Szakdolgozat

Formai követelmények:

A szakdolgozatot kemény kötésben kell leadni, 1 példányban.

Fedőlapjának színe fekete, aranyszínű feliratokkal.

Lap: A4-es méret, színe fehér

Betűméret: 12 pont

Sorok: sorkizárt igazítás, 1,5-es sortávolság

Margó:

- belső: 3,5 cm

- külső: 2,5 cm

- alsó: 2,5 cm

- felső: 2,5 cm

Oldalszám: az oldalszámozást a tartalomjegyzéktől kezdve az irodalomjegyzékkel bezárólag folyamatosan kell végezni. Jelölése arab számokkal történik (a tartalomjegyzék oldalainak az oldalszámát nem szokás feltüntetni).

## A dolgozat fő fejezetei (1. szintű címsorok) új oldalon kezdődjenek.

A szakdolgozat a hallgató önálló munkája, melyben be kell tartani a hivatkozások és idézések standard szabályait.

(lapszámozás, címsorok, tartalomjegyzék, stílusok, futó fejléc, ábraszámozás, stb.)

Belső fedőlap

Témabejelentő

Tartalomjegyzék

Tartalomjegyzék

[Oldalszám 1](#_Toc324455965)

[A dolgozat fő fejezetei (1. szintű címsorok) új oldalon kezdődjenek. 1](#_Toc324455966)

[Bevezető 4](#_Toc324455967)

[A Program 4](#_Toc324455968)

[Környezet 4](#_Toc324455969)

[Eszközök 4](#_Toc324455970)

[Hozzáállás 5](#_Toc324455971)

[Kezdeti terv, nehézségek 5](#_Toc324455972)

[Felhasználói dokumentáció 7](#_Toc324455973)

[Bevezető 7](#_Toc324455974)

[Felhasználói esetek 8](#_Toc324455975)

[Futtatási környezet 9](#_Toc324455976)

[Kliens 9](#_Toc324455977)

[Szerver 10](#_Toc324455978)

[Üzembe helyezés 10](#_Toc324455979)

[Rendszerüzenetek 11](#_Toc324455980)

[Fejlesztői dokumentáció 12](#_Toc324455981)

[Megoldási terv 12](#_Toc324455982)

[Rendszer Architektúra 12](#_Toc324455983)

[Adatbázis 13](#_Toc324455984)

[Csomag, Modul és Osztályszerkezet 18](#_Toc324455985)

[Felhasználói Felület 18](#_Toc324455986)

[Megvalósítás 18](#_Toc324455987)

[#Ez mi a fene, cleancode-ról meséljek egy egész fejezetet? Oké... 18](#_Toc324455988)

[Tesztelés 19](#_Toc324455989)

[Junit 19](#_Toc324455990)

[MVP / Database interface / Sycron vs Asyncron hívások 19](#_Toc324455991)

[RDF Adatbázis mérete 19](#_Toc324455992)

[Eeh...? Eredményhelyesség - kihagyós 19](#_Toc324455993)

[Hatékonyság elemzés: GWT Speed Tracer 19](#_Toc324455994)

[Irodalomjegyzék 20](#_Toc324455995)

# Bevezető

A program

A programom egy weboldal, aminek két célja van. Az egyik, hogy bárki kényelmesen találhasson magának és ajánlhasson másoknak utazási célt, és ott található érdekes látnivalókat, szórakozóhelyeket. A másik, hogy népszerűsítse és egyszerűvé tegye az országon belüli utazást.

Az első célt egy térképes kereső biztosítja. Az oldalon egy térképen lehet úticélokat keresni és kinézni. A kinézett utazási célok magyarországi falvak vagy városok, ahova el lehet jutni vonattal. Ezeken az utazási céloknál meg lehet nézni a látnivalókat, szórakozóhelyeket, múzeumokat, és egyéb érdekes helyeket, illetve bárki hozzáadhat egy általa ismert új érdekességet is.

A másik célt a kereséssel összefűzött elvira MÁV menetrendje biztosítja. Az utazás keresésekor időpontot és a vonatokon igénybevehető kedvezményeket is ki lehet választani. Ezekkel az adatokkal az utazással kapcsolatos vonat információkat is megjeleníti a honlap, például a vonatok indulását, érkezését, a jegy árát, indulási vágány számát és egyéb hasznos adatokat mutat meg, minden keresés alkalmával.

Környezet

A feltelepített program az internetről bárhonnan elérhető, és bármelyik korszerű böngészővel használható. A program logikai kódját Java (1.6) nyelven készítettem, a grafikus felületet pedig XML, HTML és CSS nyelveket felhasználva raktam össze. Az elkészült alkalmazás egy war fájlként futtatható egy tetszőleges web-szerveren, például Glassfish vagy Apache Tomcat application server segítségével.

Eszközök

A keretrendszer amiben az alkalmazást fejlesztettem a Google Web Toolkit(GWT)[[1]](#endnote-1), ami roppantul leegyszerűsíti a webes alkalmazások fejlesztését, és elfedi a böngészők közötti különbséget. A keretrendszer biztosítja kliens és a szerver közötti egyszerű kommunikációt RPC[[2]](#endnote-2) segítségével, így ezt választottam az összekapcsolásukra. Sajnos ezt a technológiát nem használhattam ki a MÁV Elvira rendszerének lekérdezéséhez. Az Elvira egy REST API-n keresztül kérdezhető le, így a tőle szükséges adatokat ezen keresztül szereztem be. A program egy saját RDF adatbázist használ az útvonalak és városok meghatározásához, ezt bővíti a felhasználók által megadott adatokkal.

