Debreceni Egyetem

Informatika Kar

Informatikai Rendszerek és Hálózatok Tanszék

Földrajzi pozíciók és kapcsolódó információk megjelenítése – webes alkalmazás fejlesztése Java EE és .NET környezetben

Témavezető:
Dr. Rutkovszky Edéné
ügyvivő szakértő
Külső konzulens:
Pajna Sándor
HBMÖ. IK
igazgató

Készítette:
Tándor Ágnes
programtervező informatikus

Debrecen

2009

Tartalomjegyzék

I. Bevezetés

Mindennapi életünkben nagy szerepet játszik az Internet, és mára már az informatikai rendszerek között jelentős szerepet töltenek be a webes alkalmazások, így szakdolgozatom témájaként egy ilyen rendszer fejlesztésének bemutatását választottam. A dolgozatban egy általam fejlesztett alkalmazás segítségével tekintem át a fejlesztéshez kiválasztott technológiákat, és az alkalmazáson keresztül mutatom be a szoftverfejlesztés lépcsőfokait.

A ma használatos Internet elnevezés létrejötte, és út a Web 1.0-től a Web 3.0-ig:

A webes alkalmazások kialakulásához hosszú út vezetett, amely az Internet létrehozásával kezdődött, és a fejlődés a Web 3.0 irányába mutat.

A fejlődés rövid áttekintése, legfontosabb lépései:

1960-as évek elején hadifejlesztés keretein belül az Egyesült Államokban a RAND Corporation azon probléma megoldását kereste, hogy hogyan érhető el, hogy Amerika és a hadvezetés központjai közötti kommunikáció bármilyen nagyobb támadás esetén is fennmaradjon.

1969-ben a Defence Advanced Research Project Agency (DARPA) elvei alapján (egy többközpontú, csomagkapcsolt, (ahol az adatok továbbítása kisebb csomagokban történik) hálózati kommunikációs rendszer létrehozása) kezdte működését az APRANET.

1983-ban az ARPANET-ből MILNET néven leválasztották a hadászati szegmenst, így ezzel megszületett a mai fogalmaink szerinti Internet.

1989-ben szűnt meg az APRANET formálisan, hogy a helyét átadja a fejlettebb gerinchálózatoknak.

1980-as évekre az USA minden egyeteme rácsatlakoztatta helyi számítógépeit a hálózatra.

1980-as évek második felében Nyugat-Európában indult meg a bekapcsolódott gépek számának növekedése.

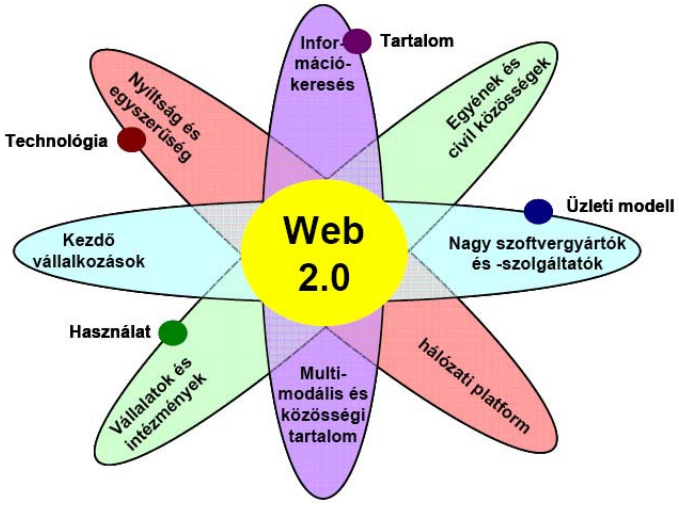
1990-es évekre Kelet-Európa is rácsatlakozott a hálózatra, így Magyarországon is ekkor jelent meg az Internet.

1992-ben Tim Berners-Lee nevéhez köthető World Wide Web (WWW) megjelent, mely már mindenki számára lehetővé tette az Internet elérését. Tim Berners-Lee és Robert Cailliau elgondolása alapján grafikus felhasználói felületek váltották fel a parancssorokat.
1990-ben Tim Berners-Lee elkészítette az első böngészőt és az első weblapokat.

Ettől kezdve **Web 1.0**-ról beszélhetünk. A weblapok statikusak voltak, csak olvasható információt tartalmaztak. A Web 1.0 idején jött létre az online kereskedelem alapja, megalakultak és megerősödtek a nagy internetes cégek (pl.: Yahoo!, Amazon.com, eBay), adás-vételi szokások átalakultak, a világhálón való keresés természetessé vált.

2004-ben Tim O'Really-nek, az O'Really Media cég alapítójának tulajdonítják a **Web 2.0** fogalmát.
„*A Web 2.0 a computeripar üzleti forradalma, amely az internetet egy alkalmazási felületté tette és segített megérteni az új platform adta lehetőségeket.*" (2006, Tim O'Really)

A Web 2.0-t négy tényező alakítja: használat, tartalom, technológia, üzleti modell.
Ezek kölcsönös egymásra hatásában van a Web 2.0 kialakulása mögötti legfontosabb hajtóerő **(1. ábra)**.



1. ábra A Web 2.0 összetevői

Kapcsolódó fogalmak:

- **Tartalommegosztás (sharing):** egyes információkat, híreket, linkeket ajánlanak fel egymás számára a felhasználók.
- **Blog (internetes napló):** a klasszikus naplók internetes megfelelője. Bárki szabadon megírhatja gondolatait, ötleteit, a vele történteket. Ennek logikus továbbfejlesztése, hogy az olvasók már hozzászólhatnak az írottakhoz, véleményezhetik azt.
- **Címkefelhő (tag cloud):** az adott oldalon előforduló, leggyakrabban használt linkek gyors és érthető összefoglalása.
- **Fórumok:** az azonos érdeklődési körű emberek közötti aránylag laza kapcsolati lehetőség. Mindenki akkor nézhet rá, amikor éppen van ideje és reagálhat is.
- **On-line irodai alkalmazások:** a saját gépen telepített programok helyett irodai alkalmazásokat lehet futtatni a weben egy böngésző segítségével.
- **Wikipédia**, illetve **wikik (ingyenes tudástárak):** ezek a szabadon fejleszthető tudástárak. Alapértelmezésben szabadon, bárki által szerkeszthetők.
- **Aukciós,** illetve on-line kiskereskedelmi **oldalak:** Bárki szabadon eladhat, illetve megvehet bármit.
- **RSS, illetve egyéb tartalomszolgáltatás:** a klasszikus hírportálok külön lehetőségként felajánlják a híreik és cikkeik rövidített verzióját – a teljes anyagra való kattintási lehetőséggel. Így a gyorshírekből csak a felhasználót érdeklőt kell átnéznie.
- **Híreket újrakeverő**, illetve szűrő egyéni oldalak: ezeket az egyes felhasználó egyéni ízlése szerint válogathatja össze, illetve konkrét tartalmi blokkokból állíthatja.
- **Podcasting:** A felhasználók feliratkozhatnak az adott anyag újdonságait tartalmazó feed-ekre (amik általában MP3 állományok). A podcasting igen egyszerűvé teszi, hogy otthon, saját ízlése szerint előállított „rádióműsorát” közzétegye az Interneten és a hallgatóit egyszerű RSS-hírekben tájékoztassa az újdonságokról.
- **On-line fájl-szolgáltatók:** az Interneten közzétett állományok között lehet böngészni, illetve az egyes tartalmakat tárolhatják a világhálón.
- **On-line játékok:** ezek a játékok többnyire egy kisebb telepítő fájllal rendelkeznek a saját gépen, de az alkalmazás lényege, hogy a hálózathoz csatlakozva játszhassanak a hasonló érdeklődésű játékosokkal. Ennek kicsit komolyabb megvalósítása a 3D-s virtuális világok létrehozása, ahol a játékosok által generált karakterek (avatárok) mászkálhatnak és kommunikálhatnak.

- **On-line térképalkalmazások:** a valódi forgatható térképek mintájára a világhálón is megjelentek a térképek (pl.: Google Maps, Yahoo! Maps, IndaGeo). Ennek továbbgondolása a felhasználók autóiban üzemeltethető Global Positioning System (GPS), ami a térképet megfelelő vevőkészülék segítségével jeleníti meg és javasol útvonalat a sofőr számára.
- **Webes alkalmazások:** kizárólag a weben létező, máshol nem is telepíthető programok (API). Ennek újabb verziójában a felhasználók maguk is írhatnak programokat (nyitott API).

A Web 2.0-nál a fejlődés nem áll meg, már manapság is újabb fejlődési lehetőségek vannak, a cél a **Web 3.0** elérése **(2. ábra)**.

2. ábra A Web várható hosszabb távú fejlődése

Többen nem egészen alaptalanul egyenlőségjelet tesznek a Web 3.0 és a személyes internet közé. De leginkább az integráló szerepe miatt kerül a figyelem középpontjába. Az eddigi honlapokra tagolt Internet helyett egységesen és földrajzi helytől, valamint hozzáférési módszertől függetlenül tudják kihasználni az informatikai technológia adta lehetőségeket.

II. Célok

A Web 2.0 elterjedésével számos technológia, eszköz jelent meg a piacon, melyek segítségével már könnyedén létre tudunk hozni egy webes alkalmazást.
Mivel a Web 2.0 világának jelentős részét képezik a webes alkalmazások, így szakdolgozatom legfőbb céljai:

1. **Egy webes alkalmazás teljes fejlesztési folyamatának és ezzel együtt a fejlesztett alkalmazás részletes bemutatása:**

   A szakdolgozatban bemutatott alkalmazás fejlesztésével egy olyan rendszer kialakítását tűztem ki célul, mely egyszerre több technológiát és esetlegesen nyitott API-t használ, illetve a szükséges információk összegyűjtése és adatbázisban tárolása után, egy használható, későbbiekben továbbfejleszthető alkalmazás lehet. A rendszer a felhasznált technológiák mellett a GPS rendszer földrajzi helymeghatározására is épít.

2. **Megfelelő technológia és eszköz kiválasztása:**
   - A fejlesztéshez technológia választásánál fontos szempont volt, hogy az egyes fejlesztői környezetek támogatást nyújtsanak az alkalmazásukhoz.
   - Tanulmányaim alatt megismert JavaServer Faces keretrendszer konkrét feladatban való alkalmazása és esetleges kiegészítőinek megismerése volt a célom, mikor a Java EE platform technológiái közül válogattam.
   - A .NET platform technológiáival tanulmányaim alatt nem találkoztam, így ezen környezetben megírt alkalmazás lehetőséget biztosított egy a Java világától különböző technológia megismerésére.
   - A dolgozatban megjelenő nyitott API által, pedig egy a fejlesztők számára lehetséges felhasználási módot akartam bemutatni. A térképek megjelenítését végző Google Maps által biztosított nyitott API az alkalmazás kezdeti követelményei alapján lett kiválasztva. Hiszen szükség volt egy olyan eszközre, mely a különböző platformok és technológiák esetén is támogatást nyújt a fejlesztéshez.

3. Saját fejlesztésű informatikai alkalmazás fejlesztése, komparatív technológiák együttes alkalmazásával:

Két különböző technológiát összehasonlítva és ez által megismerve, egy olyan webes alkalmazás létrehozása volt a cél, mely megfelelő környezetbe asszimilálva, könnyen használható, és értékes funkciókat biztosító rendszer lehet. Egy olyan alkalmazás, amely olyan technikákat, funkciókat birtokol, melyek a mai világban teret hódítanak, és az emberi munkát megkönnyítik, kiegészítik.

4. A kialakítandó alkalmazás legfontosabb funkciója:

Földrajzi koordináták és kapcsolódó attributív adatok megjelenítése térképi környezetben.

III. Anyag és módszer

Az (**1. sz. táblázat**)-ban felsorolt technológiákat, eszközöket használtam fel az alkalmazásomnál, illetve ezeket ismertetem. Majd a kiválasztott két környezet és hozzájuk kapcsolódó technológiák összehasonlítását is elvégzem a bemutatásuk után.

1. sz. táblázat – Java EE, illetve .NET környezet technológiái, eszközei

	Java EE	**.NET**
Megjelenítési réteg	JSF, Facelets	ASP.NET
Alkalmazás szerver	Glassfish	IIS
Adatréteg	JDBC	ADO.NET

A felsorolt technológiák, eszközök mellett még használom a Google Maps nyitott API-ját, mely az alkalmazásomhoz a térképet szolgáltatja. A térképen megjelenítésre kerülő földrajzi pozíciókat GPS koordináták segítségével határozom meg.

1. A Java EE környezet

1.1. Java Enterprise Edition

A Java nyelv története 1991-ben kezdődött, de a nyilvánosság előtt csak 1995-től vált ismertté. Az azóta eltelt évek során a nyelv és a hozzá kapcsolódó technológiák dinamikus fejlődésen mentek keresztül és egyre többféle feladat megoldására lettek alkalmasak. Mindez tette lehetővé a Java platform 3 „kiadásának ”(edition) megjelenését:

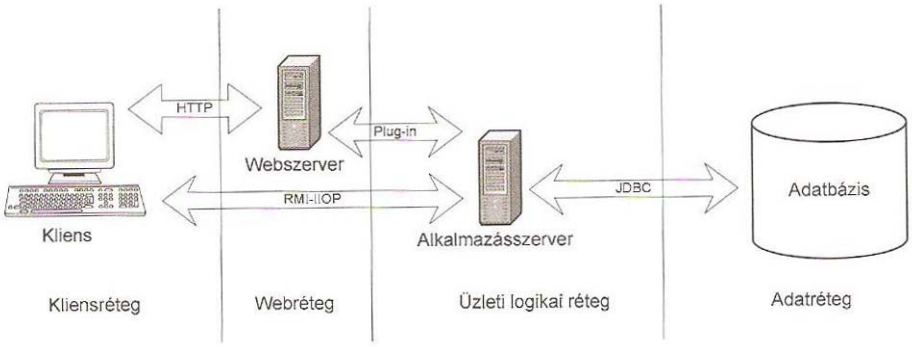
Java Standard Edition (Java SE): hagyományos desktop alkalmazások, appletek készítése.
Java Micro Edition(Java ME): korlátozott erőforrásokkal rendelkező, jellemzően mobil eszközökre való fejlesztés.
Java Enterprise Edition(Java EE): elosztott, sok felhasználóval rendelkező, vállalati méretű szoftverrendszerek kialakítása, az Enterprise technológiák mellett a Java EE-ben rendelkezésre áll a Standard Java API is.

"A Java Platform, Enterprise Edition (Java EE) ipari szabvány, amely lehetővé teszi hordozható, robusztus, skálázható és biztonságos szerveroldali Java alkalmazások fejlesztését. A Java Platform, Standard Edition (Java SE) által alkotott szilárd alapon építkezve a Java EE webszolgáltatásokat, komponensmodellt, menedzsment és kommunikációs API-kat kínál, ami a szolgáltatásorientált architektúrát alkalmazó nagyvállalati fejlesztésekben, illetve a következő generációs webalkalmazások terén ipari szabvánnyá minősíti." (2009, Sun Microsystems)

A követelmények, amelyeket egy Java EE platformon fejlesztett alkalmazásnak biztosítania kéne, middleware szolgáltatásoknak hívjuk. Ilyen például az adatok perzisztens kezelése, a többszálúság biztosítása, a tranzakciókezelés, a rendszer távoli elérése, a névszolgáltatás, a rendszer skálázhatósága, az aszinkron üzenetküldés támogatása és a rendszer biztonságossá tétele.

1.2. Az n rétegű architektúra és a Java EE alkalmazásszerver

Java EE esetében az alkalmazások futási környezetét a Java EE alkalmazásszerver adja, amely beilleszkedik a rendszerarchitektúrába **(3. ábra)**.



3. ábra Az n rétegű architektúra

Az architektúrában megkülönböztetünk rétegeket, melyek egymásra épülnek és mindig a közvetlenül alattuk lévő réteget felhasználva látják el feladatukat.

