt is jelentkezik abban a formában, hogy a kapcsolatfelépítéskor átvett adatok sajátosak. Továbbá fontosak az eszköz paraméterei is, amik a későbbi működést jelentősen befolyásolják. Fontos megjegyezni, hogy a szerver oldal egyszerre több klienssel is kommunikál. Ezen kliensek száma általában korlátos, így könnyebben kezelhető a rendszer. Lényeges paraméter még, hogy a szerver párhuzamosan kezeli-e a klienseket, vagy sem. Mivel az előbbi megoldás sokkal gyorsabb működést és nagyobb skálázhatóságot biztosít, mindenképpen amellett fogok dönteni.

1.4.2.b. Hirdetői oldal: egy egyszerű JavaServlet alkalmazás lenne, ami arra alkalmas, hogy a hirdetők tudjanak regisztrálni, megadni a saját pozíciójukat illetve a hirdetett hely egyéb adatait (pl. cím, elérhetőség, weblap, stb.). Nem terjedne azonban ki különböző, egyébként egy valós alkalmazásban fontos elemekre, mint például a fizetési paraméterek és a számlázás lehetőségei.

1.4.2.c. Lekérdező: gyakorlatilag a szerver oldal lelke lenne. Innen érnénk el elsősorban az adatbázist, ez jelentheti akár egy hirdetés tartalmának lekérdezését, egy felhasználó pozíciójának lekérdezését vagy akár térképi lekérdezést is a Spatial adatbázisban. Tulajdonképpen tárolt, paraméterezhető SQL lekérdezések gyűjteménye.

1.4.2.d. Módosító: hasonló lenne az előző komponenshez, annyi változtatással, hogy itt nem lekérdező, hanem módosító SQL utasítások lennének, továbbra is paraméterezhető módon. Tipikus felhasználása például egy új hirdető felvétele a rendszerbe vagy egy új felhasználó belépése.

1.4.2.e. Webservices: igazából ez jelentené a kapcsolatot a kliens és a szerver között a kapcsolatépítés után. Ez azt jelenti, hogy a kliens egy szolgáltatást kér a szervertől, amit az egy tárolt eljárás formájában hajt végre, majd annak eredményét visszaküldi a kliensnek. Természetesen itt inkább csak a kisebb adatmennyiség átvitelét jelentő kérésekről lehet elsősorban szó, mivel használat közben a nagyobb adatcsomagok átküldése jelentősen lassítja a rendszert a sávszélesség szűk keresztmetszete miatt.

1.5. Környezet:

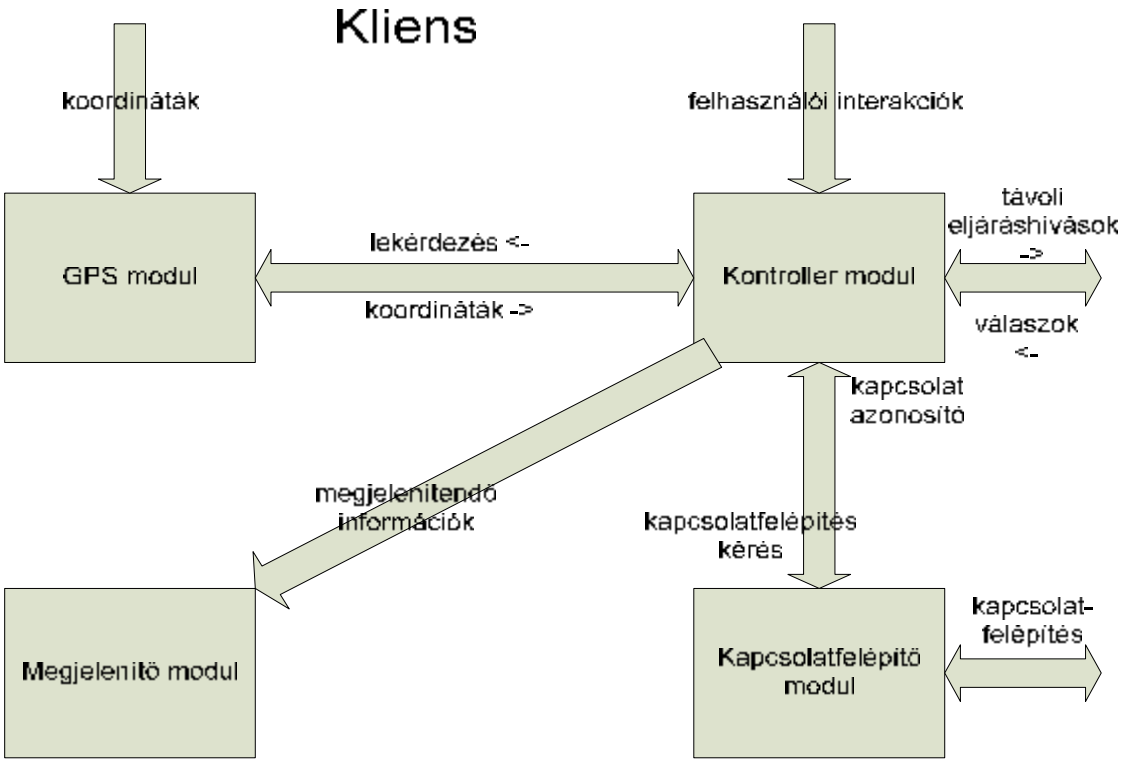
A programozási és implementációs környezet nagyrészt a feladat kiírásából fakadóan adott. Így például a program mögött elhelyezkedő adatbázisokat az Oracle 11g rendszerben fogom tárolni, erősen kihasználva a Spatial kiegészítés adta lehetőségeket. Az alkalmazás forráskódja szintén elég egyértelműen Java lesz, mivel a legtöbb mobil eszköz támogatja ezt a platformot, így nem lesz nehéz a fejlesztést áttenni rájuk. További előny, hogy az Oracle beépítetten kínál Java fordítót (JDeveloper), mely így még kényelmesebbé teszi a munkát. Ezen kívül az Oracle Spatial szerves kiegészítője a MapViewer alkalmazás, mely a térképek megjelenítését teszi lehetővé. Ezt is Java-ban írták, ráadásul a JDeveloper is támogatja a használatát. A Java alkalmazása még azzal az előnnyel is jár, hogy a kliens-szerver architektúra megvalósítása sem jelent nagy nehézséget, mivel ez funkció a Java-ban egyszerűen megvalósítható, így a kommunikációs háttér megvalósítása sem fog túl nagy