Hozzáállás

A szakdolgozat írásakor nagyon nagy hangsúlyt fektettem a kód minőségére és átláthatóságára. A munkámra nagy hatással volt Robert C. Martin[[3]](#endnote-3) és Martin Flower[[4]](#endnote-4) könyvei, amikben az olvasható, tiszta kód készítését írják le. Igyekeztem a szakdolgozatnak minden részét modulárisan, kicsi és könnyen érthető elemekből felépíteni, és bár nem sikerült mindenhol megvalósítani az elveket, de mindenhol törekedtem a kódom olvashatóságára és szépségére, az öndokumentáló változó, függvény és osztálynevek használatára.

## Kezdeti terv, nehézségek

A szakdolgozat témabejelentőjében az áll, hogy PDFből kinyert adatokat használok fel az alkalmazás RDF adatbázisának létrehozásához. A bejelentő megírásakor ugyan elég határozott elképzelésem volt az alkalmazásról, de nem tudtam sokat arról, hogy ennek megalkotásához milyen adatokra lesz szükségem. Tervezés és a kezdeti prototipizálás közben nyilvánvalóvá vált, hogy a MÁV honlapján található PDF-ek nem tartalmazzák az alkalmazáshoz elengedhetetlen információkat, például a megállók koordinátáit, vagy a közöttük lévő utak leírását, amik a térképes megjelenítést tették volna lehetővé. Így aztán más adatforrást kellett keresnem. A Google térképes tömegközlekedés útvonalainak leírásához létezik egy GTFS(General Transit Feed Specification) nevű formátum, ami szimpla szöveges txt fájlokból, formáját tekintve csv adatbázisfájlokból áll. Ezek ugyan nem a kívánt formában, de minden adatot tartalmaztak arról, amire szükségem volt a térképes megjelenítéshez, a hiányzó adatokat pedig az elvirának egy nem hivatalos REST apijától beszerezve pótoltam ki. Úgy érzem, hogy a váltás nem könnyítette, inkább nehezítette dolgomat a program írása során, mivel több és különböző módon elérhető forrásokból kellett egyedi adatbázist felépítenem. A GTFS és a pdf fájlokat egyaránt egy külső program segítségével(CSVReader illetve IText) alakítottam olyan formára, amit fel tudtam dolgozni, és egyik sem a pontos adatokat tartalmazta; szűrésekre, adatok összekapcsolására és nagy adatmennyiség feldolgozására ugyanúgy lehetőségem nyílt.

Az adatbázis építése során a nagy adathalmaz miatt beleütköztem az adatok feldolgozásának lassúságába, mert az adatbázis első verziója körülbelül 4 óra alatt készült el. Ez természetesen optimalizálásra szorult, és emiatt a feladat ezen részét izgalmassá és roppant szórakoztatóvá tették.

Ugyan tervezetten, de nagyon nagy nehézségekbe ütköztem a technológiák elsajátítása közben. A szakdolgozat megírása és megtervezése során célom volt, hogy minél több új technológiát elsajátítsak elkészítése alatt. Ezt a célt sikeresen elértem. Az írás közben újonnan megismert technológiák között volt a Java, a Google Web Toolkit webes keretrendszer, az Ant build-script megismerése, a Git verziókövető rendszer, az Remote Procedure Call technológia, amivel a szerver és a kliens közötti kommunikációt lehetett könnyedén áthidalni, a REST és a hozzá kapcsolódó elmélet megismerése és ezzel számítógépek magasszintű, http protokollt használó kommunikációjának módjai a hálózaton, a JUnit tesztek megismerése és írása, az RDF adatbázis és a szemantikus web fogalmának megismerése.

A fejlesztés elején a a használt technológiák megismerése, kipróbálása kapott kulcsszerepet. Mivel minden új volt, képtelen lettem volna előre tervezni a programot, így a kezdeti terv szándékosan elnagyolt volt, részletek nélkül. Az útközben felmerült problémákra így mindig akkor reagáltam amikor felmerültek, és nem próbáltam meg kitalálni azt, amihez nem is értek még. Ebben rengeteget segített a Git verziókövető rendszer, ami nélkül a program a kísérletezések közben biztosan darabjaira hullott volna. Az eszközeim használatáról rengeteget olvastam, a velük járó program struktúrák és jó szokásokat pedig igyekeztem útközben magamévá tenni, így alakult ki a szakdolgozat végleges állapota.