Az egyes rétegek feladata:

Adatréteg: az adatok perzisztens tárolása és azokon végezhető műveletek támogatása

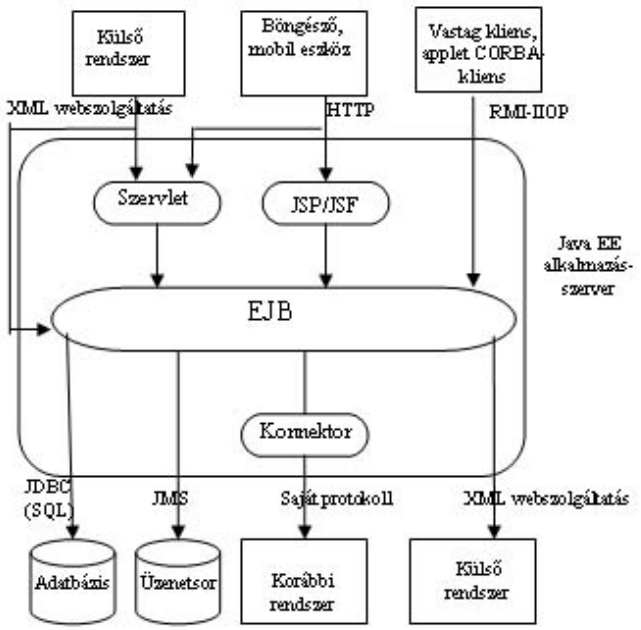
Üzleti logikai réteg: biztosítja a konkrét alkalmazási terület igényeinek megfelelő funkcionalitást, úgy hogy az üzleti szabályok figyelembevételével hívja meg az adatréteg szolgáltatásait (ez kerül a Java EE alkalmazásszerverére)

Kliensréteg: biztosítja az alkalmazás felhasználói felületeit. Alapvetően két fajta klienst különböztetünk meg. *Vastag kliens* lesz a hagyományos desktop alkalmazás, *vékony kliens* pedig a böngésző, amely webes alkalmazások estén biztosítja a letöltött oldalak megjelenítését.

Webréteg: (vékony kliens használata esetén van rá szükség) a böngészőktől érkező http-kéréseket értelmezi, meghívja a megfelelő üzleti logikát és megfelelő választ generál.

Három-rétegű architektúrának szokást azt nevezni, mikor adatrétegre, üzleti logikai rétegre és kliensrétegre osztjuk fel a szoftverarchitektúrát.

Java EE környezetben fejleszthető szoftverkomponensek és azok közül alkalmazásszerverre való telepítésük, illetve a szerverhez kapcsolódó egyéb rendszerekkel való kommunikációjukat tökéletesen szemlélteti a **(4. ábra)**.



4. ábra A Java EE alkalmazásszerver és a hozzá kapcsolódó rendszerek

Ezen komponenseknek van néhány közös jellemzőjük:

- Minden komponens konténerben fut, mely futási időben biztosítja a middleware szolgáltatásokat.
- Az alkalmazásszervert web- és Enterprise JavaBean (EJB[1])-konténerre oszthatjuk fel (kliensoldalon applet- és vastagkliens-konténerre).
- A komponensek lazán csatoltak, így nem szükséges a fejlesztésükhöz a konkrét telepítési környezet ismerete (gyakran interfészeken keresztül érik el egymást).
- Lehetnek elosztottak, ilyenkor távolról is hívhatók, illetve lehetnek helyátlátszóak.
- Újrafelhasználhatóak, alkalmazásszerverek közötti telepítési egységek.

1.3. A megjelenítési réteg

Az első webes anyagok statikus HTML oldalak voltak, melyeket linkek kötöttek össze. Később megjelent az igény arra, hogy dinamikus elemeket tartalmazzanak ezek az oldalak. A kényelmesebb felhasználás miatt ezeken az oldalakon előtérbe került a felhasználói interakció is: dinamikus linkek és űrlapok (HTML beviteli mezők és ezt elküldő gomb) segítségével választhatta ki a látogató az őt érdeklő tartalmat.

A legelső dinamikus oldalak HTML-be ágyazott kiegészítések voltak (Server Side Includes, SSI), melyek lehetővé tették külső parancsok futtatását. Helyüket lassan átvették a script nyelvek, melyek HTML oldalt állítottak elő. Egyre inkább elterjedtté válnak a weboldalak dinamikus generálására képes technológiák (CGI, PHP, ASP).

A jelenleg korszerű webes fejlesztéseknek két legfontosabb célplatformja a Java EE és a .NET. Ezek lefordított kódot futtatnak és áttekinthető, szabványos struktúrájú alkalmazások elkészítését teszik lehetővé.

Az első szerveroldali webes Java technológiát a **Servlet**-ek jelentették. A Servlet-ek mindig egy nagyobb projekt, webalkalmazás részei. Egy általános Servlet segítségével tetszőleges protokoll szerinti kérésekre generálhatunk választ, viszont vannak olyan Sevlet-ek, melyek

[1] EJB: vállalati alkalmazásokban használt szerveroldali komponensek, az üzleti logika implementációját tartalmazzák.

EJB konténer tartalmazhat: perzisztens entitás objektumokat, munkafolyamat komponenseket, üzenetvezérelt komponenseket

kifejezetten csak http-kérések feldolgozására és megválaszolására alkalmasak (Http-Servlet). A Servlet-ek életciklusát kezelő szoftverkomponens a Servlet-konténer. A webszerverrel együttműködve biztosítja a kérések eljutását a Servlet-ekhez, illetve a feldolgozás után a visszajutást. A párhuzamos kérések kiszolgálását lehetővé tevő szálkezelésről is a webkonténer gondoskodik. A Servlet-ek életciklusa három fázisra osztható: inicializáció, kiszolgálás és eltávolítás. Ezen életciklusnak köszönheti a technológia versenyképességét. Csak Servlet-ek alkalmazása a megjelenítő rétegben lehetetlenné teszi a fejlesztések szerep alapú elválasztását (programozó és designer), valamint a karbantartást is megnehezíti, mivel a Java forráskód tartalmazni fogja a HTML részleteket is. Servlet-ek ezen gyengeségeinek kiküszöbölésének érdekében jelent meg a **JavaServer Pages** (JSP) szerveroldali technológia, mely egyszerű szövegfájlként, statikus jelölőelemekkel képes dinamikus tartalmat generálni, ez jellemzően JSP-elemek segítségével történik.
Viszont a Servlet-ek és a JSP oldalak között szoros kapcsolat van, mivel a JSP-oldalak Servlet-ekké fordulnak le. A JSP-oldalak módosításakor automatikusan történik újrafordítás.

JSP technológiához tartozó fejlesztések:

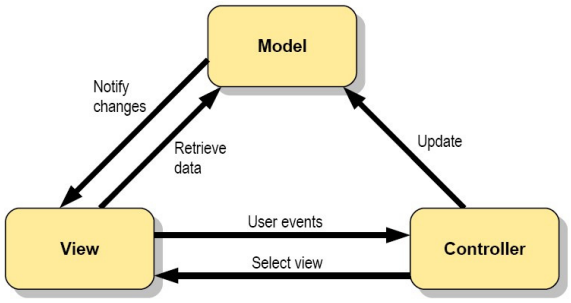
Unified Expression Language (UEL): azzal a céllal jött létre, hogy egyszerű módon lehessen adatokhoz hozzáférni JSP-oldalakon, ezzel alternatívát adva a szkriptletek kifejezései mellé.
JavaServer Pages Standard Tag Library (JSLT): JSTL-akcióelemek formájában megvalósított iterációkkal, feltételes elágazásokkal alakítja a dokumentum struktúráját, a megjelenítéshez nyújt támogatást.

Ezen két technológiára támaszkodva számos webalkalmazás-keretrendszer alakult ki. A Sun Microsystems által létrehozott webalkalmazás-keretrendszer specifikáció a **JavaServer Faces** (JSF).

1.3.1. JavaServer Faces

1.3.1.1. A Model –View – Controller tervezési minta

Több tervezési minta létrejött alkalmazások fejlesztésére. Végül az Model–View–Contoroller (MVC) tervezési minta vált elfogadottá. Az MVC-re sok keretrendszer épül. Ezek közül csak a JavaServer Faces webes keretrendszert szabványosították. A JSF keretrendszer View centrikus keretrendszer. Az **(5. ábra)** bemutatja, mi a kapcsolat a Model, View és Controller között.



5. ábra Model–View–Controller

Model: az üzleti funkciókat nyújtja, állapottal rendelkezik, mely lekérdezhető és módosítható.
View: a felhasználó ezen keresztül látja a Model valamilyen vetületét.
Controller: a felhasználótól érkező kéréseket kezeli, események hatására módosításokat küld a Model irányába, mely a View irányába közli a módosításokat, illetve a View irányába elküldi, mely új nézetet kell megjeleníteni.

1.3.1.2. A JSF alapkoncepciói

Komponensekből épül fel, melyek interfészek segítségével érhetők el. A komponenseket össze kell kapcsolni az adatmodellel, melyet az Expression Language (EL) segítségével tehetünk meg. A navigációt az egyes oldalak között irányított gráfok segítségével írhatjuk le.
A JSF minden egyes pontján az alapértelmezett mechanizmusokat ki lehet cserélni, módosítani lehet.

A komponensek, melyekből a JSF építkezik, saját állapottal rendelkeznek, mely állapotok lementhetők API-n keresztül és vissza is állíthatók. Minden komponens egy önálló felhasználói interakciót reprezentáló egység. A komponensek függetlenek az adatátviteli protokolloktól, illetve a megjelenítéstől. A komponensek élete egy kérés feldolgozásáig tart.
A felhasználó által ismert komponensek: absztrakt komponensek és Render-ek együtteséből áll. A Render-ek kimenetet gyártanak a komponens aktuális állapotából.
A JavaServer Faces a nézetet (View) komponensek fa struktúrájaként ábrázolja. A felépített komponens fát össze kell rendelni a Model-lel, azaz a bean tetszőleges tulajdonságát hozzárendeli a komponens egy tetszőleges tulajdonságához. Ezt a műveletet az EL segítségével tehetjük meg.
A Model reprezentációját Manage Bean-ek (Backing Bean) segítségével adja meg. A Manage Bean nem más, mint egy JavaBean[2], névvel és egy ún. scope[3]-pal ellátva.

1.3.1.3. A JSF kérésfeldolgozó ciklusa

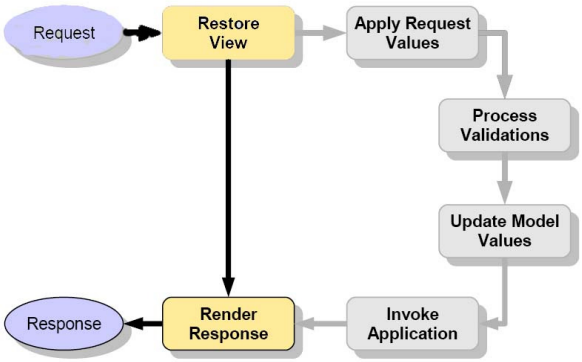
Meg kell különböztetni az első lekérést, illetve a visszaküldés fázisát:

Első lekérés esetén **(6. ábra)**:
Restore View fázisban történik meg az üres komponensfa létrehozása és a viewId beállítása a fa gyökerén. Majd rögtön a *Render Response* fázisra ugrik. Ennek a résznek a teendői: fabejárás három fázisban (encodeBegin, encodeChildren, encodeEnd), ahol is a szabványos komponensek a Render-ek ugyanilyen nevű metódusát hívják.

---

[2] JavaBean: Ez egy Java osztály, mely teljesen egységbe zárt (nincs publikus változója), rendelkezik publikus, paraméterek nélküli konstruktorral és tulajdonságai (bean-property) get- és set metódusok által definiáltak.

[3] <manage-bean-scope>: Azt szabályozza, hogy egy menedzselt osztály példánya mikor és hogyan jöjjön létre, majd mikor kerüljön szemétgyűjtésre. A megadható scope attribútumok lehetnek:
Request: a menedzselt osztály egy példánya a kérésfeldolgozás alatt jön létre és ennek végeztével erre az objektumra tovább nem lesz referencia, szemétgyűjtésre kerül.
Session: a menedzselt-osztálypéldány egészen addig érhető el, amíg a felhasználó sessionje él.
Application: a példányok az alkalmazás teljes futása alatt elérhetők, nem kötődnek felhasználóhoz vagy kéréshez.



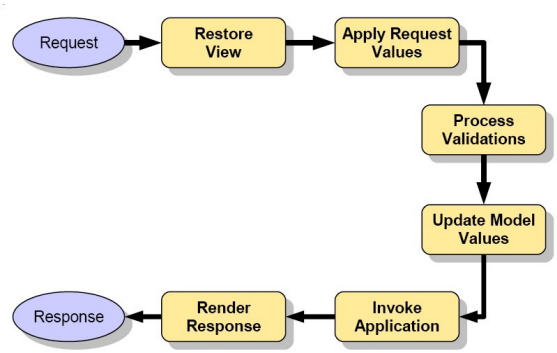
6. ábra Kérésfeldolgozás–Első lekérés esetén

Visszaküldés esetén **(7. ábra)**:

Restore View-ban a komponensfa helyreállítása történik meg, majd az *Apply Request Values*-ban értékek konvertálása megy végbe. Eseménykezelés is ebben a szakaszban folyik, melyhez a JSF a GUI-ban megszokott kezelést biztosít. Az eseménykezelés ezen szakaszon túl, az *Invoke Application* fázisban is lehetséges (az esemény fajtájától függ).

Majd következik a konvertált adatok validálása a *Process Validations* fázisban. Az ezt követő *Update Model Values* szakasz biztosítja, hogy a komponensekhez rendelt értékek a hozzájuk rendelt backing-bean property-k set metódus segítségével beállítódjanak.

Az alkalmazás „action” metódusa és a navigációk beállítása is csak ezek után történik meg az *Invoke Application*-ben. Végezetül, pedig a *Render Response* fázisban kimenet generálódik a kliens felé.



7. ábra Kérésfeldolgozás–Visszaküldés esetén

1.3.2. Facelets

A Facelets valójában a JSF rendszerekre jellemző sablonok hiányosságát pótló technológia. XHTML dokumentumokban adható meg az oldalak váza, valamint azon belül megjelölhetők helyek, és ezekre az oldalt használó JSF adja meg, hogy milyen tartalom kerüljön.

A technológia főbb jellemzői:

A Facelets egy nyílt forráskódú webes keretrendszer és egy alternatív megjelenítés kezelési technológia a JSF mellett. A keretrendszer megköveteli a validált, input XML dokumentummal való munkát. Azaz az összes weboldalt validált XHTML formátumban kell létrehozni. Komoly probléma a JSF és JSP technológia összeillesztése. A kérdés az, hogy hogyan lehet integrálni JSP dinamikus tartalmát JSF komponens központú modelljébe. A JSP a dinamikus output generálására összpontosít, míg JSF megköveteli JSP-től, hogy a komponens modell építését támogassa. A Facelets minden együttműködési problémát kiküszöböl a JSP és JSF megjelenítési technológia között. Biztosít egy séma nyelvet, melyet kibővítettek a JSF komponens modelljével.

Komponens szerelvények létrehozása lehetséges a nyelvben, amelyek közvetlenül beépíthetők egy oldalba, illetve könnyen hozzáadhatók egy facelet tag-könyvtárhoz. Így ezek felhasználásával oldalak sablonját és kisebb részek sablonját határozhatjuk meg.

A JSF összetevők használatához nincs szükség egyedi címkék megírására, mert a Facelets technológia is a JSF egyedi összetevőket használja, miután azokat deklarálták a facelet tag-könyvtárában.

Facelets-nek több pozitív jellemzője van:

- sémák, sablonok
- „composition” összetevők (újrafelhasználhatóságot biztosítják)
- egyedi logikai címkék
- tervező-barát oldalfejlesztés
- komponensekből könyvtár készíthető

1.4. Az adatréteg

A **Java Database Connectivity** (JDBC) egy API a Java programozási nyelvhez, amely az adatbázis hozzáférést támogatja. A JDBC definiálja az adatbázisok lekérdezéséhez és módosításához szükséges osztályokat és metódusokat. A relációs adatmodellhez igazodik. Az API biztosít egy mechanizmust a megfelelő Java csomagok betöltésére és regisztrálására az ún. *Driver Manager*-en keresztül.

A *driver* egy kliensoldali adapter, amely a Java program kéréseit átalakítja az adatbázisszerver által értelmezhető formára.

Miután az adatbázis kapcsolat létrejött, indíthatjuk a tranzakciókat az adatbázis felé.