feladatot jelenteni. Java-ban továbbá van lehetőség webes felületek implementálására (Java Servlet) is, így egy felületen meg tudom oldani a különböző felmerülő problémákat.
Mivel a Java alapvetően platformfüggetlen programozási nyelv, így a szoftvereket futtató operációs rendszer típusa csaknem lényegtelen. Azért nem szabad ezt a paramétert sem elhanyagolni, de nagy vonalakban elmondható, hogy a GPS tudással rendelkező mobil eszközök nagy része Windows vagy Symbian alapú operációs rendszert használ, továbbá a fejlesztés alatt én is a Windows-t fogom előnyben részesíteni. A szerver oldali alkalmazás futása elsődlegesen ugyanazon a gépen fog történni, ahol a fejlesztés is, emiatt itt problémák talán csak kisebb mértékben fognak jelentkezni. A későbbiekben lehet szó nyilvános webszerveren való üzemelésről is, ez azonban még a jövő tárgya egyelőre.
Mivel nem rendelkezem egyelőre olyan eszközzel, ami GPS tudással rendelkezne, ezért valószínűleg valamilyen, az Interneten fellelhető emulátort fogok majd használni a teszteléshez.

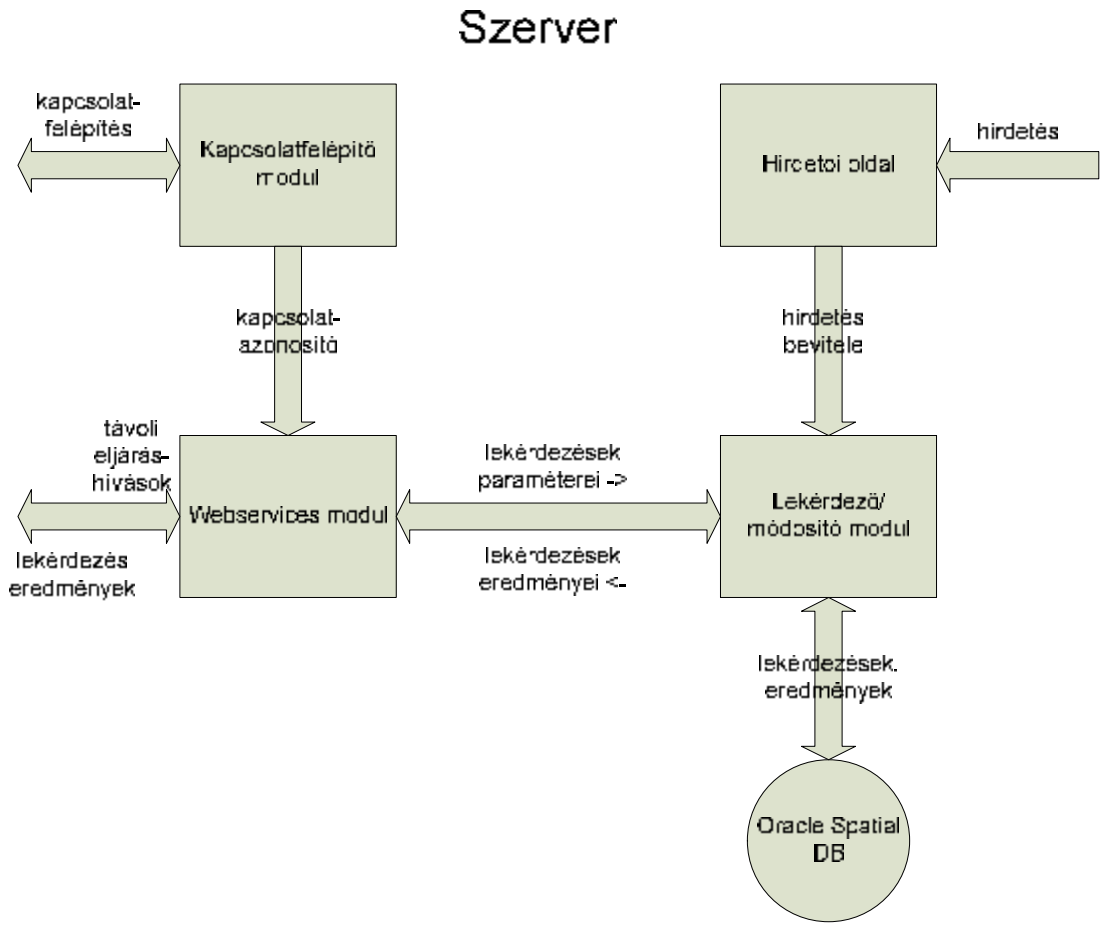
1.6. Blokkvázlatok

1.6.1. Teljes rendszer

1.6.2. Kliens oldal



1.6.3. Szerver oldal



2. Tesztelési terv

2.1. Kliens oldal

2.1.a. Kapcsolatfelépítés: mint már a korábbiakban eldöntöttem, a kapcsolatfelépítő modul egy előre gyártott és már kipróbált elem lesz. Emiatt itt a tesztelést nagy valószínűséggel már elvégezték helyettem. Mivel azonban nyilván kisebb módosítások