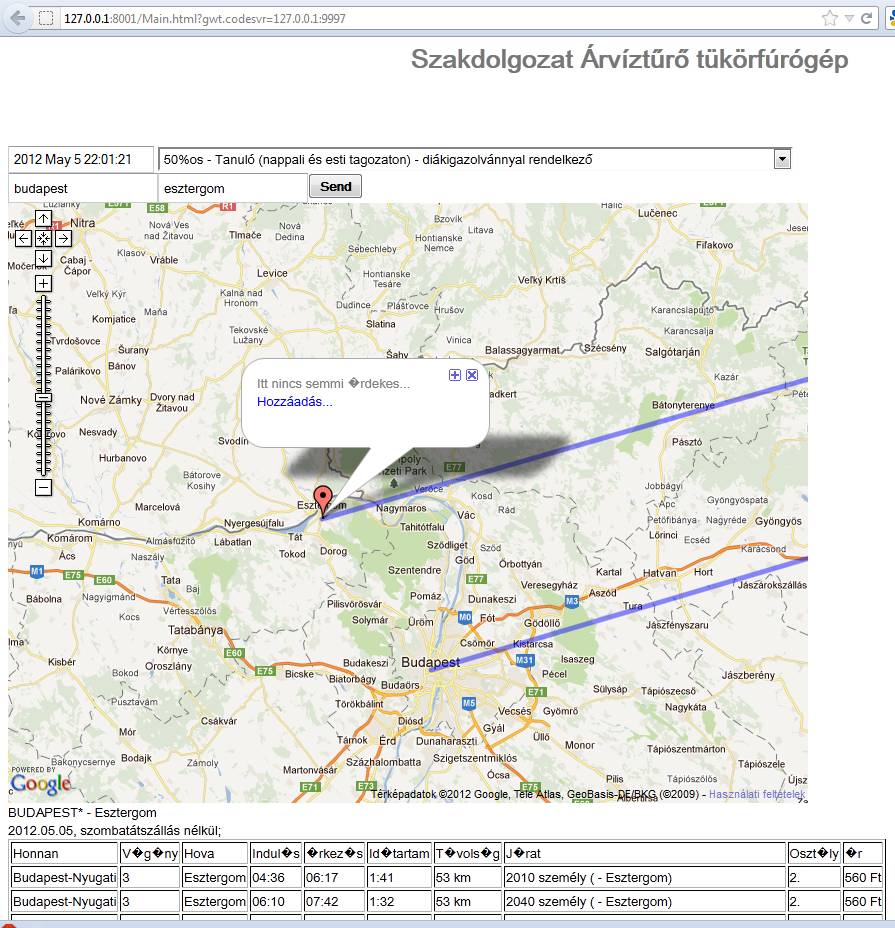
# Felhasználói dokumentáció

Bevezető

Ez a program egy webes alkalmazás ami információkat biztosít olyan kirándulásokhoz, utazásokhoz, ahol az utazók vonattal terveznek menni. A honlapon a célállomások érdekességei közt lehet böngészni, és új érdekességeket hozzáadni.

Mivel a weblapnak szükségtelen a felhasználók adatait eltárolni a feladat ellátásához, a honlap használata regisztrációhoz illetve bejelentkezéshez nem kötött, bárki szabadon használhatja. A széles felhasználói közönség miatt a felhasználói felületet a lehető legegyszerűbbre lett tervezve.

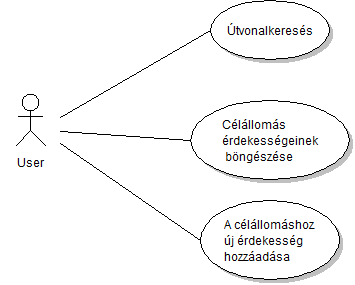
TODO Ide rakni egy olyan képet a programról, amin már nem ilyen ronda.



. Felhasználói felület

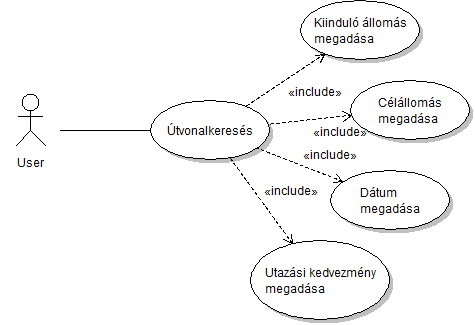
## Felhasználói esetek

A felhasználó alapvetően három interaktív dolgot csinálhat a honlapon, azaz olyasmit, amihez klikkelni vagy gépelni kell:



. Felhasználói esetek

Ebből az útvonalkeresés részletesen úgy történik, hogy a felhasználó kitölti a kereséshez szükséges mezőket. Az „honnan” és „hova” mezőket kötelező kitölteni, különben a keresés nem lehetséges. A dátum és a kedvezmények kiválasztása nem kötelező: ebben az esetben aznapi dátummal és kedvezmény nélkül, teljes árú vonatjeggyel érkeznek meg az adatok az oldalra.

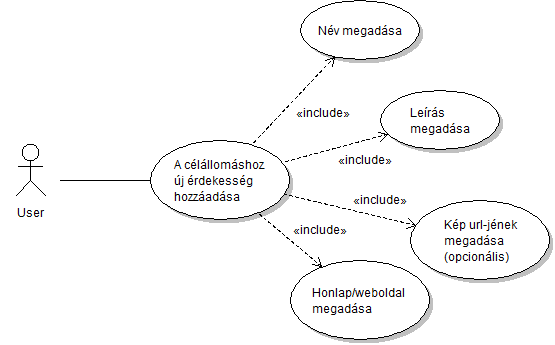


. Útvonalkeresés

Sikeres útvonalkeresés után a térképen megjelenik a vonat által bejárt út. Ennek kezdete a kezdeti, a jelölővel megjelölt vége pedig a célállomás.

A második, azaz „célállomások böngészése” felhasználói esetben a felhasználó a térképen a sikeres útvonalkeresés után a térképen felbukkanó jelölőre kattintva nyithatja meg annak a városnak a listáját, amin a jelölő áll. A felugró ablakban a listában szereplő elemekre kattintva nyithatja meg az egyes helykről szóló rövid leírásokat, valamint a leírásban szereplő linkre kattintva az adott érdekesség honlapját.

Ennek a listának a legalsó kék „Új hely hozzáadása” elemével lehet új érdekes helyet hozzáadni a listához. Ekkor ki kell tölteni a hely nevét, rövid leírását, az érdekesség honlapját, valamint opcionálisan egy neten található kép webcímét megadni.



. Új érdekesség hozzáadása

Amennyiben ez a lista üres, akkor a hozzáadási funkción kívül csak egy felirat látszik, közölve a felhasználóval, hogy „Itt nincsen semmi érdekes...”.

## Futtatási környezet

### Kliens

A weboldal minden modern böngészőben működik, ahol a „modern” a jelenleg letölthető legfrissebb böngészőket jelentik. Ezen kívül a böngészőben engedélyezve kell lennie a javascript kód futtatásának, és mivel honlapról van szó, természetesen szükséges az internet hozzáférés is.

### Szerver

A szerver futtatási környezete egy tetszőleges Web Application Server(WAS)[[5]](#endnote-5), azaz alkalmazásszerver (például Glassfish vagy Tomcat), amire a war kiterjesztésű fájlban lévő alkalmazást fel kell telepíteni. Ennek a telepítésnek a részletei változóak, attól függően, hogy milyen WASra telepítünk. A szerveroldali program, mint minden Java program, egy Java Virtual Machine (JVM)[[6]](#endnote-6)-ban fut, aminek virtuális memória mérete legalább 500 MB méretű kell hogy legyen.