Az adatbázis kapcsolatot URL formában kell megadni, melynek felépítése a következő:

jdbc:adatbázis_típusa://adatbázis_host[:port]/adatbázis_neve?user=név&password= jelszó

Például: (az általam készített alkalmazás adatbázis kapcsolata)

jdbc:derby://localhost/GPSDB;create=true

Az adatbázis utasítás végrehajtása háromféleképpen történhet, ehhez három interfész adott:

- Statement: Egyszerű SQL utasítás végrehajtására képes.
- PreparedStatement: Olyan utasítások hajthatók végre vele, melyeknek bemeneti értéke is van.
- CallableStatement: Tárolt, ki és bemeneti értékekkel rendelkező SQL utasítások végrehajtását biztosítja. Az adatbázis tárolt eljárásainak futtatására is ez alkalmazható.

1.5. Alkalmazásszerver

Glassfish

Szükség van egy alkalmazásszerverre, mely biztosítja a megírt alkalmazás futtatását és elérését, illetve az adatréteggel való kommunikációt is képes elvégezni.

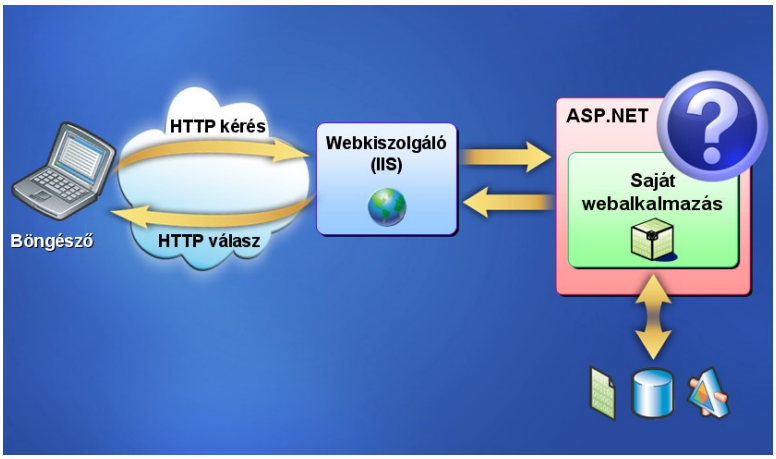
A GlassFish egy Java EE specifikációval kompatibilis alkalmazásszerver, amelyet a Sun Microsystems fejlesztett ki. A kereskedelmi verzió neve Sun Java System Application Server 9.x. A fejlesztése 2005-ben kezdődött az első verzió már 2006-ban meg is jelent. 2007. szeptember 17-én a Glassfish V2 is megjelent, melynek kereskedelmi célú és támogatott megfelelője, a Sun Java System Application Server 9.1.

A Glassfish V2 olyan nagyvállalati funkciókat is kínál, mint az adatreplikáció, a fürtözhetőség és a központi adminisztráció (adminisztrációs felület elérése futó szerver esetén: http://localhost:4848/).

2008-ban a Sun Microsystems bejelentette, hogy elérhető a GlassFish Enterprise Server version 3 Prelude alkalmazásszerver, mely a többrétegű webes alkalmazások éles üzemeltetését célozza meg.

2. .NET környezet

Az n rétegű architektúrát a Java EE mellett a .NET Keretrendszer is használja és támogatja. Csak itt az egyes rétegekben a Microsoft eszközei vannak jelen **(8. ábra)**.



8. ábra Az n rétegű architektúra .NET Keretrendszer esetén

Microsoft a kilencvenes évek végén elindította a Next Generation Windows Services projektet, amelynek eredménye a .NET Keretrendszer lett.

A Microsoft .NET Keretrendszer egy szoftver komponens, melyet a Microsoft Windows operációs rendszer tartalmaz. Az előre leprogramozott megoldások alkotják a keretrendszer Base Class Library elnevezésű gyűjteményét. A .NET Keretrendszerre írt alkalmazások egy Common Language Runtime (CLR) nevű futási környezetben hajtódnak végre, ez kezeli az alkalmazás futás közbeni szükségleteit. A CLR egy virtuális gépet nyújt az alkalmazás számára. A .NET Keretrendszert az osztálykönyvtár és a CLR együtt alkotja.

A CLR négy főbb részből tevődik össze:

- Common Type System (CTS): Definiál minden, a CLR által támogatott lehetséges adattípust és programszerkezetet, és az ezek közötti együttműködések, interakciók miként jét, ezáltal többféle programozási nyelvben végezhető a fejlesztés.

- Common Language Specification (CLS): Azokat a szabályokat írja le, amelyeket a Common Language Infrastructure (CLI[4]) kompatibilis nyelveknek be kell tartania.
- Just-In-Time Compiler (JIT): Fejlesztői nyelveket egy köztes nyelvbe fordítják, majd ezt natív gépi kódra fordítják fordítási időben. Ezt a fordítást végzi a JIT.
- Virtual Execution System (VES): a CLI nyelven megírt programok betöltéséért és végrehajtásáért felel.

2.1. A webkiszolgáló

A Microsoft webes platformjának alapja az Internet Information Services (Internet Információs Szolgáltatások, IIS).

Amikor egy kérés érkezik a kiszolgáló dolga, hogy értelmes módon válaszoljon rá (http-kéréseket a 80-as porton, HTTPS esetén 443-as porton figyeli a beérkező kérést az IIS).

Amikor a böngésző Microsoft-platformon futó kiszolgálónak küld kérést, az IIS fogadja és megkeresi az URL segítségével azonosítható erőforrást. Az IIS saját könyvtár térközét kezelhető darabokra, ún. *virtuális könyvtárak*ra bontja. A virtuális könyvtárak általában egy alkalmazásra hivatkoznak. A Microsoft architektúra Dinamic Link Library (Dinamikus Csatolású Könyvtár, DLL)-ek segítségével válaszol kérésekre. A virtuális könyvtárfájlokat mind leképezik egy-egy DLL-re, így ezek szabják meg az IIS-nek, hogy az adott kérést melyik DLL-nek kell kezelnie. A statikus fájltípusokat (htm) a rendszer közvetlenül az ügyfélnek küldi. A dinamikus lapok további feldolgozást igényelnek, ezért a rendszer meghatározott ISAPI DLL-ekhez rendeli. A webes kéréseket kezelő DLL-ek, az Internet Services Application Programming Interface (Internetszolgáltatások Alkalmazásprogramozási Interfésze, ISAPI) DLL-ek.

[4] CLI: célja, hogy egy fejlesztői nyelvtől független platformot kínáljon a programfejlesztés és a program végrehajtás területén. A Microsoft általi megvalósítása a CLR néven ismert.

2.2. A megjelenítési réteg

2.2.1. A klasszikus ASP

A Microsoft annak érdekében, hogy a webfejlesztést elérhetőbbé tegye Microsoft-platformon kifejlesztette az Active Server Pages-t (aktív kiszolgálóoldalak, ASP).
Az alapja, hogy egyetlen ISAPI DLL az ASP.DLL értelmezi .asp kiterjesztésű fájlokat. Az ASP-fájlok HTML kódot és esetenként szkriptkódot tartalmaznak. A szkriptkódot az ASP ISAPI DLL végrehajtja, és a HTML-t küldi vissza az ügyfélnek. Az ASP jóval szélesebb körben használt programozási nyelveket (pl.: Visual Basic, VBScript) biztosít.

2.2.2. ASP.NET

Az aspnet_isapi.dll az ASP.NET ISAPI DLL-je. Ha a keresett oldal megfelel az ASP.NET valamelyik kiterjesztésének(.ashx, .aspx, .asax) az IIS ehhez a DLL-hez irányítja a kérést.
Amint a kérést az IIS továbbította az aspnet_isapi.dll-nek ez továbbküldi az ASP.NET dolgozófolyamatának (asp_wp.exe).

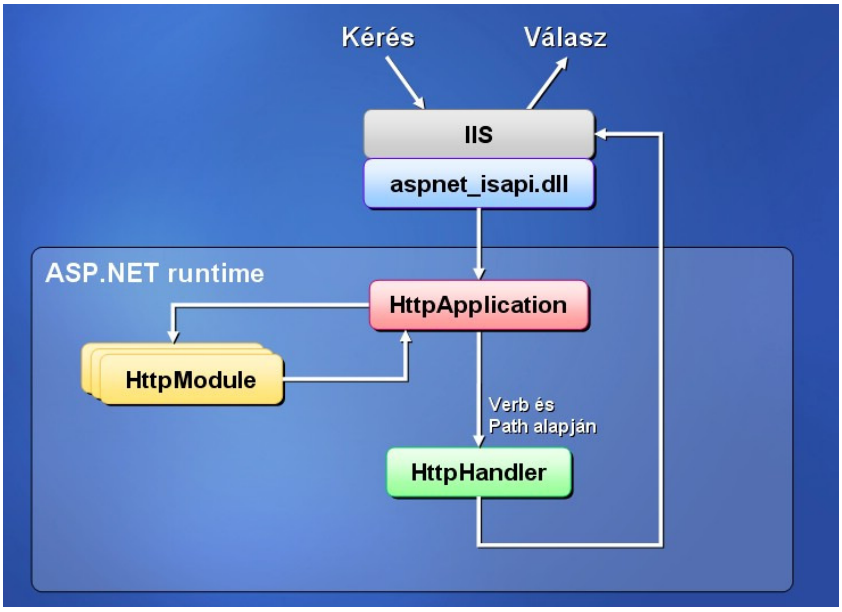
A dolgozófolyamat:

Az ASP.NET dolgozófolyamat kezeli az ASP.NET-futószalagot. Helyettesítő folyamatként működik, mely az ASP.NET történéseit szolgálja ki. Az összes ASP.NET-szoftverösszetevő (HttpApplication objektum és a kérések által meghívott HttpModules modulok és HttpHandler kezelők) ezen helyettesítő folyamat egy példányában fut.

A kérés útvonala ASP.NET esetén (9. ábra):

- IIS-hez megérkezik a kérés
- IIS továbbítja az aspnet_isapi.dll-nek
- az ASP.NET a kérés környezetét a HttpContext egy példányába csomagolja
- az ASP.NET a kérést a HttpApplication objektum egy példányán keresztül a futószalagra helyezi

- ha az alkalmazásobjektumot érdekli bármely kérés-előfeldolgozó esemény azt elküldi neki a HttpApplication (értesítést kapnak az eseményről az erre előfizetett HttpModules-ok is)
- a futtatórendszer létrehoz egy kezelőpéldányt és elvégzi a kérés kezelését



9. ábra A kérések feldolgozása

A Microsoft legfőbb fejlesztése az ASP fejlesztői környezetében, hogy a webes kéréskezelő keretrendszert osztályokból építi fel. A kéréskezelés osztályalapú architektúrába helyezése a fordíthatóságot biztosítja.

Egy oldal életciklusa:

Az oldal életciklusán azt a folyamatot értjük, amely meghatározza, hogy az egyes objektumai mikor születnek, mikor hajtódnak végre az eseményeik, illetve mikor szűnnek meg az objektumok. A folyamatkezelés legelején létrejönnek az objektumok, majd lefutnak a hozzájuk rendelt események. Két fajta esemény lehet: változás, illetve akció. Az események kezelése után végrehajtódik a renderelés (kinézet kialakítás), majd végül pedig megszűnnek az objektumok.

A PostBack egy esemény, amely automatikusan akkor hajtódik végre, amikor egy akció generálódik a kliens oldalon. Ennek az eseménynek az a feladata, hogy küldjön egy újabb kérést a szervernek megadva azt is, hogy mely esemény váltotta ki a PostBack-et.
Változáskor nem történik automatikusan PostBack, ezért ezt nekünk kell beállítani. Az oldal életciklusához köthető eseményeket a **(2. sz. táblázat)** foglalja össze, melyben a Page események mellett az őket felülbíráló metódusok szerepelnek, melyek segítségével az események védhetők.

2. sz. táblázat – Az oldal életciklusához köthető események

Szakasz	**Page esemény**	**Megelőzhető metódus**
Page inicializálás	Init (az oldal vezérlői inicializálódnak)	–
Nézet-státusz betöltés	–	LoadViewState
Visszaküldési adatfeldolgozás	–	LoadPostData metódus bármely vezérlőben, amely megvalósítja az IPostBackDataHandler interfészt
Page betöltés	Load (Page vezérlői betöltődnek)	–
Visszaküldési változásértesítés	–	RaisePostDataChangedEvent metódus bármely vezérlőben, amely megvalósítja az IPostBackDataHandler interfészt
Visszaküldési eseménykezelő	Bármely vezérlő által definiált visszaküldő esemény	RaisePostBackEvent metódus bármely vezérlőben, amely megvalósítja az IPostBackEventHandler interfészt
Page előfordító fázis	PreRender (XHTML kóddá generálás előtti tevékenységek)	–
Nézet-státusz mentés	–	SaveViewState
Page fordítás	–	Render (objektumok XHTML jelölőnyelvre történő fordítása)
Page felszabadítás	Unload (az objektumok itt szűnnek meg)	–

Az ASP.NET már több vele kompatibilis nyelvet támogat. Tehát az eseményeket kezelő hátsó-kódok több különböző nyelven megírhatók (pl.: C#, C++, VB.NET, Java).

2.2.3. Generikus kezelők (az ASHX fájlok)

A HTTP-kezelők egyszerű, IHttpHandler interfészt megvalósító osztályok. A kezelők a Web.config fájlban vannak felsorolva. Eleve regisztrált kezelőket is tartalmaz az ASP.NET, amelyek olyan szolgáltatások megvalósítását biztosítják, mint a nyomon követés vagy az oldal érzékeny fájljaihoz való hozzáférés megelőzése. A kezelőelemek formátuma négy elemet tartalmaz (*add* – tartalmazza a fájlnevet, esetlegesen kiterjesztést, *verb* – parancsszavak listáját tartalmazza, *type* – kérés kezelésére hozzárendelt .NET-típus neve, *validate* – meghatározza, hogy közvetlenül az ASP.NET indításánál betöltődjön-e az osztály vagy kérésre várjon).
A generikus kezelők igény szerint azonnal fordíthatók. Kiterjesztésük: .ashx. Olyan osztályokat tartalmaznak, amelyek az IHttpHandler-t (mindössze a ProcessRequest metódus megvalósítása szerepel benne az aktuális HttpContext-tel és egy IsReusable logikai értéket tartalmazó változó, melyhez tartozik egy get metódus) valósítják meg.

2.3. Az adatréteg

Az ActiveX Data Objects (ADO) a Component Object Model (COM - a .NET előtti fő fejlesztői platform) adatelérési rétege volt. Hidat képezett a programozási nyelv és az adatforrás között, vagyis anélkül írhattunk programot, hogy ismertük volna az adatbázist.
A .NET Keretrendszer felújította, így jött létre az ADO.NET.
Két fajta adatbázis-kezelést különböztetünk meg:

- „hagyományos” mód
- Language Integrated Query (LINQ[5]) család által biztosított lehetőségek

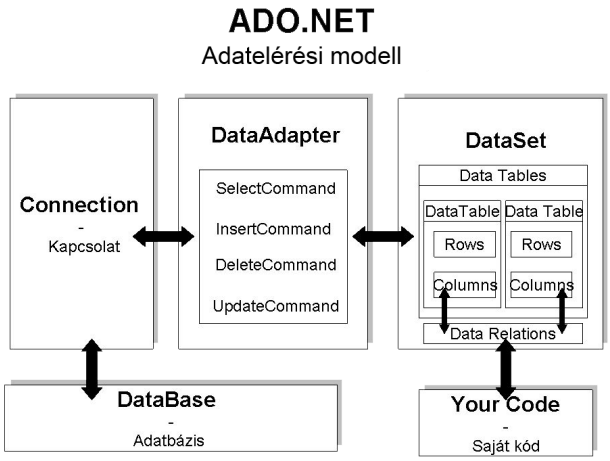
A *„hagyományos” mód*ban az adatbázist ún. Data Provider -eken keresztül érjük el, a lekérdezett adatokat, pedig DataSet objektumokban tároljuk. Így nincs szükség folyamatos

[5] LINQ: közvetlenül a forráskódból a nyelvben szereplő eszközökkel SQL-szerű lekérdezéseket írhatunk. Nemcsak relációs adatbázisból kérdezhetünk le, hanem bármilyen forrásból, amelyre implementálva van (hagyományos objektum vagy listák esetében is). Nem minden rész kapcsolódik az ADO.NET-hez.

kapcsolatra az adatbázisszerverrel. Az ADO.NET is ezt a szétkapcsolt adatarchitektúrát használja.