szükségesek lehetnek a modulon, hogy megfelelően illeszkedjen az alkalmazás egészébe, ezért itt is szükséges lehet tesztelés. Egy tipikus tesztvektor a kliens oldali kapcsolatfelépítő modul számára így nézne ki: `<szerver ip cím><szerver port szám><kliens ip cím><kliens port szám><eszköz paraméterek>`.
Az első két mező nem igényel különösebb magyarázatot. A második két mező értékei a szerver számára szükségesek, hogy tudjon megfelelő válaszüzenetet generálni. Az utolsó mezőben a kapcsolódó eszköz paramétereit állíthatjuk be.
A helyes tesztvektor nyilván a pontos szerver és kliens oldali címeket tartalmazná, jól beállított paraméterekkel. Helytelen tesztvektorban lehet rossz mind a szerver, mind pedig a kliens oldali címzés. Előbbi esetben hibaüzenetet várunk, mely szerint nem lehetett a szerverhez csatlakozni, míg utóbbi esetben időtúllépés miatt nem sikerülhet a kapcsolatfelépítés, mivel a szerver nem tud hova válaszolni.
Önmagában egy kliens oldali kapcsolatfelépítő modult tesztelni nem igazán érdemes, egy valódi tesztet a szerver oldallal együtt kell elvégezni.

2.1.b. Megjelenítés: a megjelenítő modul nem igényel komoly tesztelést. A megjelenített képernyő kép mindig ugyanúgy néz ki, az egyetlen dinamikus elem benne a térkép. Az igazán tesztelendő feladat, hogy a különböző minőségű és felbontású képernyőkön is elfogadható minőségű képet adjon az alkalmazás anélkül, hogy a felhasználónak minden alkalommal ezt be kéne állítania.