## Üzembe helyezés

A szerver üzembe helyezése három pontban:

1. Fel kell telepíteni az alkalmazásszerverre a programot tartalmazó warfájlt[[7]](#endnote-7)
   1. Ennek menete az alkalmazásszervertől függ, telepítési segédletért annak a dokumentációjából kell ihletet meríteni.
2. Megfelelően fel kell paraméterezni az alkalmazás JVM-jét
   1. A JVM maximális memóriaméretét (és ajánlott a minimálist is) legalább 1000 MB nagyságúra kell állítani. Ehhez a JVM indítási paramétereként meg kell adni a következő parancsokat:  
      -Xmn100M -Xms500M –Xmx1000M
   2. Szintén a program indításakor JVM paramétereként meg kell adni a program konfigurációs fájljának helyét és nevét, relativ vagy abszolút elérési útvonallal:  
      -Dconfig.location=<konfigurációs fájl>  
      pl.: –Dconfig.location=c:\work\Szakdolgozat\config.properites
   3. A program a Java alap temp könyvtárát használja futás közben az ideiglenes fájlok tárolásához. Amennyiben ez nem megfelelő, vagy a programnak nincs joga ezt a könyvtárat használni, egy alternatív helyet kell megadni neki:  
      -Djava.io.tmpdir=<abszolút elérési út a végén per jellel>
3. Megfelelően ki kell tölteni a program konfigurációs fájlt
   1. Ezzel a fájllal lehet beállítani a program különböző szerveren lévő erőforrásainak helyét, például az adatbázis fájl helyét. Ez egy properties fájl[[8]](#endnote-8), amiben jelenleg a következő paraméterek adhatók meg:  
      pricing.properties.file=<a kilistázott kedvezmények helye és neve>  
      rdf.database.file=<az RDF adatbázis helye és neve>

mav.source.zip.url=<a forrásokat tartalmazó gtfs zip fájl URL-je>

példafájl:

\*\*\*config.properties\*\*\*  
1 pricing.properties.file=../resources/pricing.properties  
2 rdf.database.file=/home/matyi/downloads/data.rdf  
3 http://data.flaktack.net/transit/mav/latest/feed/gtfs.zip

## Rendszerüzenetek

A kliensen kétfajta hibaüzenet jöhet elő, információ-jellegű és kritikus.

Az információ-jellegű az oldal részeként jelenik meg, szöveges üzenetként. Például olyan városnév beírásánál ami nem található az adatbázisban, információ-jellegű hibaüzenet jelentkezik, ami jelzi, hogy rossz/nem található a beírt adat. Itt a hibaüzenet közli a teendőket a hiba elhárításához.

A kritikus hibaüzenetek a rendszer belső hibáját, vagy egyes erőforrások hiányát jelzik. Ezek a szervertől „HTTP 500 hibakódú, internal server error” felirattal jelennek meg a honlap bal felső sarkában egy felugró ablakban. A hibák hatékonyabb kiszűrése miatt ezek közül a hibák közül a legtöbb részletes információt is közöl a szerverben keletkezett problémáról. Ilyen hiba lehet például ha az adatbázis nem létezik, ha nem elérhető vagy nincs jogosultsága írni egy szükséges fájlt a szervernek, és egyéb belső hibák.

# Fejlesztői dokumentáció

## Az alkalmazás dióhéjban

Ez a program egy webes alkalmazás amit GWT keretrendszerben Java nyelven készült, de nem kizárólag abban, előfordul HTML, CSS, illetve Javascript kód is. A programból egy war fájl készül, amit egy tetszőleges alkalmazásszerver futtat, és néhány fontos paramétere megadható induláskor egy konfigurációs fájlban. A kliens és a szerver RPC-n kommunikálnak egymással, aszinkron hívásokkal hívja a kliens a szerver által biztosított szervleteket. A kliens egy weblapot jelenít meg, amin le lehet kérdezni és megnézni egy térképen egy belföldi MÁV-os vonatutazás útvonalát, a célállomás környékén lévő érdekes helyeket, valamint információkat(ár,időpont,stb.) az utazásról. A szerver feladatköre, hogy a kapott paraméterek alapján kikeressen adatokat egy RDF adatbázisból a kliensnek, ami tartalmazza az információt a térképen az utak, a városok és a helyek kirajzolásához. Ezen kívül feladata még nem létező adatbázis esetén annak létrehozása egy GTFS specifikációnak megfelelő adatbázisból úgy, hogy letölti ezt a forrás adatbázist tartalmazó zip fájlt, kitömöríti, beolvassa a CSV formátumban lévő adatokat, és legenerálja belőle az RDF adatbázist. A vonatindulással kapcsolatos lekérdezéseknél egy külső REST API-t hív segítségül, tőle kérdezi le a kliens által kért vonatindulás információit.

A programban a fontosabb/rizikósabb, illetve a fejlesztés közben cserélt részeihez Junit tesztek készültek. Elsősorban az adatbáziskezeléssel kapcsolatos szervlet, a szerver és a kliens között utazó osztályok és a REST API-t hívó osztály lett letesztelve.

A program a fejlesztés kezdeti szakaszától verziókövető rendszer alatt van, forráskódja és forráskódja fejlődésének részletes történelme megtalálható a www.github.com/croo/Szakdolgozat cím alatt.