Kapcsolat nélküli működés:

Létrehozunk az adatbázisból egy másolatot, mellyel dolgozunk, majd legvégül a változtatásokat visszavezetjük az adatbázis szerverre. Az adatbázis másolatot adatkészletnek, vagy DataSet**-**nek fogjuk hívni. Az adatszinkronizációért felelős objektumot, pedig adatillesztőnek, DataAdapter**-**nek nevezzük **(10. ábra)**.



10. ábra ADO.NET – Adatelérési modell

A példány létrehozása:

Létrehozunk egy ún. *connection object*-ot, amely a kapcsolatot jelenti köztünk és az adatforrás között. A kapcsolatot az *SqlConnection* osztály segítségével hozzuk létre, ehhez szükségünk lesz egy ún. connectionstring-re, amely a kapcsolódáshoz szükséges információkat tartalmazza. A connectionstring-ben elsőként a szerver nevét adjuk meg, ahol az adatbázis megtalálható (ha a szerver a saját gépünk, akkor ez a (local) lesz). Majd következik az SQLEXPRESS az SQL Server Express példányának a neve. Ezt követi az adatbázis neve, illetve egyéb kapcsolódási információk (felhasználónév, jelszó…).
Például: (a fejlesztett alkalmazás connectionstring-je)

```
SqlConnectionStringBuilder scsb = new SqlConnectionStringBuilder();
        scsb.DataSource = @"(local)\SQLEXPRESS";
        scsb.InitialCatalog = "WebGPS";
```

3. Java EE és .NET környezet összehasonlítása

Mindkét környezet technológiáiban vannak egyedi csak az adott fejlesztésre jellemző megvalósítások, de számos hasonlóságot is mutatnak.
Minden Java EE szereplő implementálja a saját termékeit a specifikációnak megfelelően. A Java-nak megtalálható sok ingyenes, nyílt forráskódú eszköze. Így az egyes specifikációk kompatibilitása megkérdőjelezhető. Ezzel szemben a .NET esetében nemcsak specifikáció, hanem konkrét termékek állnak rendelkezésre, melyek fejlesztését kizárólag a Microsoft végezte.

A JSF és az ASP.NET összehasonlítása

A JSF a Java EE platformon a webes megjelenítésért felelős keretrendszer, míg az ASP.NET valósítja meg a megjelenítést .NET környezetben.

Komponensek:

A JSF és ASP.NET közös célja a komponens orientáltság és az esemény vezérelt megközelítés.
JSF és ASP.NET weblapok komponensek összeillesztéséből jön létre. Mindegyik keretrendszer szolgáltatja az összetevők egy alapvető készletét, de ezen felül még elérhetőek más egyedi összetevők és új komponens könyvtárak is.

Eseménykezelés:

Az üzleti logikát, mely egy oldal eseményeiért felelős, kódfájlokban fogjuk megvalósítani mindkét technológia esetében. ASP.NET-ben mindegyik weboldalt egy megfelelő .NET osztályfájlhoz társítjuk, amely az ASP.NET Page osztályának leszármazottja. Ezeket a fájlokat hívják „code-behind” fájloknak. JSF-ben minden weboldalhoz társul egy azt támogató JavaBean osztály.
ASP.NET „code-behind” osztályai kezelik a hozzá társított oldalak eseményeit. Viszont az a Java osztály, ami kezeli egy összetevő eseményeit, nem szükségképpen azonos az oldalt támogató JavaBean osztállyal, de ennek ellenére az események kezelésére ez a legmegfelelőbb hely.

Konfigurációs fájlok:

Web.config – web.xml:

Az ASP.NET esetében létezik egy Web.config fájl, mely az alkalmazás indulásakor a megfelelő beállításokat elvégzi, illetve itt tároljuk az adatbázis eléréséhez szükséges connectionstring-et is. JSF esetében is létezik egy WEB-INF könyvtár, melyben az egyes konfigurációs fájlok vannak. ASP.NET Web.config fájljához hasonlóan a Java web.xml létezik, de ebben nincsenek adatbázis csatlakozáshoz szükséges információk, mert azok mind a szerver konfigurációs fájljaiba találhatók meg. A web.xml fájl specifikál egy Java Servlet típusú FacesServlet-et. A FacesServlet felelős azért, hogy elvégezze az alkalmazás használatára vonatkozó konfigurációkat. Egy JSF alkalmazásban minden kérés a FacesServleten keresztül érkezik. Ez a Servlet irányítja az egyes események eredményeit és az oldalak közötti navigációt is.

Faces-config.xml:

ASP.NET-ben az oldalak közötti navigáció ún. Server.Transfer-rel működik, azaz a hivatkozás közvetlen módon megy végbe. A JSF-ben ezzel szemben az egyes események kezelését végző metódusok egy-egy sztring értékkel térnek vissza. A faces-config konfigurációs fájlban definiáljuk, hogy mely sztring érték melyik oldalra mutat. A faces-config.xml-be tároljuk még a „managed beans”-nek nevezett Java objektumokat. Az itt megadott nevével fogjuk tudni később hivatkozni az objektumokat.

Az oldal életciklusa:

ASP.NET és JSF oldalak életciklusa nagyon hasonló. A különbség egyrészt a felhasználó kérésére visszaküldött válasz megjelenítésében van. ASP.NET esetén az egyes komponensek önmagukhoz rendelődve jelennek meg, míg a JSF esetén is történhet ugyanígy, de ott vannak speciális Render objektumok, melyek a RenderKit-ből elérhetőek. Ugyanazt a JSF komponenst különböző webböngészőben vagy kapcsolatnélküli eszközön másik Render objektumhoz rendelhetjük hozzá.
Különbséget mutat másrészt az oldal inicializálásának időpontjában is. ASP.NET esetén, amint a felhasználótól kérés érkezik az adott oldalhoz tartozó Page_Load metódussal

inicializálódik az oldal. JSF esetében ez az osztályhoz tartozó konstruktor hívásával ekvivalens esemény.

4. Térképek és GPS koordináták

4.1. Google Maps

A Google Maps (http://maps.google.com/) a Google által fejlesztett ingyenes online térképszolgáltatás. 2005. október 6-a óta a Google Maps hivatalosan a Google Local név alatt érhető el.
Ahogy a legtöbb Google által fejlesztett alkalmazásnak, ennek a szolgáltatásnak is a JavaScript képezi az alapját.
Funkciói:

- megadott helység, középület megtekintése (nemcsak neve alapján, hanem GPS koordináták alapján is kereshető)
- útvonalkereső (a keresésnél figyelembe veszi, hogy milyen közlekedési eszközzel akarunk menni, néhol elérhetőek a buszjáratok, illetve az útakadályok is)
- utcaképek, műholdképek (Budapesten az utcák fotózását 2009.május 4-én kezdték, de a munkálatok felfüggesztésre kerültek adatvédelmi okok miatt)
- bizonyos országokban a forgalom is megtekinthető

2005 júniusában megjelent a Google Maps API (http://code.google.com/intl/hu-HU/apis/maps/), amellyel a teljes felület személyre szabható. A programozáshoz szükséges fejlesztői kulcs bárki számára ingyenesen regisztrálható.
Az API-ban JavaScript kódok segítségével mutatja be a térképszolgáltatás saját alkalmazásban való felhasználását. A Google Maps API-n túl, a fejlesztésben JSF keretrendszert felhasználók könnyebb előrejutása érdekében létezik a GMaps4JSF.
Előnyei:

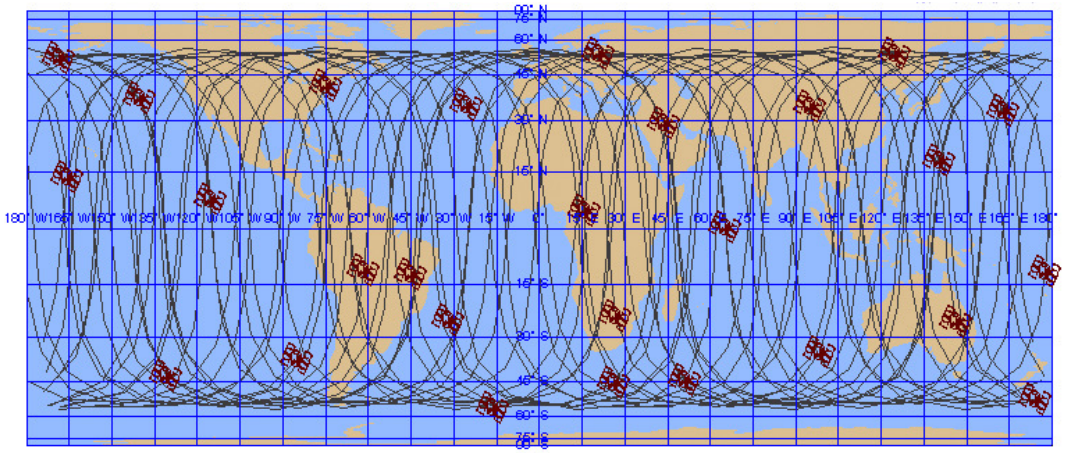
- A GMaps4JSF használatával nem zárjuk ki annak lehetőségét, hogy JavaScriptet alkalmazzunk, mivel minden JSF-tag-nek létezik egy „jsVariable” attribútuma, melynek segítségével JavaScript-ből is elérhető a térkép.
- Támogatja a Facelets-et.

4.2. GPS koordináta

„A GPS (Global Positioning System) egy olyan mesterséges holdakon alapuló navigációs rendszer, amelynek segítségével a földfelszín bármely pontján, vagy a földi légkörben, tetszőleges időpontban, az időjárástól függetlenül, gyorsan, pontosan meghatározhatók a navigációhoz szükséges adatok: a pillanatnyi tartózkodási hely, a pillanatnyi sebesség és az időpont. […]A Föld felszínén vagy az a fölött levő pontok helyzetét általában ellipszoidi földrajzi koordinátákkal adjuk meg. Valamely felületi pontban az ellipszoid felszínére merőleges egyenesnek (normális) az egyenlítő síkjával bezárt szögét nevezzük ellipszoidi földrajzi szélességnek (jele: Φ vagy φ, angol neve: latitude, rövidítése: lat.). A normálison átmenő, és az egyenlítő síkjára merőleges sík és a kezdőmeridián síkja által bezárt szöget a pont ellipszoidi földrajzi hosszúságának nevezzük (jele: Λ vagy λ, angol neve: longitude, rövidítése: long.).”
(2002, Dr. Varga József)

A mérés a geodéziai térbeli ívhátrametszésen alapul az űrben keringő műholdak segítségével. A mérőeszközünk, pedig egy olyan vevőberendezés, amely rádiókapcsolaton keresztül kommunikál a műholdakkal.

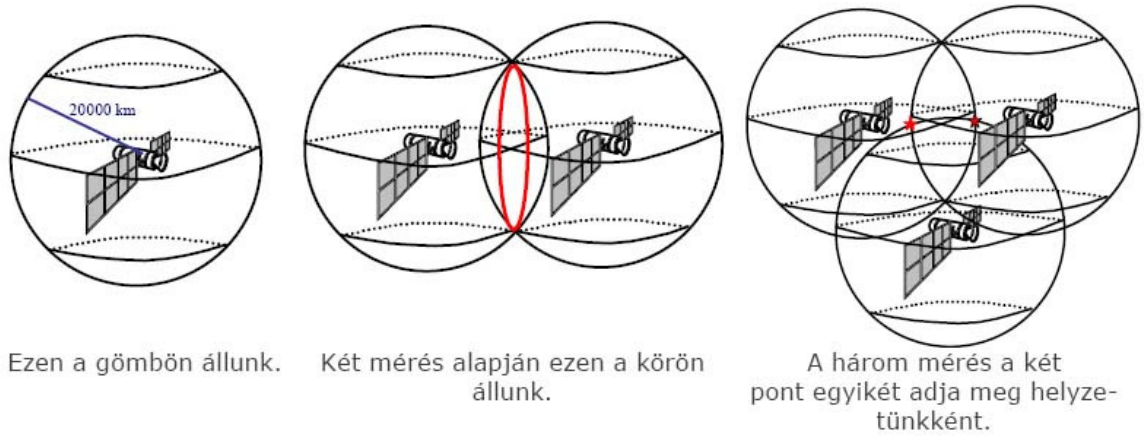
A rendszer 24 műholdból áll, melyek minden nap ugyanazt a pályát járják be a földfelszínhez képest **(11. ábra)**. A hat különböző pályán vannak a műholdak elosztva 60 fokonként (mindegyikre 4 műhold jut). Így a felhasználók a Föld bármely pontján mindig egyszerre 5-8 műholdat láthatnak.



11.ábra A műholdak elhelyezkedése a Föld körül

A GPS pozíció meghatározásának alapja:

Meghatározzuk minimálisan három műholdtól a távolságot. Ekkor az aktuális pozíció egy gömb felületén található. Két gömb metszése egy körvonal, míg három gömbé, pedig két pont **(12. ábra)**. Ezen két pontból az egyik az aktuális elhelyezkedés. Mivel a két pontból az egyik olyan értéket vesz fel, hogy az valahova az űrbe vagy a Föld belsejébe esik.



12. ábra Elhelyezkedés a műholdakhoz képest

A műholdaktól való távolság mérése:

Alapképlet: sebesség * idő = távolság

sebesség: A rádiójel sebessége körülbelül 300000 kilométer másodpercenként.

idő: A rádiójel menetidejéből adódik. Ezt az értéket egy ún. Pszeudo Radom Kód[6] (PRC) vevőműszerhez való érkezéséből határozzák meg, azaz azt az időt mérik le, míg ez a kód a műholdtól a vevőműszerig eljut.

Ezen értékek segítségével meg tudják határozni az aktuális pozíciót. A pozíció meghatározásának pontosításához további lehetőségek vannak, illetve további tevékenységeket is elvégeznek. Például egyszerre négy atomóra alkalmazása a műholdakon vagy a műholdak helyes elhelyezkedésének vizsgálata, melyet a földi állomások végeznek.

[6] PCR: egy digitális kód, 0 és 1 értékek váltakozása. Minden műholdnak van saját egyedi Pszeudo Random Kódja, így az egyes műholdakat azonosítja. Olyan bonyolult a kód, hogy az érkező jel nem egyezik meg más jelekkel.

IV. Eredmények

1. Az alkalmazott szoftverfejlesztési eljárás általános elemei

A kiválasztott technológiák felhasználásával készítettem el egy webes alkalmazást. Mivel alkalmazásról van szó, így egy szoftver fejlesztésének megfelelő fejlesztési folyamatból keletkezik.

Alapvető tevékenységek, melyek minden szoftver fejlesztésében közösek:

1. Szoftverspecifikáció: a szoftver funkcióinak, illetve annak megszorításának definiálása.
2. Szoftvertervezés és implementáció: A specifikációnak megfelelő szoftver előállítása.
3. Szoftvervalidáció: A szoftver verifikálása és validálása, mellyel megbizonyosodunk róla, hogy azt fejlesztettük ki, amit az ügyfél kívánt.
4. Szoftverevolúció: A szoftvernek meg kell felelnie a megrendelő kívánsága szerint történő változtatásoknak.

2. A kifejlesztett alkalmazás fejlesztési folyamata

2.1. Specifikáció

Kettős, egymásra épülő fejlesztési lépésben adható meg, amelyben a követelmények száma bővül.

Első szinten: egy olyan alkalmazást létrehozása volt a cél, mely Java EE, illetve .NET környezetben egyszerű JPG formátumú térképen meg tud jeleníteni néhány GPS koordinátát. Mivel nem volt konkrét koordináta-halmaz megadva, így véletlen szám generálással határoztam meg a koordinátákat.

Második szinten: a Google Maps által biztosított térképeken való koordináták megjelenítésével bővült a cél.