2.1.c. Kontroller: a kliens oldal legbonyolultabb része. Funkcionális tesztelés során le kell ellenőrizni, hogy helyesen működnek-e a különböző funkciók, vagyis:

- felhasználói interakciók (navigálás, funkciógombok) hatására milyen események következnek be, megtörténnek-e a szükséges függvényhívások, egyéb modulok felé fordulás
- GPS modultól visszaérkező információk helyes továbbítása
- szerver oldalról érkező információk helyes feldolgozása
- megjelenítendő adatok összeállítása

Itt szükséges egy igen komoly hibatűrési teszt végrehajtása is, mivel előfordulhat, hogy helytelen formátumú vagy tartalmú adatok érkeznek be, ekkor meg kell próbálni az információt újra megszerezni. Mivel ez a komponens áll kapcsolatban a felhasználóval, így szükséges annak tesztelése is, ha a felhasználó helytelenül kezeli az eszközt (rossz billentyűkombináció, meg nem engedett parancs, stb.).

2.1.d. GPS: a GPS modul is valószínűleg előre gyártott modul lesz, így ennek tesztelése is megtörtént már korábban. Tesztelni valószínűleg úgy tudom majd, hogy az adott eszközön lekérem a GPS koordinátákat, majd ezek átalakítása után megnézem, hogy mennyire pontos eredményt mérek. Itt valószínűleg több mérésre lesz szükség, melynek során megállapítható,

hogy milyen jellegű és mekkora eltéréssel mér az eszköz. Az eszközt úgy kell majd beállítani, hogy ez a hiba minél kisebb legyen.

2.2. Szerver oldal

2.2.a. Kapcsolatfelépítés: mint már a korábbiakban eldöntöttem, a kapcsolatfelépítő modul egy előre gyártott és már kipróbált elem lesz. Emiatt itt a tesztelést nagy valószínűséggel már elvégezték helyettem. Mivel azonban nyilván kisebb módosítások szükségesek lehetnek a modulon, hogy megfelelően illeszkedjen az alkalmazás egészébe, ezért itt is szükséges lehet tesztelés. Egy tipikus tesztvektor a szerver oldali kapcsolatfelépítő modul számára így nézne ki: `<kliens ip cím><kliens port szám>`.
A két mező ahhoz szükséges, hogy a szerver a megfelelő címre válaszoljon, nyilván a szervernek a saját címét nem kell elküldenie, az a kliensben tárolódik.
Önmagában a szerver oldali kapcsolatfelépítő modul tesztelése felesleges, a tesztet a kliens oldallal együtt kell elvégezni. A kliens oldal mindig csak egyetlen szerver oldali alkalmazással van kapcsolatban, azonban a feladat jellegéből adódóan a szerver egyszerre több klienssel is kommunikálhat. Ezért a teszt fontos eleme kell, hogy legyen több kliens kiszolgálásának vizsgálata. A teszt során követni kell, hogy a modul eleget tesz-e a többfelhasználós működés követelményeinek, nem növekszik-e meg túlságosan a válaszideje sok kliens esetén, mi történik, ha egy vagy több kliens nem válaszol, megszakad a kapcsolat, stb.

2.2.b. Hirdetői oldal: a hirdetői webes felület önmagában is igen könnyen tesztelhető. A helyes működés tesztelésénél egy új felhasználó első beregisztrálását, későbbi bejelentkezését és hirdetésfelvételét, majd esetleg regisztrációjának törlését lehet vizsgálni. Hibás működés kezelését vizsgálva olyan eseteket kell megnézni, mint bejelentkezés regisztráció nélkül, hibás felhasználónév vagy jelszó bejelentkezéskor, értelmezhetetlen hirdetés feladása, hibásan kitöltött hirdetési űrlap. Itt a hibás használat tesztelése igen fontos, mivel a felhasználók kommunikálnak ezen a modulon keresztül az alkalmazással, így igen nagy a hiba lehetősége. Regisztrációnál érdemes kiszűrni a robotok által gyártott hirdetést egy humánfelismerő módszerrel.

2.2.c. Lekérdező: valószínűleg a legtöbb tesztelést igénylő modul a módosító modullal együtt. Tárolt eljárásokat tartalmaz, ez a modul fog az adatbázishoz fordulni. Itt valószínűleg érdemes kimerítő tesztelést végezni, vagyis minden lehetséges lekérdezést letesztelni. Nyilván a bemeneti tesztvektorok a paraméterezhető SQL utasítások paraméterei lesznek. Ez a modul tesztelhető úgy is, hogy csak a modul kimenetét vizsgáljuk, vagyis hogy megfelelő SQL utasítást hajtott végre a megadott paraméterekből. A másik tesztelési lehetőség pedig, hogy az adatbázissal együtt teszteljük, vagyis kimenetként a lekérdezések eredményeit várjuk, ekkor már magukat a lekérdezéseket is teszteljük. Ez utóbbihoz viszont szükség van az adatbázisra is, ami jelenleg még nem is létezik. Így ezeket a teszteket konkrétan majd a fejlesztés egy későbbi fázisában lehet összeállítani.