## Felhasznált technológiák

### Nyelvek:

* Java
* Javascript
* HTML
* CSS
* XML
* JSON

### Technológiák:

* **GWT – Google Web Toolkit**
  + A keretrendszer amiben az alkalmazás készült. Lehetővé teszi, hogy a kliens oldali kódot Javascript helyett Java nyelven lehessen megírni, amit aztán a GWT fordítója Javascript nyelvre fordít. A fordító nem csak a végtelenségig kioptimalizálja az írt kódot, de minden böngészőre külön permutációt készít, és később a csatlakozó kliensnek az ő böngészőjéhez illeszkedő kódot küldi el, így biztosítja, hogy a honlap mindenhol ugyanúgy működjön és nézzen ki.
* **REST - Representational state transfer**
  + Ezzel a technológiával készült az Elvira API amitől az alkalmazás egy utazás információit elkéri. Az alkalmazás ezt hívja, egyszerű GET kérésekkel.   
    A REST elsősorban nem egy technológia, hanem egy hozzáállás, hogy hogyan kellene hálózaton a gépek kommunikálniuk egymással. Az ötlet az, hogy ne találják fel a spanyolviaszt, hanem a jól bevált protokollt, a HTTP-t használják információcserére, ami már sokkal jobban megoldotta ezt. Az erőforrások/adatok uri-kon keresztül érhetőek el a HTTP különböző hívástípusaival : GET, POST, UPDATE, DELETE
* **RPC – Remote Call Procedure**
  + Ez egy másik módszer amivel gépek a hálózaton kommunikálhatnak egymással. A szerver és a kliens ennek segítségével beszélget. Ennek lényege, hogy az alkalmazásban minél jobban elfedje ezt a távolságot, lehetővé téve, hogy a kliensből közvetlenül lehessen hívni a szerver egy függvényét.
* **RDF – Resource Description Framework**
  + Speciális adatbázis, amiben az utak és városok megjelenítéséhez szükséges adatait tárolja a szerver. Az RDF a W3C egy szabványa, ami az adatokat gráfként avagy elemek és azok közötti relációkként tárolja.
* **GTFS – Google Transit Feed Specification**
  + Egy szabványos adatbázis formátum, amit a googlemaps fel tud használni. Ebben írhatja le egy közlekedési vállalat az útvonalait és az azokkal kapcsolatos információkat, ami aztán megjelenik a Google térképes szolgáltatásában. Ezek az adatok publikusak és mindenki számára elérhetőek.
* **CSV – Comma-Separated Values**
  + Ahogy a neve is mutatja, vesszővel elválaszott értékek (néha fejléccel). Az egyik legprimitívebb adatbázisfajta. A GTFS specifikációban ilyen táblákban vannak eltárolva az utazás adatai. A táblák formátumai a specifikáció része.
* **Junit**
  + Java osztályok automatikus függvény/osztályszintű tesztelését lehetővé tevő technológia. A kód néhány fontos osztálya illetve olyan interfésze aminek implementációja drasztikusan változott, ezzel lett tesztelve.
* **Verziókövetés**
  + A verziókövető rendszerben a kód készülés közbeni állapotait el lehet menteni, és később visszatérni hozzájuk, ezzel biztonságossá válik a kód változtatása, és nem probléma a kísérletezgetés vagy egy korábbi verzióra visszaállás.

### Külső könyvtárak:

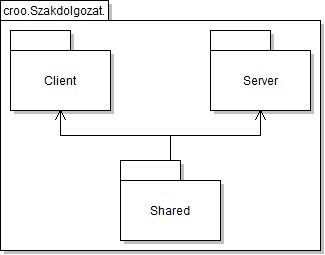
* Junit4
  + Ez a könyvtár biztosítja a Junit tesztelést.
* Apache Jena
  + Az RDF adatbázisbeli lekérdezéseket lehetővé tevő könyvtár.
* GoogleMaps API v2
  + Ennek a könyvtárnak a segítségével jelenik meg a honlapon a térkép illetve az azokon megjelenő funkciók – utak kirajzolása, felugró menük.
* OpenCSV
  + Ez az egyszerű könyvtár megkönnyíti a CSV fájlok beolvasását és feldolgozását Java-ban.
* GWT
  + Robosztus keretrendszer saját fordítóval, és rengeteg beépített webes tecnológiával. Néhány a fontosabb és a kódban használt funkciója:   
    - Java kódból javascript kód generálás  
    - Javaba beilleszthető natív javascript kód támogatással  
    - Grafikus felület összeállítás XML-ben  
    - Egyedi eseménykezelés  
    - RPC kommunikáció a szerver és a kliens kód között
* Resty
  + Kis könyvtár, ami leegyszerűsíti a REST API-k hívását.

## Rendszerarchitektúra

A rendszer három nagy részre, Java csomagokra tagolódik:

* Client
* Server
* Shared

A Client és a Server értelemszerűen a kliens illetve a szerver kódját tartalmazó csomagok. A Shared csomagban azok az adatstruktúrák vannak, amik a hálózaton utaznak a kliens és a szerver között.



. Magasszintű csomagdiagram

Az egyes csomagok szerkezeti felépítésének megértéséhez szükség van két tervezési minta elsajátítására.

### Programtervezési minták a kódban

Két programtervezési mintát használtam a program megépítése során. Az első és legfontosabb az MVP tervezési minta. Ez a minta határozta meg a kód szerkezetét a kliensben. Megértése elengedhetetlen a kód struktúráltságának megértéséhez. A másik minta a Singleton tervezési minta, aminek segítségével elérhetővé tettem a program paraméter-fájl értékeinek lekérdezését az egész kliensben.

#### Model-View-Presenter(MVP) Pattern

Az MVP pattern, azaz programtervezési minta a GWT kliens fejlesztésében kap nagy szerepet. Feladata, hogy elválassza a grafikus felületet a logikától és az adatszerkezetektől. A View részt az alkalmazásban a view csomagban található xml felületleíró fájlok és a hozzájuk tartozó java osztályok alkotják. A view-ben nem található logika, minden olyan feladatot, ami nem a kinézettel kapcsolatos, a saját presenterének delegál. A presenter kommunikálhat a view-el egy display interfészen keresztül, ami biztosítja, hogy a grafikai felület teljesen elváljon az alkalmazás többi részétől, cserélhető és moduláris legyen. A presenter intézi a hívásokat az alsóbb rétegekbe, azaz a model-be, illetve kapja a válaszokat, amiket szükség esetén delegál a view-hez. A modelben az alkalmazáshoz tartozó erőforrásokat és adatokat érhetjük el, ami esetünkben a szerver interfészeit jelenti.