Végleges követelmény, azaz az alkalmazás pontos funkcióinak meghatározása:

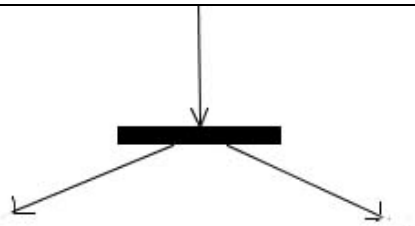
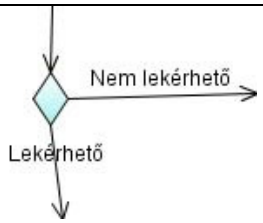
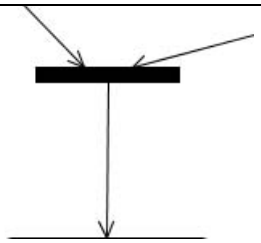
- Az alkalmazás használója bárki lehet, azaz nem szükséges előzetes regisztráció, illetve bejelentkezés a rendszerbe.
- 10 különböző rendszámú gépjármű GPS koordinátáinak megjelenítése 3 különböző térképen (2 darab Föld térkép, 1 darab Debrecen térkép).
- Majd ugyanezen koordináták megjelenítése Google Maps térképein, illetve az itt megjelenített gépjárművek pontos azonosítása rendszámuk és GPS koordinátáik alapján.

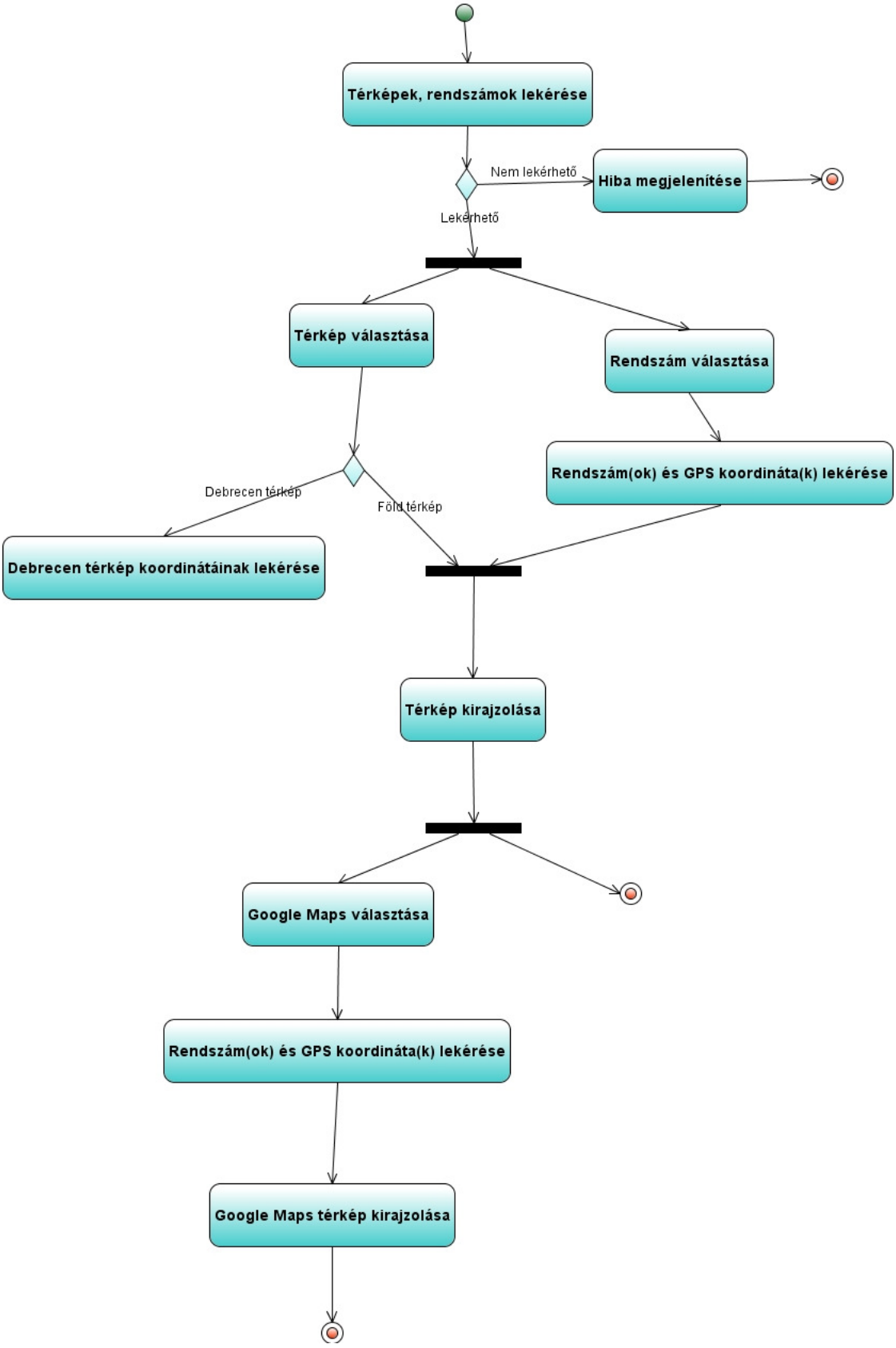
2.2. Tervezés és implementáció

Az alkalmazás követelményeinek megvalósítását, illetve az ehhez kapcsolódó algoritmusok, osztályok megtervezéséhez az Unified Modeling Language (UML[7]) a diagramjai által nyújt segítséget. A specifikációban definiáltam, hogy a felhasználóknak nincsen különböző jogosultságokkal rendelkező csoportja, csak minden joggal rendelkező felhasználó létezik.
A specifikáció megadásánál az UML diagramjai közül a használati eset (Use Case) diagram segítségével adhattam volna meg, hogy pontosan mit tehet egy felhasználó, mire használhatja a rendszert. A rendszer funkcióinak meghatározásában nyújt segítséget. Mivel a felhasználói tevékenység kismértékű, így helyette a rendszer működését leíró aktivitás diagramot (Activity Diagram) használtam. Az aktivitás diagram modellezi az egymás utáni és párhuzamos tevékenységeket a rendszerben.
A diagram segítségével pontosan meg tudtam határozni, hogy milyen tevékenységek folyhatnak párhuzamosan, feltétel nélküli elágaztatást hol alkalmazzak, illetve feltételes elágazások hol szerepeljenek **(3. sz. táblázat)**. Az alkalmazást ezen összetevők alapján megrajzolva a **(13. ábra)**-n látjuk.

[7] UML (Egységesített Modellező Nyelv): egy grafikus modellező nyelv a szoftver-rendszer különböző nézeteinek modellezésére. Segítségével tervezni és dokumentálni tudjuk a szoftvereket. Az UML-ben modellek és diagramok adhatók meg, különböző nézetekben.

3. sz. táblázat – Az aktivitás diagram alkotói

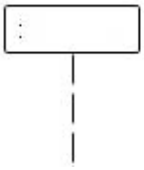
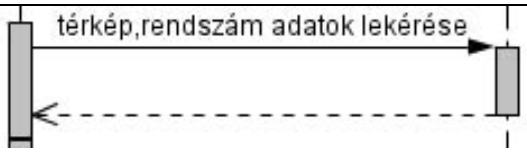
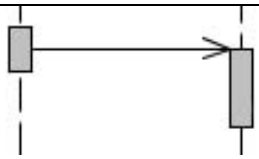
kezdőpont, kiindulási pont	
feltétel nélküli elágaztatás	
feltételes elágaztatás	
szinkronizáció	
tevékenység	
végpont	



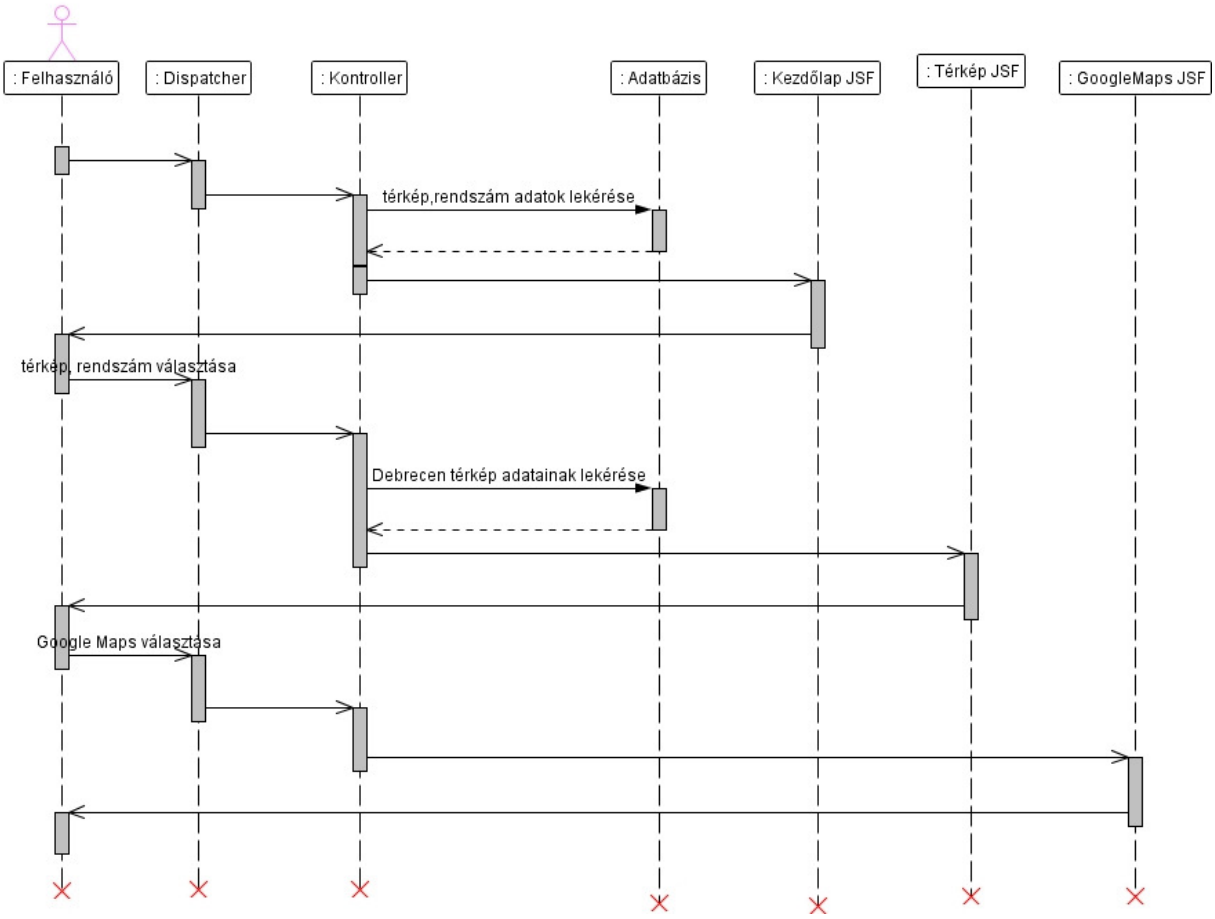
13. ábra Az alkalmazás aktivitás diagramja

Az aktivitás diagramban leírt folyamatok működését és sorrendjét tovább pontosíthatjuk az UML szekvencia diagramjával. A szekvencia diagram (Sequence Diagram) megadja az objektumok közötti interakciót, és itt az interakciók időrendi sorrendje fontos. A diagramot alkotók által kirajzolható, hogy az alkalmazás elindítása után milyen folyamatok zajlanak le a rendszer egyes részei között, illetve mikor aktívak az egyes alkotói. A **(4. sz. táblázat)**-ban felsorolt diagram alkotókkal adhatók meg az alkalmazás működése.

4.sz. táblázat – A szekvencia diagram alkotói

aktor	
a rendszer egyes részei	
szinkron üzenet	
aszinkron üzenet	
aktivitás	
inaktivitás	

A **(14. ábra)** ezen alkotók felhasználásával pontos leírást ad, hogy az alkalmazás indítása után milyen rendszer összetevők tevékenysége hatására jelennek meg az egyes JSF oldalak. A *:Dispatcher* a *:Kontoroller* és a *:Felhasználó* között tartja a kapcsolatot, közvetít és esetlegesen adatokat oszt el. A *:Kontoroller* van összeköttetésben az *:Adatbázis*-sal, amely a szükséges adatokat szolgáltatja. A *:Kezdőlap JSF*, a *:Térkép JSF* és a *:Google Maps JSF* a konkrét megjelenítési oldalakat jelentik. Azok a lapok, amelyek a felhasználó egy-egy választása után a böngészőben megjelennek.



14. ábra Az alkalmazás szekvencia diagramja

A diagramok segítségével megterveztük a rendszer funkcióit és az egyes interakciók időbeli lefolyásának sorrendjét. A tervezés folyamatához tartozik még az adatbázis kialakítása, a táblák és mezőik létrehozása. A két fejlesztői környezetben az adatbázisok felépítése hasonló **(5. sz. – 8. sz. táblázat)**. Néhány mezőtípusban térnek el mindössze. A képek adatbázisban való tárolásánál Java EE környezetben egy ún. BLOB típust használunk, mely segítségével a

képek byte-tömbben tárolódnak. .NET esetén már létezik egy ún. image típus, mely képek tárolását szolgálja. Ebben az esetben is byte-okban tárolódnak a képek. A dőlt betűkkel szedett sorokban az egyes táblák elsődleges kulcsai találhatók meg.
A Cars/Datas táblában az autókra vonatkozó információk vannak, míg a Maps táblában a térképekre vonatkozó információk találhatók meg.

Java EE környezetben a GPSDB adatbázis táblái **(5. sz. – 6. sz. táblázat)**:

5. sz. táblázat – GPSDB adatbázis Cars táblája

Mező neve	**Mező típusa**
CID	integer
Rendszam	*varchar*
Datum	date
GpsKoordX	double
GpsKoordY	double

6. sz. táblázat – GPSDB adatbázis Maps táblája

Mező neve	**Mező típusa**
MID	*integer*
Kep	blob
Kepnev	varchar
NLat	double
NLon	double
SLat	double
SLon	double

.NET környezetben a WebGPS adatbázis táblái **(7. sz. – 8. sz. táblázat)**:

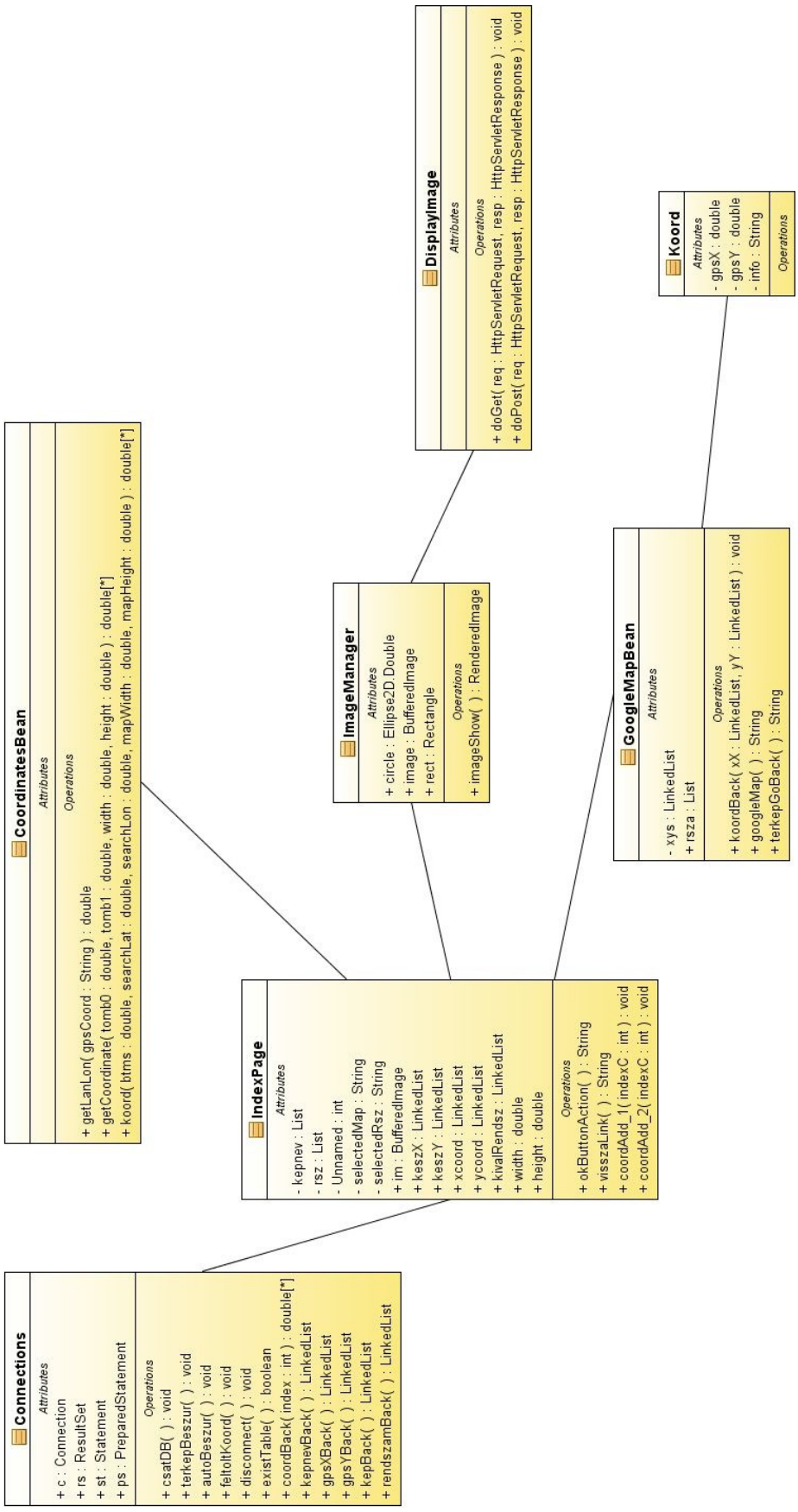
7. sz. táblázat WebGPS adatbázis Datas táblája