2.2.d. Módosító: valószínűleg a legtöbb tesztelést igénylő modul a lekérdező modullal együtt. Tárolt eljárásokat tartalmaz, ez a modul is az adatbázishoz fog fordulni. Itt valószínűleg érdemes kimerítő tesztelést végezni, vagyis minden lehetséges módosítást letesztelni. Nyilván a bemeneti tesztvektorok a paraméterezhető SQL utasítások paraméterei lesznek. Ez a modul tesztelhető úgy is, hogy csak a modul kimenetét vizsgáljuk, vagyis hogy megfelelő SQL utasítást hajtott végre a megadott paraméterekből. A másik tesztelési lehetőség pedig, hogy az

adatbázissal együtt teszteljük, vagyis kimenetként a módosítások megfelelő végrehajtását várjuk, ekkor már magukat az SQL módosító utasításokat is teszteljük. Ez utóbbihoz viszont szükség van az adatbázisra is, ami jelenleg még nem is létezik. Így ezeket a teszteket konkrétan majd a fejlesztés egy későbbi fázisában lehet összeállítani.

2.2.e. Webservices: hasonló feladatú modul, mint kliens oldalon a kontroller. A tesztelés célja itt is az, hogy egy beérkező kérés hatására a megfelelő eseményláncon haladjunk végig, a végén egy jól paraméterezett lekérdező vagy módosító utasítást hívjunk meg. Tesztelhető még továbbá, hogy a visszaérkező eredmények hogyan alakulnak át abba a formába, ahogy azt visszajuttatjuk a kliens felé. Igen fontos tesztelni ezeken túl a több felhasználó kezelését, mivel ez az a modul, amelyik szétválasztja a különböző kliensek kéréseit, innen kezdve a különbözőség már csak az SQL utasítások egy paraméterében lesz nyomon követhető.

2.3. Globális tesztelés

A fentiekből jól látható, hogy a tesztelés igen bonyolult és többszintű lesz. Az egyes modulok tesztelését követően külön a kliens és a szerver oldal tesztjét is el kell végezni, immáron nem apró részletekre figyelve, hanem a globális működést vizsgálva. Ezután következhet az egész rendszer tesztje.
A tesztelés során jól elkülöníthető problémákra kell helyezni a hangsúlyt, vagyis az általános funkcionális tesztelésen kívül vannak olyan területek, amelyekre különösen figyelni kell. Az egyik ilyen azon interfészek tesztelése, amelyeken keresztül a felhasználók a rendszerrel kommunikálnak. Mivel jelentősen megnő a hibás használat esélye, tisztességesen le kell tesztelni, hogy az alkalmazás mennyire tűri ezt.
A másik ilyen terület a többfelhasználós működés. Itt tesztelni kell, hogy a rendszer hány felhasználóig képes működni, hogyan alakulnak a válaszidők a felhasználók számának növekedésével, mennyire stabil és skálázható a rendszer, mennyire bírja a hirtelen terhelésnövekedést (pl. egyszerre több tíz felhasználó kér le adatot a szerverről). Ez nagyon fontos lehet egy későbbi bővítés vagy fejlesztés során.
Nem szabad elfelejteni, hogy az alkalmazást embereknek fejlesztjük, vagyis nem szabad csak funkcionálisan és informatikus szemmel tesztelni, hanem a leendő felhasználókkal is. Ennek érdekében a már általunk letesztelt verziót oda kell adni az embereknek, hogy próbálják ki, használják és mondják el a véleményüket róla.

3. Üzemeltetési terv

A szoftver üzemeltetése során több alkalommal is szükség lehet külső beavatkozásra. Ilyen módosítások lehetnek a konkrét alkalmazásra nézve az alábbiak:

- új térkép adatbázis feltöltése vagy a térképek számának bővítése
- új felhasználók regisztrációjának ellenőrzése, felhasználó esetleges törlése
- új hirdetők regisztrációjának ellenőrzése, hirdetés esetleges törlése
- felhasználók maximális számának változtatása
- naplófájlok mentése, vizsgálata

Ezeknek a feladatoknak az ellátására mindenképpen szükség lesz egy adminisztrációs felület létrehozására. Ez gyakorlatilag egy interfész felület lesz melyen keresztül el lehet érni

bizonyos beállítási lehetőségeket, ellenőrizni lehet a működést. A két legfontosabb funkció tehát a monitorozás és a karbantartás.