A jól ismert Model View Control(MVC) mintához nagyon hasonló. Eltérésüket az alábbi ábra szemlélteti:



. MVC vs. MVP

Az MVP programtervezési minta nagyobb alkalmazásokban a struktúráltsák mellett a tesztelést segíti elő; a GWT által generált javascript tesztelése ugyanis nehézkes, és a felületi elemek elválasztása lehetővé teszi, hogy a minden szükséges logikát javascriptre fordítás nélkül lehessen letesztelni. A kliens kód szerkezete ezen minta köré lett felépítve. Ugyan a gyakorlatban a grafikát és a logikát nem mindig triviális elválasztani, ez a minta a legtöbb helyen jól biztosítja az olvashatóságot és a rétegek elkülönülését.

Ebben a dokumentációban gyakran előfordul a *modul* kifejezés használata. A kliens egy *modul*ja az elkövetkezendő leírásokban egy view és a hozzá tartozó presentert jelent.

#### Singleton Pattern

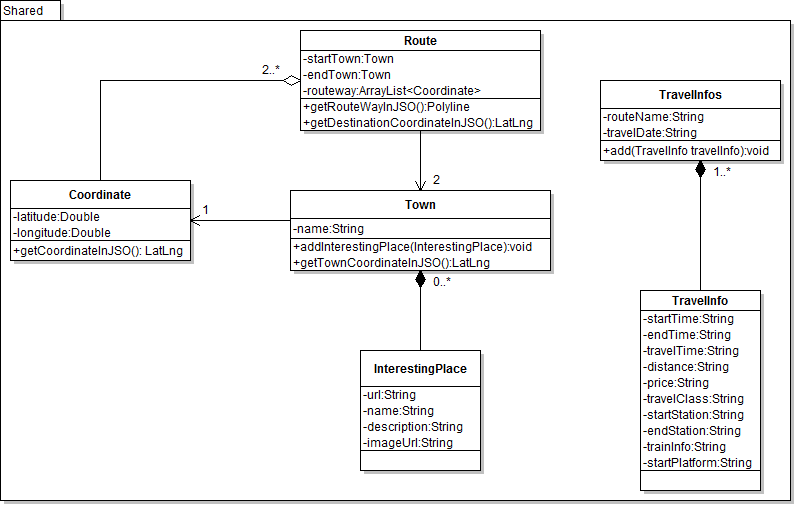
A singleton tervezési minta biztosítja, hogy egy objektumból legfeljebb egy létezzen az egész kódunkban, ezzel biztosítva egy globális változót az objektumorientált nyelvekben. Használata ugyanúgy kerülendő, mint a globális változóé, de természetesen itt is vannak olyan helyzetek, amikor alkalmazásuk hasznos és ajánlott.

A kódban egy singleton található, aminek neve SystemProperties. Ez az osztály olvassa be a kulcs-érték párokat tartalmazó properties fájlt ami felparaméterezi az alkalmazást.

### Csomag, Modul és Osztályszerkezet

#### Shared csomag

Ez a csomag tartalmazza azokat az adatstrutúrákat, amik a szerver és a kliens között utaznak a hálózaton. Mindegyik osztály szerializálható, mivel örökli a GWT által biztosított IsSerializable interfészt. A feltüntetett függvényeken kívül gettereket illetve settereket tartalmaznak.



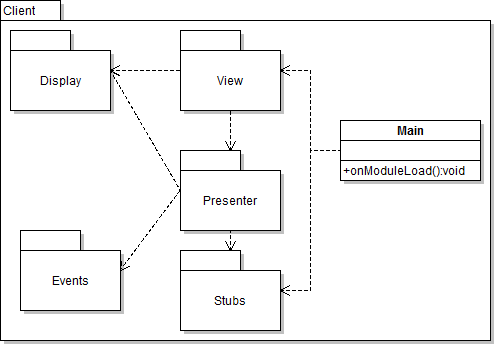
A TravelInfos adatstruktúra tartalmazza az Elvira REST API-tól lekérdezett táblázatot, valamint az utazás nevét és dátumát. A TravelInfo osztály ennek a táblázatnak egy sorát reprezentálja.

Az ábrán négy osztálynál is előfordul a getCoordinateInJSO függvény valamilyen formában. A JSO rövidítés azt jelenti, hogy JavaScripObject. A GoogleMaps API koordináta-osztálya a natív Javascript kódban megírt LatLng osztály, és ezek a függvények egy ilyen objektummá konvertálva adják vissza az adatot. Így megjelenítésnél sokkal kényelmesebb használni az adatstruktúrákat.

#### Client csomag

Ebben a csomagban található az a kód, ami a kliens oldalon, böngészőben fut. Csak a szerverrel áll kapcsolatban, az RPC által biztosított aszinkron hívásokon kereszül kommunikál vele. Ezeknek az iterfészeknek a leírása a Server csomag fejezetben kerül részletesebb kifejtésre.