Mező neve	**Mező típusa**
Sorszam	int
Rendszam	*varchar(20)*
Datum	datetime
GpsKoordX	float
GpsKoordY	float

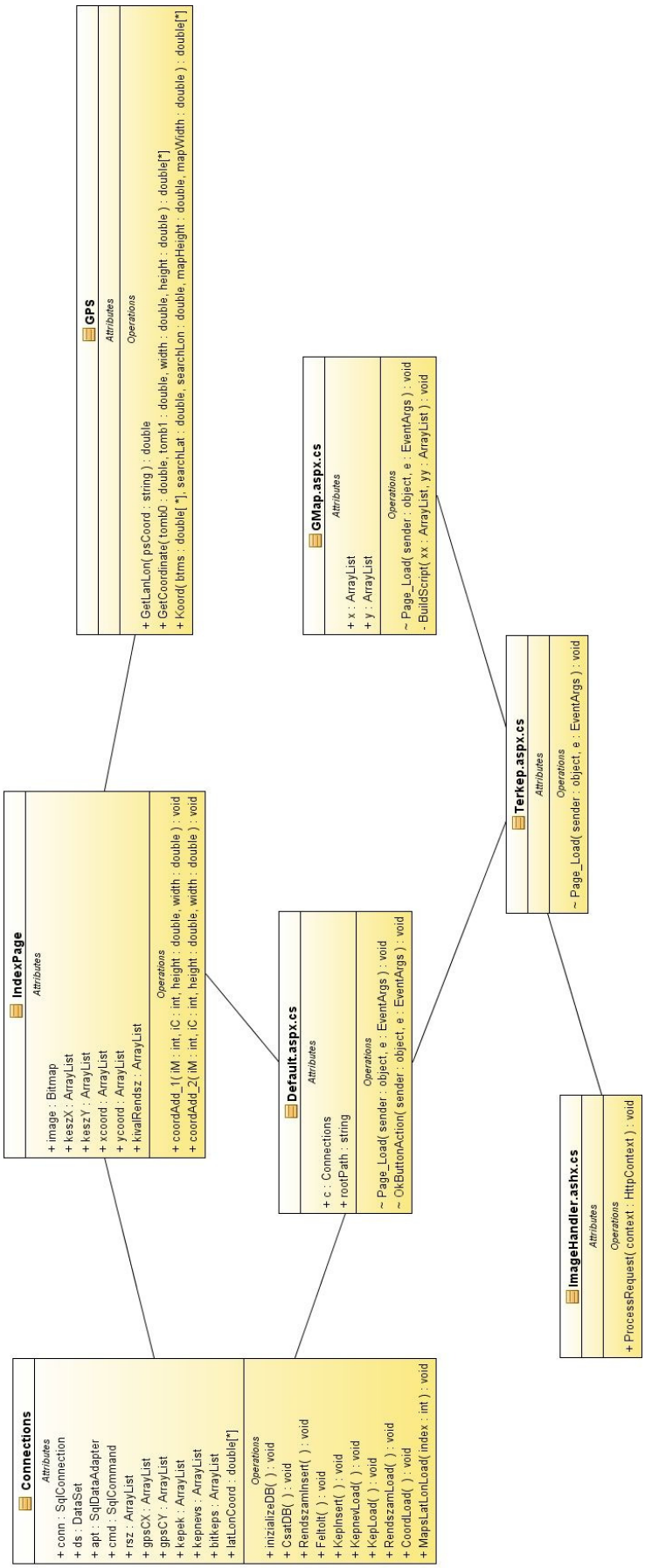
8. sz. táblázat WebGPS adatbázis Maps táblája

Mező neve	**Mező típusa**
ID	*int*
Kepnev	varchar(50)
Kep	image
NLat	real
NLon	real
SLat	real
SLon	real

Az alkalmazás fejlesztésének tervezését az adatbázis táblák létrehozásával fejeztem be. A tervezést az implementáció követi, mely a megrajzolt diagramok alapján történik. Az UML az egyes osztályok implementálásához is biztosít egy diagramtípust. Az osztály diagrammal (Class Diagram) az osztályok, típusok, interfészek és közöttük lévő kapcsolat szemléltethető. Minden összetevő előtt szerepel egy láthatóságot jelző szimbólum (public: +, private: -, protected: ~). A **(15. és 16. ábra)**-n a két környezetben implementált alkalmazás osztályai és kapcsolatuk látható. Az osztály diagramban ábrázolt osztályok attribútum és metódus listája nem teljes, csak a főbb, az adott osztályhoz kapcsolódó osztályok által használt attribútumok, illetve metódusok szerepelnek. Így a segédváltozók, get-, set metódusok és konstruktorok nem láthatók a felsorolásban. Mindkét esetben a Connections osztály implementálja az adatbázis-kapcsolatot, illetve azon metódusokat, melyek az adatokat le- és feltöltik az adatbázisba. A további osztályok mind a megjelenítésért és az egyes események kezeléséért felelősek, illetve a szükséges segédfüggvényeket tartalmazzák. A CoordinatesBean és a GPS osztályok kezelik a GPS koordinátákat, illetve a gépjárművekhez tartozó GPS koordinátáknak térképen való megfeleltetést az osztály metódusai végzik. A Google Maps térképeit a GoogleMapBean, illetve a GMap.aspx.cs osztályok kezelik.



15. ábra Java EE környezetben megírt alkalmazás osztály diagramja



16. ábra .NET környezetben megírt alkalmazás osztály diagramja

Google Maps térképei az alkalmazásban:

A JSF keretrendszerhez a Google biztosít egy GMaps4JSF névű .jar könyvtárat. Ennek segítségével JSF címkék használatával tudjuk a térképet megjeleníteni. A megjelenítéshez nem JSF technológiát használó oldalakhoz a Google Maps API lehetőséget biztosít JavaScripttel történő térkép és összetevőinek a megjelenítésére. A Java EE környezetben a .jar fájl által biztosított lehetőségeket használom ki, míg a .NET platformján JavaScript segítségével jelenítem meg a koordinátákat Google Maps-ben.

GPS koordináták az alkalmazásban:

Az adatbázisban csak Debrecen városnak megfelelő GPS koordináták tárolódnak. Debrecen pontos közepét a 47°31’47.91’’ É 21°38’21.69’’ K koordináták azonosítják. Az egyes koordináták pontos latitude, illetve longitude meghatározására általános képlet:
fok + perc/60 + másodperc/3600
Ez alapján Debrecen esetében:
latitude = 47 + 31/60 + 47.91/3600 = 47.529975
longitude = 21 + 38/60 + 21.69/3600 = 21.63935833

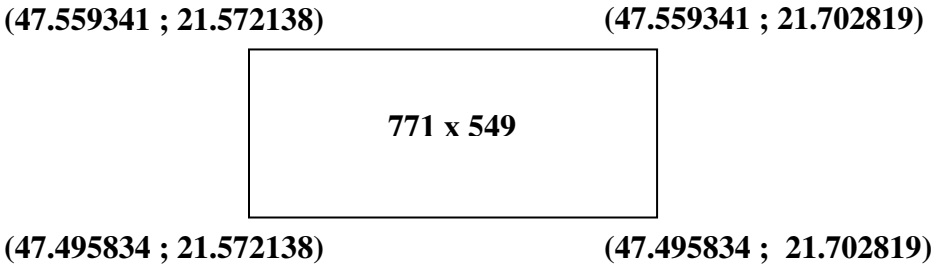
Mivel ezen koordináták térképen való megjelenítéséhez pontosan tudni kell a térkép adott méretét és, hogy azon ezen koordinátájú pont hol helyezkedik el, így további számításokra van szükség. Az alkalmazás két különböző számítást végez attól függően, hogy teljes Föld térképre, vagy az általam megadott Debrecen térképre akarom a helyeknek megfelelő pontokat elhelyezni.
Az általános képlet alapján kiszámított latitude, illetve longitude értékek átszámolása a Föld térképnek megfelelő (x,y) koordinátákra:

x = (longitude + 180) * (térkép_szélessége / 360)

y = ((latitude * (-1)) + 90) * (térkép_magassága / 180)

Ha a megadott Debrecen térképre kerülnek fel a koordinátáknak megfelelő pontok, egy jóval bonyolultabb átalakítási folyamaton kell keresztül menniük. Nem a térkép arányai alapján működik a számítás, hanem a trigonometria függvényeit használva kapjuk meg a megfelelő

(x,y) koordinátát. A számításokhoz szükség van a térkép sarkainak koordinátáira, illetve a térkép szélességére és magasságára.
Így Debrecen térképhez megadott sarok koordináták a következőképpen alakulnak:



Ha a szükséges információkat sikerült összegyűjteni a pontok megjeleníthetők.
A Google Maps API-ja támogatást nyújt a GPS koordináták megjelenítésére, így a megadott latitude, longitude értékek átalakítás nélkül megjeleníthetők a térképen.

2.3. Verifikálás és validálás (V & V)

Az implementációt követően a V & V által bizonyosodunk meg arról, hogy az eredeti követelményeket sikerült-e teljesíteni, illetve ellenőrizni kell, hogy az implementálás során nem követtünk-e el hibákat. A rendszert tesztelni kell, fel kell készíteni minden esemény kezelésére.
Az alkalmazásom esetében az egyetlen hibaüzenetet generáló folyamat, ha az adatbázis elérése nem lehetséges, illetve ha a szerverek nem megfelelően működnek. Ezen események bekövetkezéséről a felhasználót hibaüzenetben tájékoztatom.

3. A fejlesztett alkalmazás

A kétszintű feladatspecifikációnak **(2.1. fejezet)** megfelelően készült el az alkalmazás. Egy áttekinthető megjelenítési felülettel rendelkezik, melyen a felhasználó könnyen tájékozódhat és használhatja az egyes funkciókat. Az alkalmazás alapfunkciójaként gépjárművek pozícióit jeleníti meg Debrecen területén. Rendszámaik alapján választhat a felhasználó megjeleníteni kívánt járművet, majd azt is meghatározhatja, hogy milyen térképen akarja megtekinteni

azokat. A véglegesített követelményekben szereplő Google Maps térképein való koordináták megjelenítését is elvégzi a rendszer.

3.1. Az alkalmazás indítása

Valamilyen web böngészőbe a következő URL-ek begépelését követően indul el:
http://localhost:8080/WebGPSFacelets/ → Java EE környezetben működő alkalmazás
http://localhost:3258/Default.aspx → .NET környezetben működő alkalmazás

3.2. Az alkalmazás felületei és az egyes komponensek funkciói

A két alkalmazásban az egyes oldalak elnevezése és azonosítása **(9. sz. táblázat)**:

9. sz. táblázat – Az alkalmazás oldalai

URL-ben megjelenő fájlnév Java esetén	URL-ben megjelenő fájlnév .NET esetén	Böngésző lapjain megjelenő elnevezés
index.jsf	Default.aspx	Kezdőoldal
terkep.jsf	Terkep.aspx	Térkép
google.jsf	GMap.aspx	Google Maps

Általános kezelő elemek:

Lenyíló lista **(17. és 18. ábra)**:

17. ábra Térképválasztás

18. ábra Rendszámválasztás

Megtalálható az alkalmazás indításakor megjelenő kezdőoldalon.
A lenyíló lista egy-egy térkép, illetve autóra vonatkozó információt tartalmazza (a választható térkép nevét, illetve a választható autó rendszámát). A lefelé mutató nyílra kattintva mutatja meg az adattartalmát **(19. és 20. ábra)**.

19. ábra Választható térképek

20. ábra Választható rendszámok

A listában mozogni egérrel és kurzormozgató nyilak segítségével is lehet. A kívánt érték kiválasztása történhet egérkattintással, vagy az enter leütésével is.

„OK" gomb (21. ábra):

21. ábra „OK" gomb

Az „OK " gomb a kezdőlapon található meg.

Ezen gomb segítségével az adatbázisban tárolt autók GPS koordinátái meghatározódnak, és a lenyíló listákban történt választásunk alapján ezen koordináták a megadott térképen jelenítődik meg a következőekben betöltődésre kerülő lapon.

Lapozási funkciót betöltő feliratok:

Először a Térkép oldalon találkozhatunk velük **(22. ábra)**.

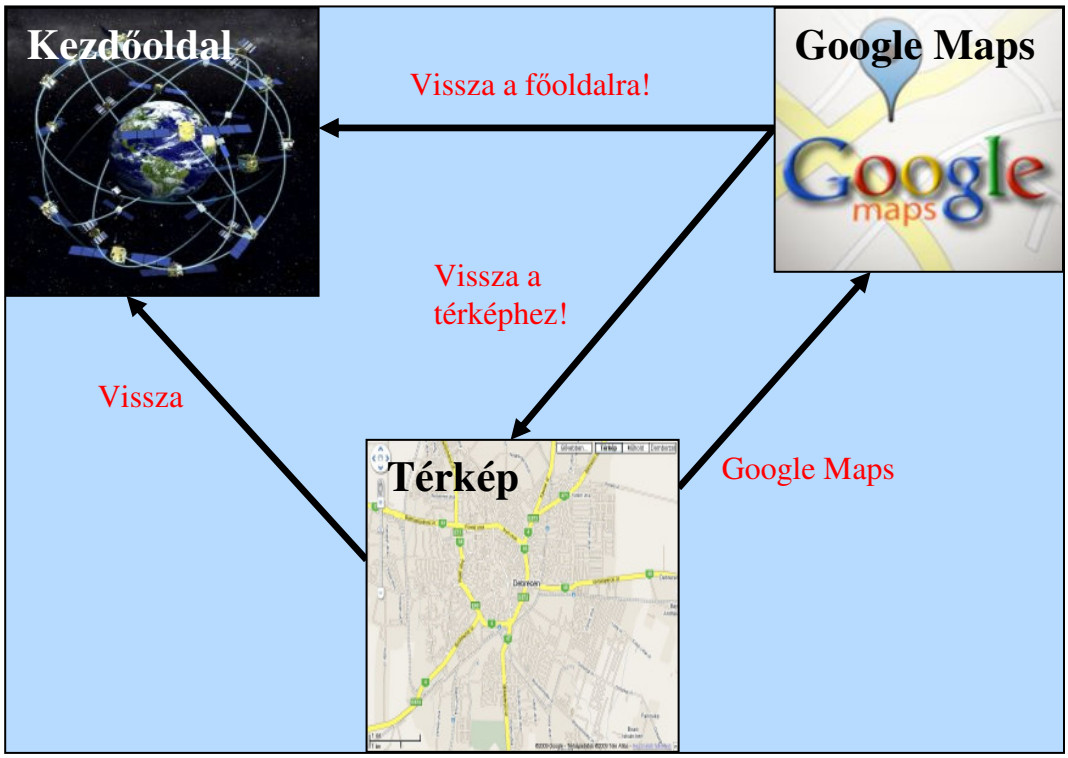
Vissza felirat mindig a kezdőoldalra, míg a *Google Maps* felirat a Google Maps oldalra visz át.

22. ábra A Térkép oldal lapozási funkciói

Lapozási funkcióval bíró feliratok még találhatók a Google Maps oldalon is **(23. ábra)**, melyek a *Vissza a főoldalra!,* illetve *Vissza a térképhez!* felirattal rendelkeznek. Elnevezésük utal funkciójukra, hisz a *Vissza a főoldalra!* feliratú link a Kezdőoldalra, míg a *Vissza a térképhez!* feliratú, pedig a térképet megjelenítő oldalra navigálja vissza a felhasználót.