3.1. Monitorozás

A monitorozás alapeleme a naplókészítés lesz. Tehát minden fontos eseményről naplóbejegyzés készül, mely később visszanézhető és archiválható lesz. A napló nyilván legegyszerűbben az adatbázisban tárolódna, ahonnan mondjuk naponta egyszer fájlba kiíródna. A naplóba az alábbi eseményeket tartom fontosnak belevenni:

- új felhasználó regisztrációja
- felhasználó bejelentkezése (kapcsolatfelvétel a szerverrel)
- felhasználó lekérdezési művelete
- felhasználó kijelentkezése (kapcsolat megszakadása)
- felhasználó törlése (akár általunk, akár saját magát)
- új hirdető regisztrációja
- hirdető bejelentkezése
- új hirdetés felvétele vagy meglévő hirdetés módosítása
- hirdető törlése (akár általunk, akár saját magát)
- minden adminisztrátor által elkövetett esemény

A fentieken kívül a napló bizonyos időközönként kell, hogy tartalmazzon egy összefoglalást a rendszer állapotáról, a könnyebb áttekinthetőség miatt. Itt az alábbi tulajdonságokat lehetne összegezni, például 12 vagy 24 óránként:

- új felhasználók száma
- új hirdetők száma
- új hirdetések száma
- összes felhasználó száma
- összes hirdető száma
- összes hirdetés száma
- bejelentkezések száma
- összes lekérdezés száma
- adatbázis mérete (meglévő szabad hely nagysága)

Ezekből az adatokból naponta összeállna egy naplófájl, amit külső táron lehetne utána elhelyezni. Az adminisztrációs felületen lenne lehetőség akár a régebbi, akár az aktuális naplófájl megtekintésére, szükség esetén azonnali mentés végrehajtására.

3.2. Karbantartás

A monitorozás mellett fontos biztosítani a felületet arra is, hogy a monitorozás során felmerülő rendellenességek egy része orvosolható legyen. Mivel a készülő alkalmazás nem a piacra készül, hanem alapvetően a tanulmányi előmenetelt hivatott segíteni és bemutatni, ezért nem tervezem olyan szintű beállítási lehetőségekkel felvértezni, mint amilyenre a való életben szükség lenne. Tehát olyan állítási lehetőségek mindenképpen kimaradnak belőle, mint például az adatbázis felé irányuló lekérdező és módosító függvények változtatása. Ennek ellenére azért néhány dolgot mindenképpen el kell tudni érni a programkód megváltoztatása nélkül is. Ilyen beállítási lehetőségek lesznek elérhetőek:

- felhasználók adatainak megtekintése, szükség esetén felhasználó felvétele vagy törlése
- hirdetők adatainak és a hirdetéseik megtekintése, szükség esetén törlésük
- maximális felhasználószám változtatása