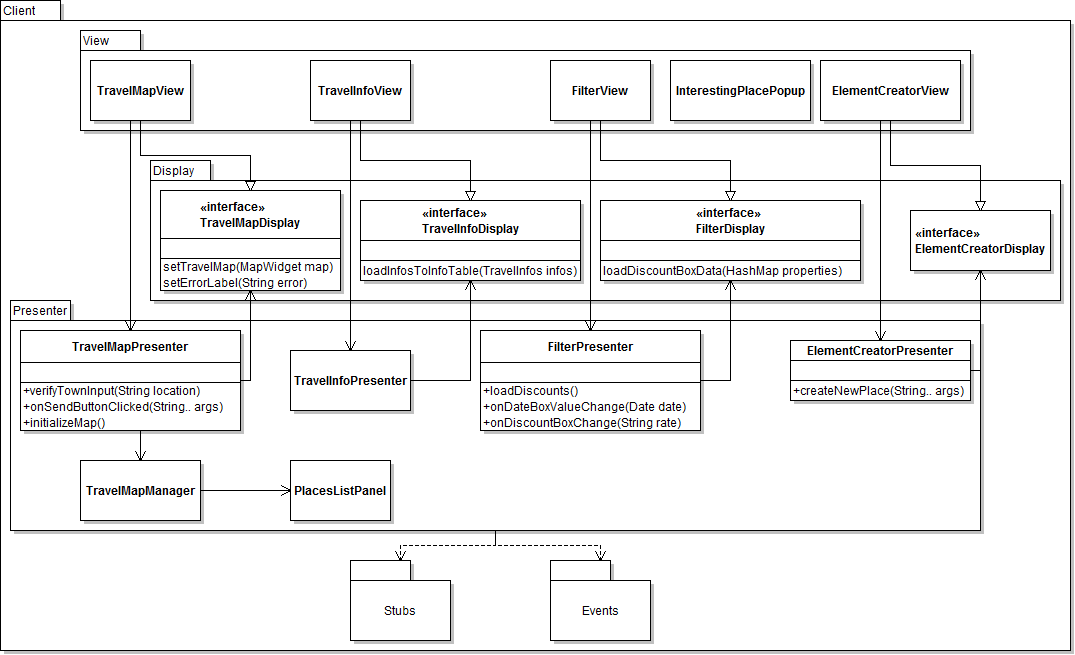
A kliens sok kisebb csomagból épül fel, jól magyarázva kliens különböző részeit:



. Kliens Csomag diagram

TODO:Még egy kicsit mesélni a csomag diagramról, ha lehet...

Az ábrán jól láthatóan különül el a kód az MVP minta szerint:



. Kliens osztálydiagram

A Modellt, az MVP harmadik rétegét itt elsősorban a szerverrel kommunikálás, a Stubs csomag alkotja. Ennek leírása részletesen a következő fejezetben, a „Server csomag és az RPC kommunikáció”-nál található meg.

A View csomagban lévő kódok tartalmazzák az oldalon megjelenő dinamikus elemeket. Mindegyik View a GWT keretrendszer UiBinder technológiája segítségével lett összeállítva. Ez azt jelenti, hogy a grafikus elemek egy XML plusz Java osztály párosok, amik együtt írják le a kinézetét az elemnek és kezelik az eseményeit a bennük lévő elemeknek. A View osztályai a megjelenítési logikán és az esemény továbbításon kívül semmilyen logikát nem tartalmaznak, minden feladatot az alattuk lévő csomagban a hozzájuk tartozó Presenternek adnak át.

A Presenter végzi a logikai feladatokat, ellenőrzi a bemenetelt, szólít meg más modulokat a kliensen belül( Events csomag) és kommunikál a szerverrel(Stubs csomag). A hozzá tartozó View-et egy interfészen keresztül éri el, ez látható a Display csomagban. Ezek az interfészek szeparálják el teljesen a logikát a grafikus felülettől a kliensben, úgy, hogy laza kapcsolattá avanzsálják a View és a Presenter közötti kapcsolatot. Ez a program javíthatóságán és modularitásán javít. Az osztálydiagramon feltüntetett függvények a Presenter és a View közötti kommunikáció interfészei, egyben az osztály publikus függvényei is.

#### Eseménykezelés modulok között

Az egyes modulok közötti eseménykezelést a GWT egy speciális osztálya, az EventBus kezeli. Ez egy osztály, amire esemény-objektumokat lehet küldeni, illetve feliratkozni egyes eseményekre. Az események megírt osztályok példányai, így bármilyen adatot képesek tárolni. A modulok létrehozásakor megkapják az EventBus objektumot, és az Events csomagban lévő események használatával tudnak üzenetet küldeni egymásnak.

#### ErrorHandlingAsyncCallback osztály

A kliens aszinkron hívásokkal híjva a szervert, ahonnan néha hibaüzenetek jöhetnek, ha nem elérhető a szerver, vagy ha a szerver belső hibába ütközik. Ekkor a hibaüzenetek kezelését ez az osztály végzi.

**public** **abstract** **class** ErrorHandlingAsyncCallback<T> **implements** AsyncCallback<T>

{

PopupPanel popup = **new** PopupPanel(**true**, **true**);

@Override

**public** **void** onFailure(Throwable caught)

{

popup.clear();

popup.add(**new** HTML(caught.getMessage()));

popup.show();

}

@Override

**public** **abstract** **void** onSuccess(T result);

}

Az eredeti AsyncCallback interfész két függvényét kell implementálni minden szerverhíváskor, az onSuccess, illetve az onFailure metódust. Az első akkor hívódik meg, ha az eredmény sikeresen megjött a szervertől, az onFailure pedig akkor, ha valamilyen hiba vagy http error jön vissza eredményül, azaz nem jött használható eredmény. Az ErrorHandlingAsyncCallback absztrakt osztály az onFailure metódust implementálja, és egy popup ablakban értesíti a klienst a felmerült hibáról. Így a hibakezelés egy helyen van, és tényleges szerverhívásokkor csak az sikeres működést kell implementálni. Amilyen egyszerű ez az osztály, annyira hasznos. Rengeteg kódismétléstől szabadít meg, és szervertől jövő hibakezelést gyakorlatilag ő végzi.