23. ábra A Google Maps oldal lapozási funkciói

A lapozási funkcionalitású feliratok segítségével bármely oldalról eljuthatunk a kívánt oldalra, illetve megtekinthetjük a korábban már lekért oldalakat újból **(24. ábra)**.



24. ábra Navigációk összefoglalása

3.3. Az alkalmazás működési leírása

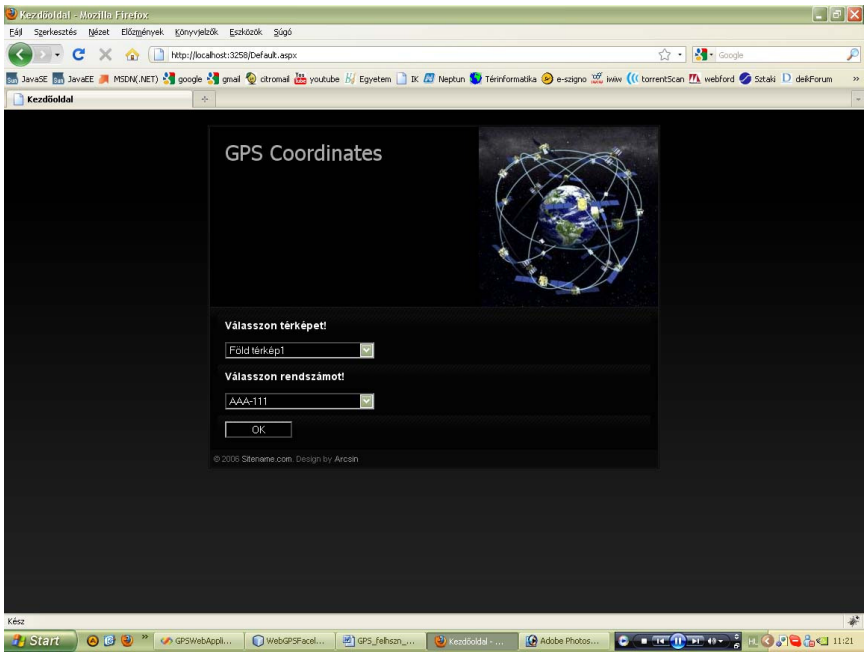
Kezdőoldal **(25. ábra)**:

Az oldalon található 2 lenyíló lista és egy „OK” gomb.

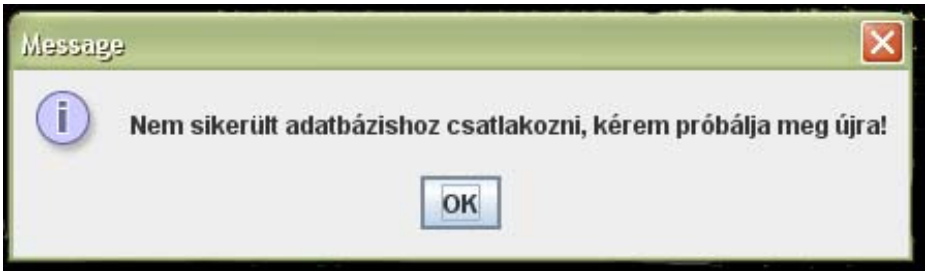
Az első lenyíló listában három térkép megnevezést látunk (Föld térkép 1, Föld térkép 2 és Debrecen térkép). Alapértelmezett érték, amelyet felvesz az a Föld térkép 1-nek megfelelő térkép. Ezen listából választhatjuk ki, hogy az autók elhelyezkedését, mely térképen szeretnénk megtekinteni.

A hasonló kinézetű listában azon gépjárművek rendszámai találhatók, amelyekhez a háttérben futó adatbázisunkban megadott GPS koordináták tartoznak. Ezen rendszámokon felül látható még egy plusz választási lehetőség is: „*Összes rendszám…*”. Ezt választva a letárolt összes autó megjelenik az előzőekben kiválasztott térképen. Alapértelmezetten ezen listában az első rendszám fog szerepelni.

Az „OK” gomb lenyomása után, ha nem megfelelően működnének az alkalmazás futtatásához szükséges szerverek az alkalmazás nem fog elindulni, illetve ha az adatbázishoz való csatlakozás nem lehetséges, akkor viszont hibaüzenet jelenik meg **(26. ábra)**.



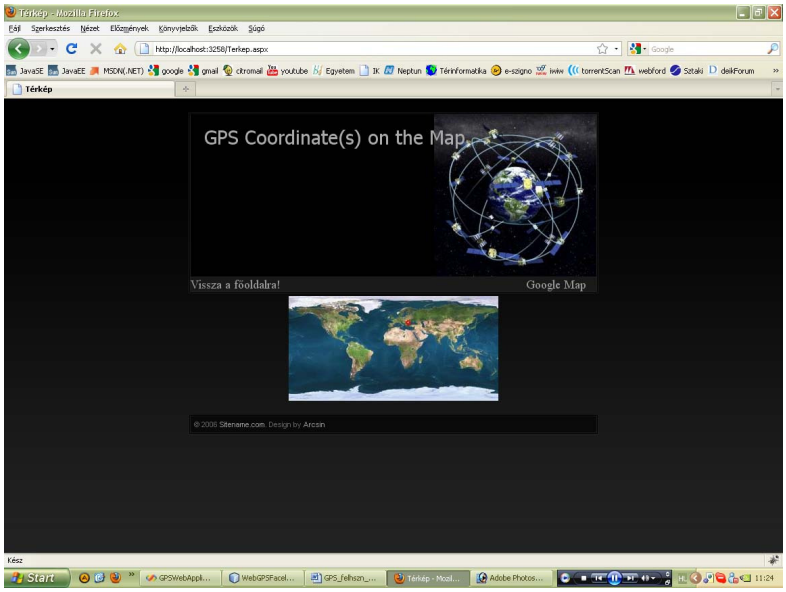
25. ábra Kezdőoldal



26. ábra Hibakezelés

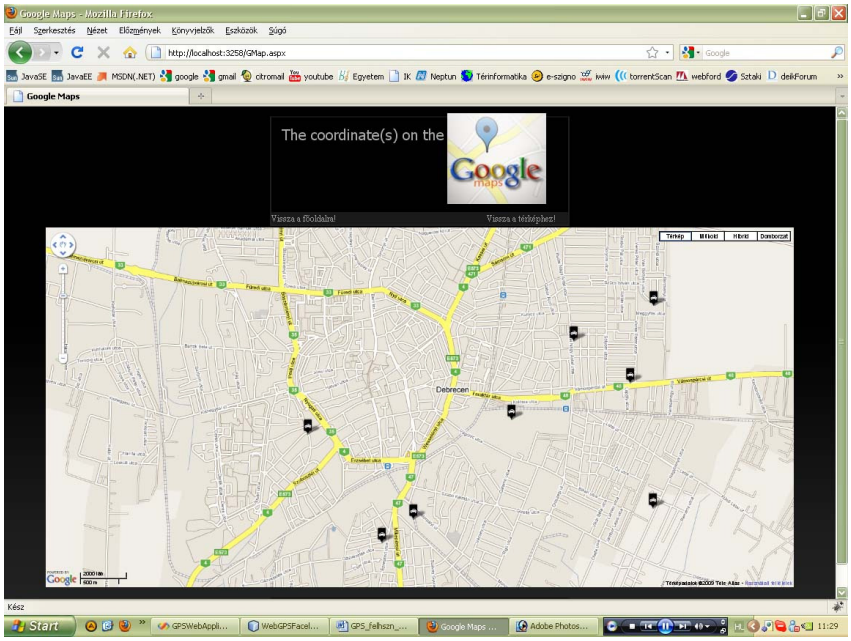
Térkép **(27. ábra)**:

Az „OK” gomb hatására teljesen új oldal töltődik be böngészőbe. A lenyíló listákból kiválasztott térképen az autórendszámnak megfelelő GPS koordináta piros jelölő körrel jelenik meg. A térkép felett szereplő szürke csíkban választhatunk, hogy továbblépünk a következő oldalra és ugyanezen koordinátákkal rendelkező autókat Google Maps térképén is megtekintjük, vagy visszatérünk az előző oldalra, és választunk egy másik térképet és másik autórendszámot.



27. ábra A Térkép oldal

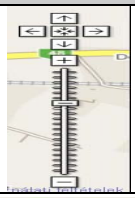
Google Maps **(28. ábra)**:



28. ábra A Google Maps oldal

A Google által biztosított térkép jelenik meg a hozzátartozó funkcionalitásokkal együtt. A térképhez tartozik egy scroll, mely segítségével a térkép zoom alapú nagyítását, illetve kicsinyítését végezhetjük el. Lehetőségünk van a térkép megjelenítési módok közül választani. A két alkalmazásban némi eltérés mutatkozik **(10. sz. táblázat)**: a scroll tekintetében csak kinézetbeli, viszont a megjelenítéseknél az ASP.NET technológiával készült alkalmazásban a *Térkép, Műhold, Hibrid* megjelenítésen túl *Domborzat*-os felülettel is megtekinthető a térkép.

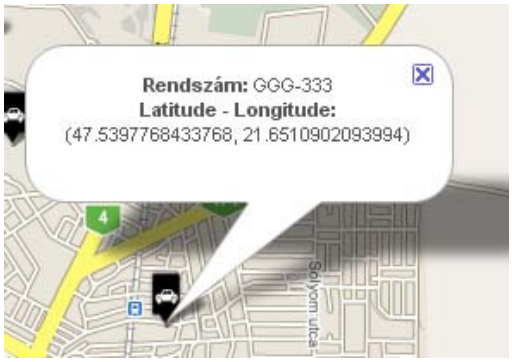
10. sz. táblázat – Google Maps térkép kiegészítői

A két alkalmazás	Megjelenített scroll	Térkép megjelenítési lehetőségek
Java EE		
.NET		

Ezen felületen piros jelölő körök helyén már autót ábrázoló markerek képviselik az adatbázisból betöltött autókat a megfelelő GPS koordinátájú helyeken. 

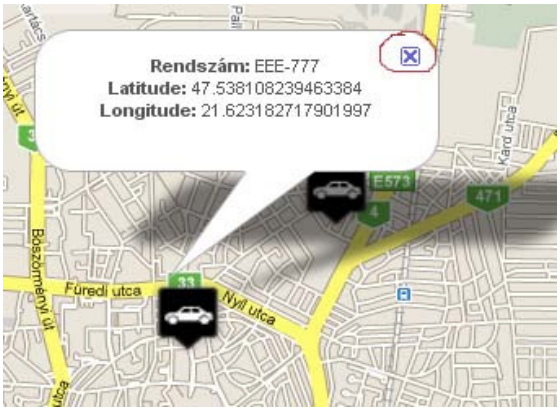
Amennyiben ezen autójelekre kattintunk, egy információs buborék jelenik meg a jel felett **(29. és 30. ábra)**.

29. ábra Java EE információs buborék

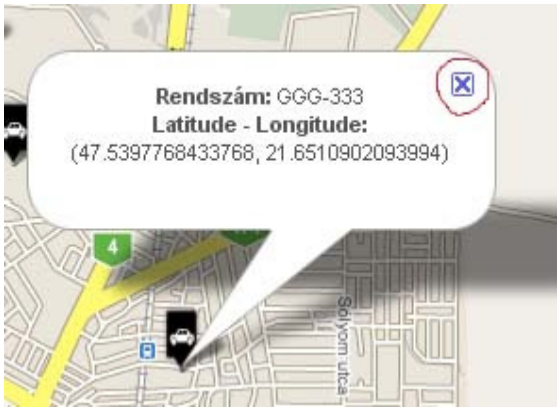


30. ábra .NET információs buborék

Pontos információt nyújt a buborék, hogy mely rendszámú autóról van szó és a GPS koordinátájában található szélességi, illetve hosszúsági koordinátáit adja meg. A jel feletti információs buborékot jobb felső sarkában található kék színű X segítségével zárhatjuk be **(31. és 32. ábra)**.



31. ábra Java EE információs buborék bezárása



32. ábra .NET információs buborék bezárása

Az oldalon hasonlóan az előző oldalhoz 2 navigációt szolgáló link található a térkép felett szereplő szürke csíkban.

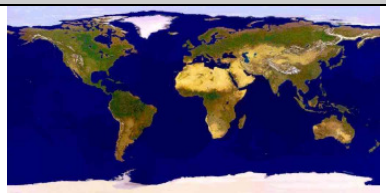
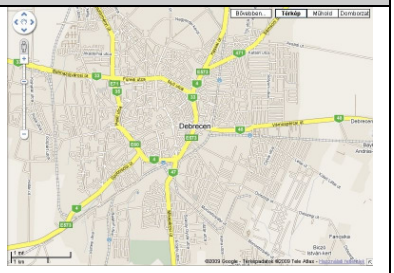
Mindkét alkalmazásban különböző hatást keltenek az oldalak közötti navigációt biztosító feliratok. Ennek oka az, hogy az ASP.NET technológiát használó alkalmazásban a navigáláson túl egy egyszerű frissítés is végbemegy a feliratra kattintva. A gépjárművekhez

tartozó GPS-koordináták minden kezdőoldal irányába mutató navigációval frissítésre kerülnek, így a kiválasztott térképen és a Google által biztosított térképen is a már frissített adatbázisnak megfelelő koordináták jelennek meg.

3.5. A felhasználó által választható elemek összefoglalása

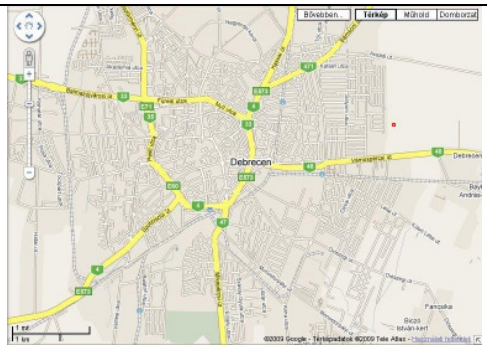
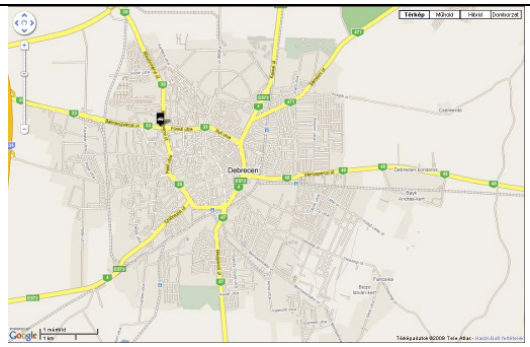
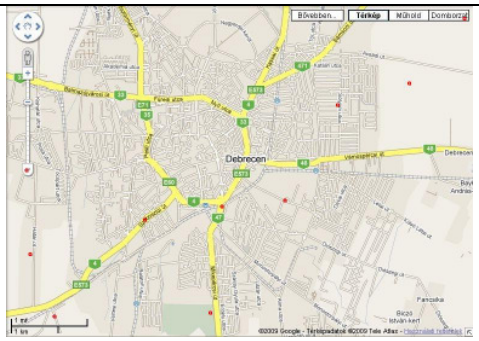
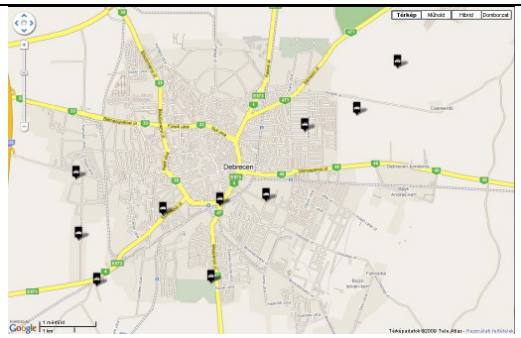
A **(11. sz. táblázat)**-ban azok a JPG formátumú térképek láthatók, amelyeken a választott gépjárművek pozíciói jelennek meg piros jelölő körrel.