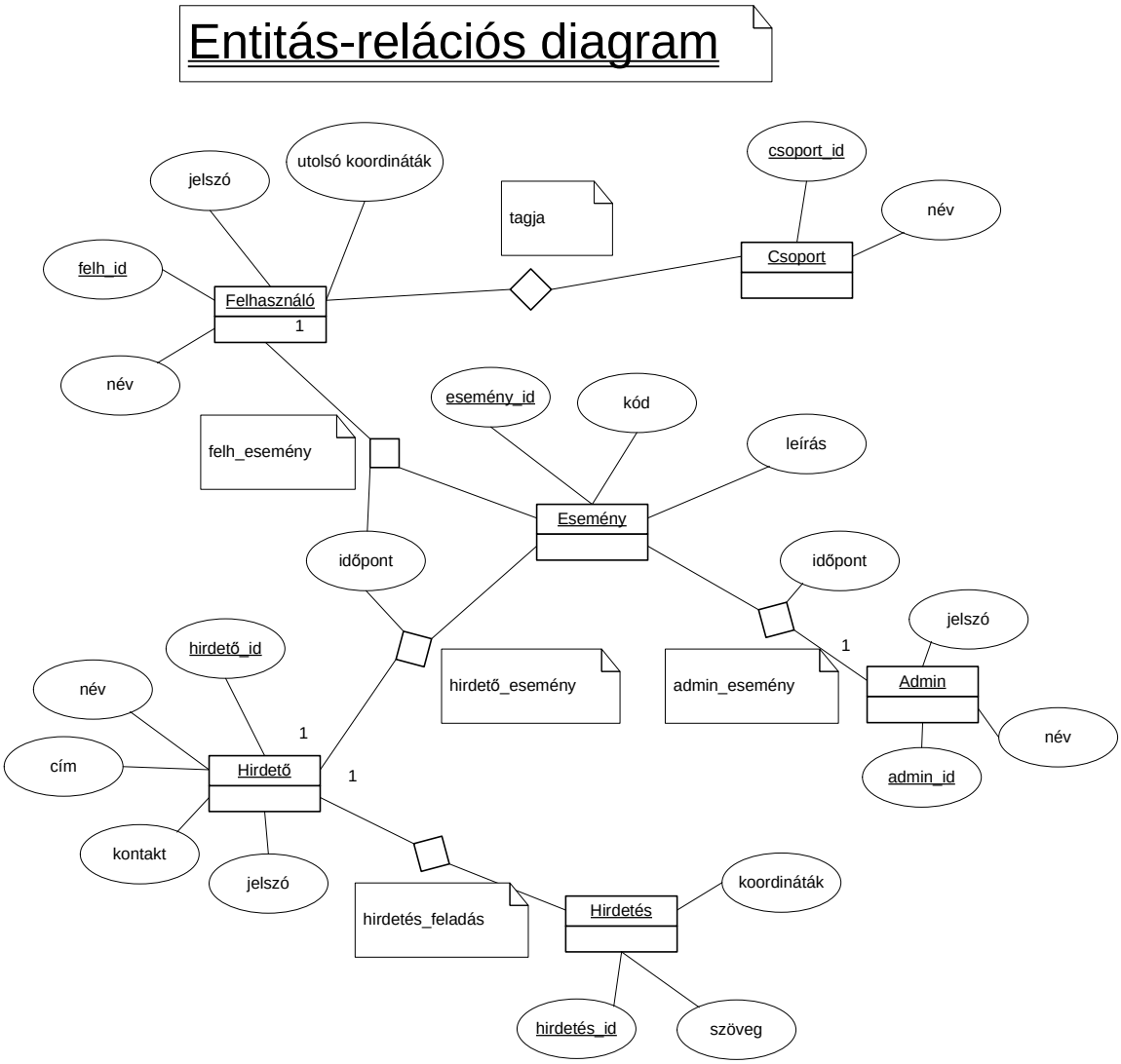
Ezekre a műveletekre akkor lesz szükség, ha azt a naplófájlok tartalma indokolttá teszi. Esetleg lehetséges volna automatikus karbantartási műveletek végzése is, vagyis olyan feltételeket vagy korlátokat definiálni, melyek valamilyen akciósorozatot hajtanak végre. Ilyen lenne például, hogy a naplófájl túl nagy mérete esetén a mentése külön fájlba automatikusan megtörténne, vagy például új felhasználót csak akkor engedne be a rendszer, ha van is erre kapacitása. Ez a funkcionalitás azonban egyelőre nem a terv része, csak mint lehetséges fejlesztési irányt jelöltem meg. Nyilván egy éles rendszerben ezek a komponensek nem hagyhatóak el.

4. Adatszerkezeti terv

Az adatszerkezeti terv a passzív erőforrások leírását tartalmazza. Passzív erőforrások közé szokás sorolni mind az adatbázisokat, mind pedig a naplófájlokat. Én ezt némiképpen összemostam, amennyiben a naplóbejegyzéseket is az adatbázisban szeretném tárolni, és csak bizonyos időközönként (például 12 vagy 24 óránként) menteném ki külső fájlba a bejegyzéseket. Így a friss, nem túl régi bejegyzések keresése és megtekintése gyorsabb lehet, viszont fontos, hogy az egy napnál régebbi események is visszakereshetők legyenek. Így tulajdonképpen az adatbázisban is lesz eseménynapló és önállóan is lesznek naplófájlok.

4.1. Adatbázis terv

Az adatbázis tulajdonképpen két nagy részre bontható. Az egyik része ennek maga a térkép, amit ugyebár készen fogok beszerezni, kezdetben nagy valószínűség szerint a Navteq cég által kiadott, ingyenesen elérhető Magyarország térkép lesz a rendszer alapja. Később természetesen a térképek száma bővülhet.
A másik rész az, amit én tervezek meg és implementálok. Ezekben a táblákban fognak tárolódni a felhasználók adatai, a hirdetők adatai és hirdetéseik, valamint a friss naplóbejegyzések.
Adatbázis megtervezéséhez az általunk tanult legjobb eszköz az entitás-relációs diagram készítése, majd ebből a relációs adatbázisséma létrehozása. Én is ezt a folyamatot követtem, így először a diagramot készítettem el, majd abból a tanult megoldásokkal készítettem el a relációs sémát.