11. sz. táblázat – A felhasznált térképek

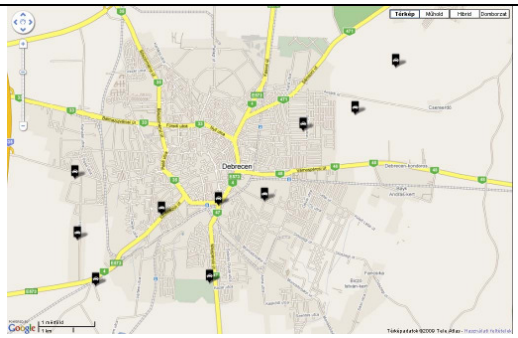
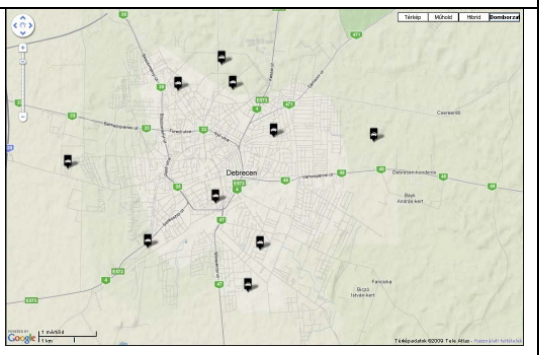
Föld térkép 1	***Föld térkép 2***	***Debrecen térkép***
		

A rendszámokat tartalmazó lenyíló listából amennyiben egy adott rendszámot, vagy ha az *Összes rendszám…* lehetőséget választjuk, a térképeken való megjelenítése nyilvánvalóan különbözőséget mutat **(12. sz. táblázat)**.

12. sz. táblázat – Megjelenített térképek rendszám szerinti összehasonlítása

	Általam megadott Debrecen térképen	**Google által biztosított térképen**
Egy rendszám		
Az összes rendszám		

13. sz. táblázat – Google Maps térképei által biztosított megjelenítési lehetőségek

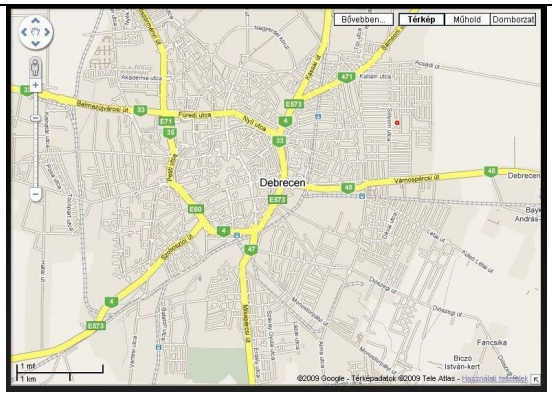
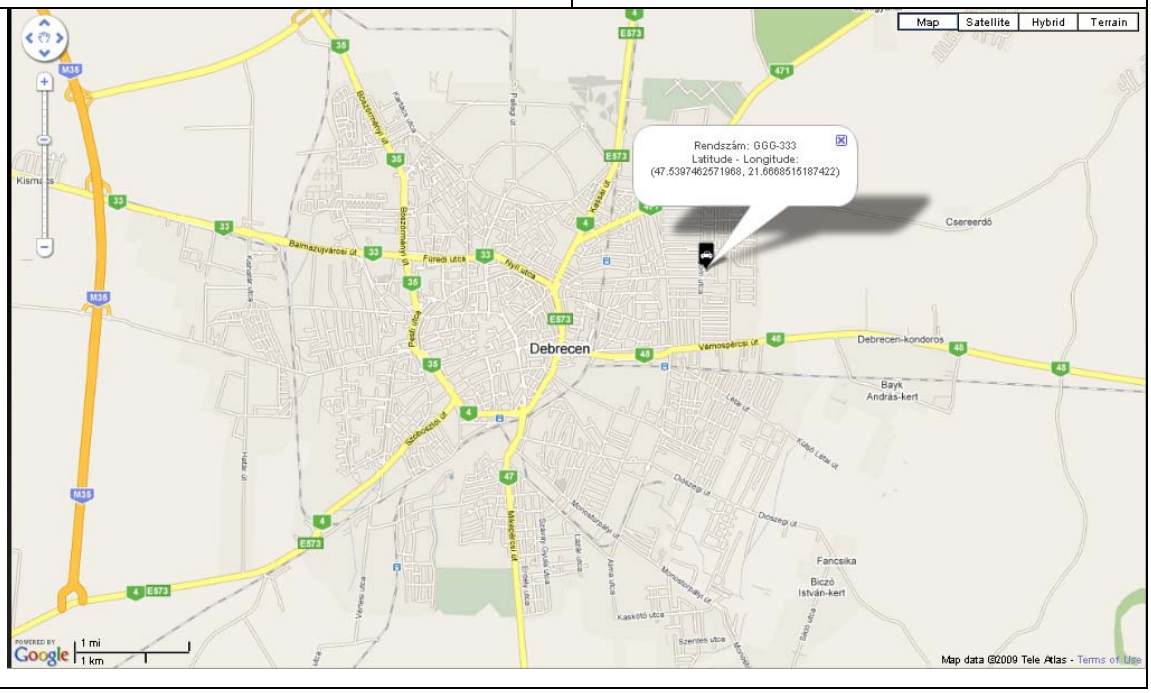
Térkép	
Műhold	
Hibrid	
Domborzat	

3.6. Gyakorlati példa az alkalmazás bemutatására

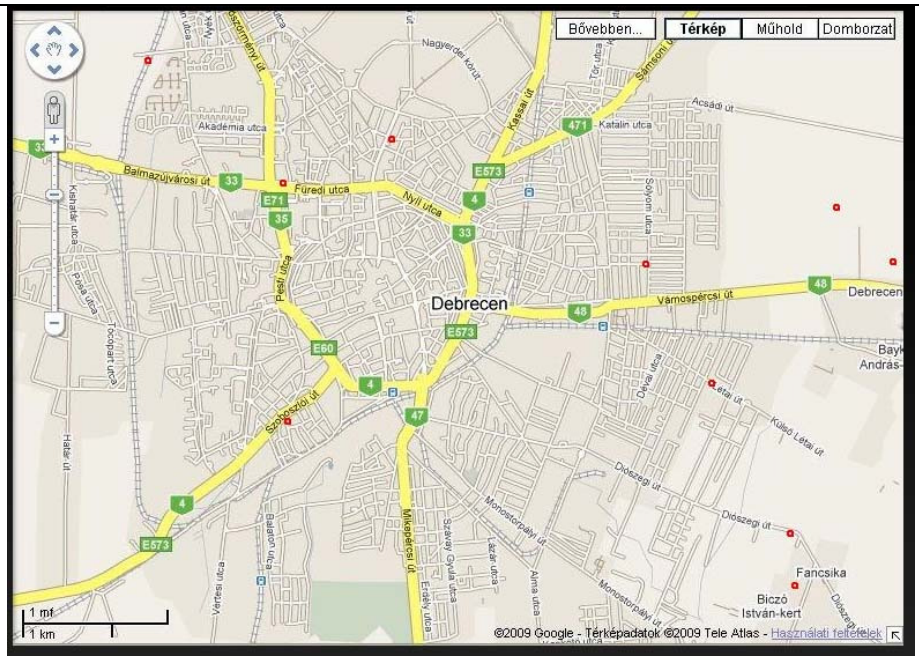
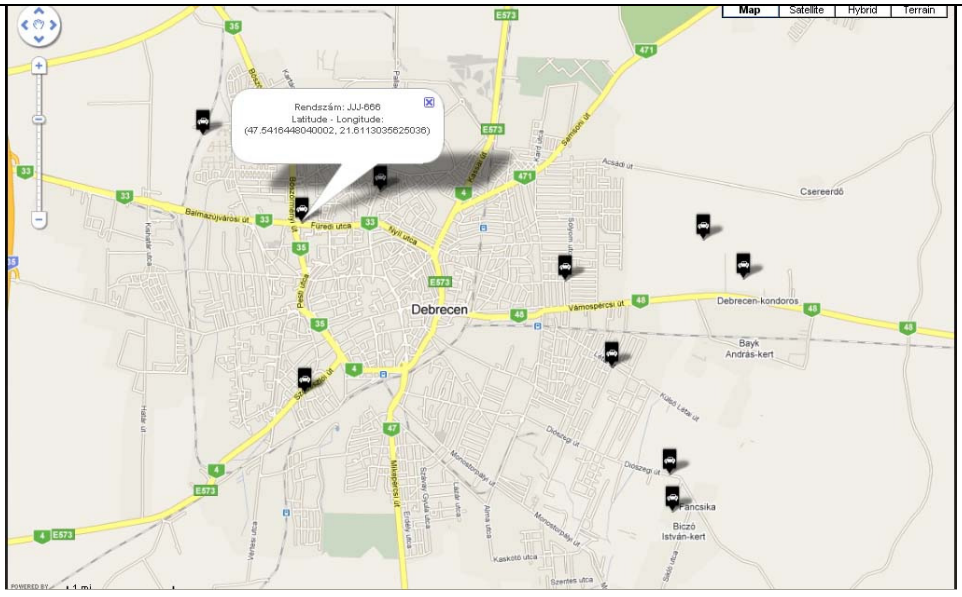
Az alkalmazásban a felhasználó döntéseinek függvényében jelennek meg az információk. Két konkrét esetet mutatok be a **(14. sz. és 15. sz. táblázat)**-ok segítségével. Ekkor a felhasználó először egy Föld térképet választ, és egy rendszámot (GGG-333) szeretne megtekinteni rajta, majd a koordinátákat megjeleníti Google Maps-ben is. Az összehasonlítás érdekében az első példát Debrecen térképen is megjelenítjük, így az adott rendszámot mind a Debrecen térkép mind a Google Maps térképe ugyanott jeleníti meg **(14. sz. táblázat)**. A második esetben Debrecen térképet választ a felhasználó, az *Összes rendszám…* megtekintése mellett dönt, és később a Google Maps-ben is megtekinti a gépjárművek elhelyezkedését **(15. sz. táblázat)**.

14. sz. táblázat – Az első alkalmazási példa leírása

<table>
<tr><th colspan="2">Az első példa</th></tr>
<tr><td>1. lépés</td><td> 


</td></tr>
<tr><td>2. lépés</td><td> 

</td></tr>
<tr><td>3. lépés</td><td> 
</td></tr>
</table>

15. sz. táblázat – A második alkalmazási példa leírása

<table>
<tr><th colspan="2">A második példa</th></tr>
<tr><td>1. lépés</td><td> 
</td></tr>
<tr><td>2. lépés</td><td> 
</td></tr>
<tr><td>3. lépés</td><td> 
</td></tr>
</table>

V. Összefoglalás

Szakdolgozatom keretein belül elkészítettem egy webes alkalmazást, két különböző technológia segítségével, mely az alábbiakat valósítja meg:

- Adatbázisban tárol le gépjárművekre vonatkozó információkat (rendszámokat, GPS koordináták által megadott földrajzi pozíciókat).
- A földrajzi pozíciók Föld és Debrecen térképen való megjelenítése.
- A gépjárművek pozícióinak Google Maps-ben való megjelenítése, illetve az egyes járművekhez tartozó információk lekérdezése.

Az alkalmazás elkészítéséhez használt JSF és ASP.NET technológiákat a fejlesztés során jobban megismertem. Mindkét technológiában további még nem használt lehetőségek vannak. A fejlesztés során Java EE platformon a sokféle létező komponens, kiegészítő technológia, illetve ezeknek a már elkészült rendszerbe való beillesztésük némi problémát okozott, mert külön konfigurációkra volt szükség. Ezzel szemben a .NET platformon minden összetevő együttműködése megoldott, mivel egyazon fejlesztőtől származnak. De a JSF, illetve az azt kiegészítő Facelets technológia esetében a Google Maps által biztosított JSF-címkéket alkalmazó könyvtár könnyítette meg a fejlesztésem.

A rendszer működésbe állítása után további átalakítások, fejlesztések végezhetők el rajta. További funkciókkal bővíthető és akár nagyobb programok részeként újrafelhasználásra kerülhet. Például, egy olyan rendszerbe lehetne integrálni, mely a gépjárművek tilosban való parkolásának megfigyelésére, illetve térbeli eloszlásának feltérképezésére lett kifejlesztve. Ha konkrét alkalmazásként egy ilyen rendszer részeként használnák, akkor szükség lenne az adatbázisban szereplő adatok felviteléhez, felhasználói felületet biztosítani. A felhasznált térképek, illetve a GPS koordináták meghatározása is mind Debrecen területére korlátozódnak. Esetleges továbbfejlesztéssel a kiválasztott terület határai kitolhatóvá válhatnak, illetve globálissá tehetőek. A feltöltött térképek köre is bővíthető, új térképek tölthetők fel, de ehhez is szükség lenne újabb adatbeviteli felületekre. Ezen felületekkel együtt igény lehet a rendszer regisztrációs, illetve bejelentkezési oldalakkal való kiegészítésére. Az oldalak rendszerbe való beillesztésével az egyes felhasználói tevékenységek elvégzését jogosultság alapján korlátozhatják. Google Maps térképein

megjelenített gépjárművek mellett a Google Maps API lehetőséget biztosít a térképet további funkciókkal való kibővítésére, illetve a megjelenítés esztétikai mértékének növelésére.
A két platform által biztosított további technológiák gyorsabbá, megjelenítés szempontjából szebbé tehetik, illetve a használatát megkönnyíthetik.
A lehetséges technológiai továbbfejlesztési irány: az adatbázis kezelését Hibernate vagy EJB-k végezhetnék, így még inkább elválnának egymástól az egyes architektúrális rétegek. A megjelenítésnél a szerveroldali tevékenységek gyorsíthatók, és a megjelenítés dinamikussá válhatna az Ajax technológiájával.

VI. Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani azon személyeknek, akik nélkül ez a szakdolgozat nem valósulhatott volna meg.
Szeretném megköszönni:

- Pajna Sándor, külső konzulensemnek, hogy lehetőséget és témát biztosított a szakdolgozatomhoz.
- Dr. Rutkovszky Edéné, témavezetőmnek, hogy támogatta a téma kifejtését.
- Pázmányi Sándornak a sok segítséget, tanácsot és a rám fordított időt.

VII. Irodalomjegyzék

Bevezetés:

O'Really, Tim: 2006, A Web 2.0 tömör definiálása

http://radar.oreilly.com/archives/2006/12/web_20_compact.html

http://ecdlweb.hu/index.php?title=Internet_-_Az_internet_t%C3%B6rt%C3%A9nete

http://hu.wikipedia.org/wiki/Web_2.0

http://www.origo.hu/techbazis/internet/20060318webketto.html

Anyag és módszer:

Java EE környezet:

Imre Gábor: 2007, Szak Kiadó, Szoftverfejlesztés Java EE platformon

Sun Microsystems: 2009, Java EE Technologies,

http://java.sun.com/javaee/technologies/

http://java.sun.com/javaee/

http://java.sun.com/javaee/javaserverfaces/

http://www.ibm.com/developerworks/java/library/j-facelets/

http://hu.wikipedia.org/wiki/GlassFish

http://hirek.prim.hu/cikk/70121/

.NET környezet:

Shepherd, George: 2006, Szak Kiadó, Microsoft ASP.NET 2.0 lépésről lépésre

Shepherd, George: 2008, Microsoft Press, Microsoft ASP.NET 3.5: Step by Step

http://hu.wikipedia.org/wiki/Microsoft_.NET

http://www.softwareonline.hu/Article/View.aspx?id=4395

http://www.softwareonline.hu/Article/View.aspx?id=2585

http://msportal.hu/blogs/csalap/archive/2007/10/10/18-ado-net-adatel-233-r-233-si-modell.aspx

Java EE környezet és .NET összehasonlítása:

http://www.developer.com/design/article.php/3572721/JavaServer-Faces-and-ASPNET---A-Side-by-Side-Look.htm

http://www.javaworld.com/javaworld/jw-06-2002/jw-0628-j2eevsnet.html?page=1

Térképek és GPS koordináták:

Belényesi Márta; Kristóf Dániel; Magyari Julianna: 2008, Térinformatika elméleti jegyzet, http://www.kti.szie.hu/TTTT/letoltes/terinformatika/elmeleti_jegyzet.pdf

Dr. Varga József: 2002, GPS alapismeretek, http://www.agt.bme.hu/staff_h/varga/publik/publikaciok.htm

http://hu.wikipedia.org/wiki/Google_Maps

Eredmények:

Sommerville, Ian: 2007, Panem Könyvkiadó, Szoftverrendszerek fejlesztése

Ábrajegyzék

Táblázatjegyzék