A diagram elkészítésénél igyekeztem betartani azokat a konvenciókat, melyeket korábban tanultunk. Az ábrát Microsoft Visio programmal készítettem, sajnos ez a program nem támogatja az általunk tanult entitás-relációs diagramok elemkészletét. Így különböző kényszermegoldásokat kellett alkalmaznom, de körülbelül sikerült a tanultaknak megfelelő ábrát készítenem. A Visio által támogatott adatbázis modell elemkészlettel is el fogom készíteni ezt az ábrát, az sokkal közelebb áll már a séma formához, így valószínűleg gyorsítani fogja az adatbázis felépítését.

Séma:

- felhasználó (felh_id, név, utolsó koordináták, jelszó)
- csoport (csoport_id, név)
- hirdető (hirdető_id, név, cím, kontakt, jelszó)
- hirdetés (hirdetés_id, szöveg, koordináták)
- esemény (esemény_id, kód, leírás)
- admin (admin_id, név, jelszó)
- csoport_tagság (felh_id, csoport_id)
- hirdetés_feladás (hirdető_id, hirdetés_id)
- felh_esemény (sorszám, esemény_id, felh_id, időpont)
- admin_esemény (sorszám, esemény_id, admin_id, időpont)
- hirdető_esemény (sorszám, esemény_id, hirdető_id, időpont)

A sémában szereplő entitások meglehetősen egyértelműek, a rendszert használó szereplők és az általuk létrehozott elemek (hirdetés, esemény) listája. Attribútumaik között minden esetben szerepel egy xxx_id tag, ami elsődleges kulcsként a megkülönböztethetőséget szolgálja. A többi attribútum egyértelmű, talán csak a jelszavak szorulnak némi magyarázatra. Első körben úgy gondoltam, hogy külön táblában tárolnám a jelszavakat és további három táblában az összekapcsolódást a háromféle felhasználóval. Ezt azonban meglehetősen helypazarlónak éreztem, egyszerűbb megoldás, ha a felhasználó mellett kerül eltárolásra. A titkosság biztosítására megoldást jelenthet egy hash függvényes leképezés, vagyis az adatbázisban nem a jelszavakat tárolnám, hanem azok hash-lenyomatát.
Az entitásokat összekötő relációk is értelemszerűen következtek az eddigiekből. Majdnem minden ilyen relációnak van egy sorszám nevű attribútuma, amely elsődleges kulcsként is funkciónál. Ez főleg naplóbejegyzések esetén gyorsan igen nagy szám lehet, emiatt biztosítani kell, hogy amikor a naplóbejegyzések kiírásra kerülnek külső naplófájlra, akkor ezt a sorszámot vissza kell állítani a kezdőértékére. A relációk többi attribútuma két idegen kulcs, hivatkozás a relációban részt vevő két entitásra, emellett két relációban ezek töltik be az elsődleges kulcs szerepét is. Ezen kívül a naplóbejegyzéseknek van még egy attribútuma, ez az időpont. Ezt azért vettem ide, mert igazából sem az esemény leírásához, sem pedig az eseményt elkövető felhasználóhoz nem köthető, így jó megoldásnak tűnik magához az eseményhez kötni ezt az információt.

4.2. Naplófájlok

A naplófájlok szerkezete nagyban fog hasonlítani naplóbejegyzésekhez tartozó adatbázissorokra. A különbség csak annyi, hogy egyetlen bejegyzésben lesz minden információ egy eseményről, nem szétszórva több helyre. Így egy naplóbejegyzés a következőképpen nézne ki:

<időpont><felhasználó típus><felhasználó név><esemény leírás><…>

A felhasználó típusa azt adja meg, hogy egy sima felhasználó, egy hirdető vagy egy admin végzett el valamilyen műveletet. Erre azért van szükség, mert nem követeljük meg, hogy két különböző típusú felhasználónak ne lehessen ugyanaz a felhasználóneve. Az esemény leírása egy előre megadott listából kerülhet ki, ezen olyan események szerepelnek majd, mind például belépés, kilépés, adatmódosítás, stb. Amennyiben szükséges, egyéb mezők is lehetségesek a fentieken kívül.

Mint már korábban a 3.1. fejezetben leírtam, amikor a naplófájlokat külső fájlba kimentem, akkor lehetséges egy összefoglaló jellegű bejegyzés is, mely segíthet a rendszer működésének áttekintésében. Ennek leírása fentebb található.