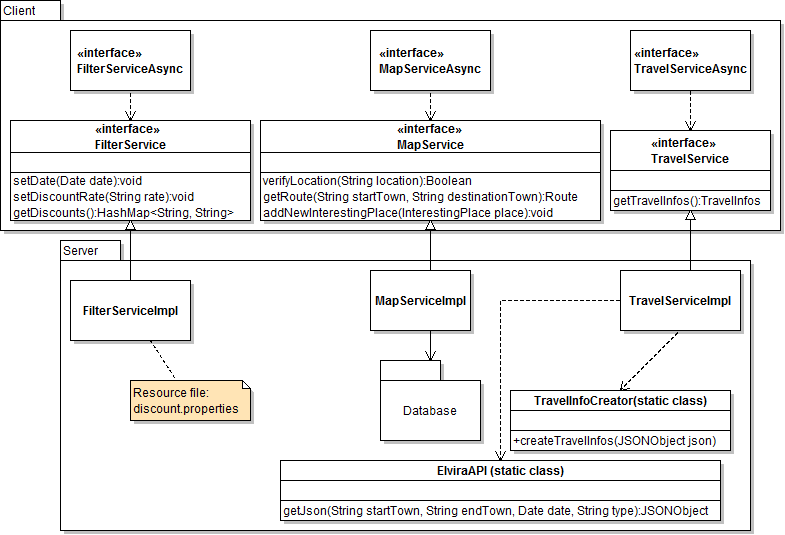
Mivel a szerverből jövő hibaüzenetek esetén a kliens nem sokat tehet, így ennél részletesebb hibakezelés implementálása nem szükséges.

#### Server csomag és az RPC kommunikáció

A szerver három interfészen keresztül kommunikál a külvilággal. Adatokat biztosít a kliensnek, adatokat ír és olvas az adatbázisból, valamint egy távoli REST interfészen keresztül kérdez le adatokat.

Ebből Sever csomagban a klienssel kommunikáló szervletek vannak. Ezek a szervletek a FilteringService, a MapService és a TravelService implementációi.

Ezeknek az osztályoknak az interfészei, un. stub-jai a kliens csomagban vannak, egy hozzájuk tartozó aszinkron interfész párral együtt. Ez a hármas: a kliensben lévő interfész és aszinkron párja, valamint a szerverben lévő implementáció együtt valósítja meg az RPC hívásokat a szerver és a kliens között. A kliensben a GWT keretrendszer segítségével példányosítani lehet az aszinkron interfészeket, amikhez fordításkor legenerálódik a kommunikációhoz szükséges kód.



. Server osztálydiagram és RPC mechanizmus

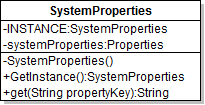
TODO: A Server package képről kimaradt a SystemProperties

A három szervlet három különböző feladatot lát el.

* FilterServiceImpl
  + Elmenti a szűrésként beállított adatokat a felhasználóhoz kötött session-ban, valamint biztosítja a szűrés megjelenítéséhez szükséges adatokat a kliensnek.
* MapServiceImpl
  + A térképes műveletek futnak be hozzá, ő szerkeszi és olvassa az RDF adatbázist
* TravelServiceImpl
  + Ő kommunikál a külső REST API-val, kéri le az adatokat a szűrés alapján, és állítja őket össze RPC-n átküldhető adatstruktúrává, segédosztályok segítségével.

#### SystemProperties

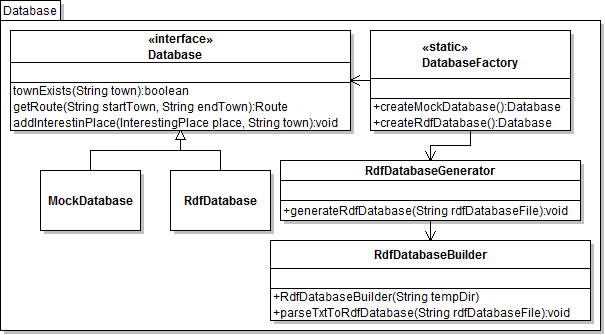
A szerver csomagban található a SystemProperties singleton osztály is. Ez az osztály a szerver konfigurációs fájljában megadott paraméterek lekérdezését teszi lehetővé, hasonlóan a Java System.getProperties() statikus függvényéhez hasonlóan.



. SystemProperites singleton

### Adatbázis

A program adatait egy RDF adatbázisban tárolja. Az adatbázis egy rdf kiterjesztésű fájlban van tárolva egy tetszőleges, a szerver indításakor megadott helyen.



. Adatbázis osztálydiagram

TODO:Az adatbázis készítő/menedzselő osztálydiagram elmagyarázása  
  
Adatbázis létrehozása

Amennyiben az adatbázis nem létezik, a szerver létrehozza azt.

A szerver elindításakor a MapService servlet létrehozásakor indul el az adatbázis generálás folyamata. Ennek az adatbázisnak a létrehozása a forrásadatok méretétől és a szerver kapacitásától függően 10 perc is lehet, és program memóriaigénye is jelentősen megnőhet. Ezalatt a szerver letölti az előre megadott helyről azt a zip fájlt, amiben található fájlok megfelelnek a GTFS(General Transit Feed Specification) specifikációnak, kitömöríti és beolvassa a benne található csv fájlokat, és felépíti belőlük az rdf adatbázist amelyben szerepelnek a megállók/városok, a közöttük menő útvonalak, valamint minden városhoz egy üres lista, ami annak a helynek a látnivalóit fogja tartalmazni. Ezután a zip-fájlt és az ideiglenes fájlokat törli.  
  
Szerkezeti felépítés  
Az RDF adatbázis hármasokból áll, amit *statement*nek hívunk. A hármas két elem, és a közöttük lévő relációból áll. A *subject* az első eleme a hármasnak, ami relációban áll a hármas harmadik elemével, az *object*el. A köztük lévő reláció a hármas középső tagja, ezt *predicate*nek nevezzük. Ezekből a relációkból sok előre definiált létezik, amiknek használata hasznos, mert így egy reláción mindenki ugyanazt fogja érteni.  
például a Város Neve Budapest hármasban a Neve reláció definiálva van és elterjedt reláció. Ezek a relációkat az RDFben felsorolt névterek azonosítanak egyértelműen.

#### RDF névterek

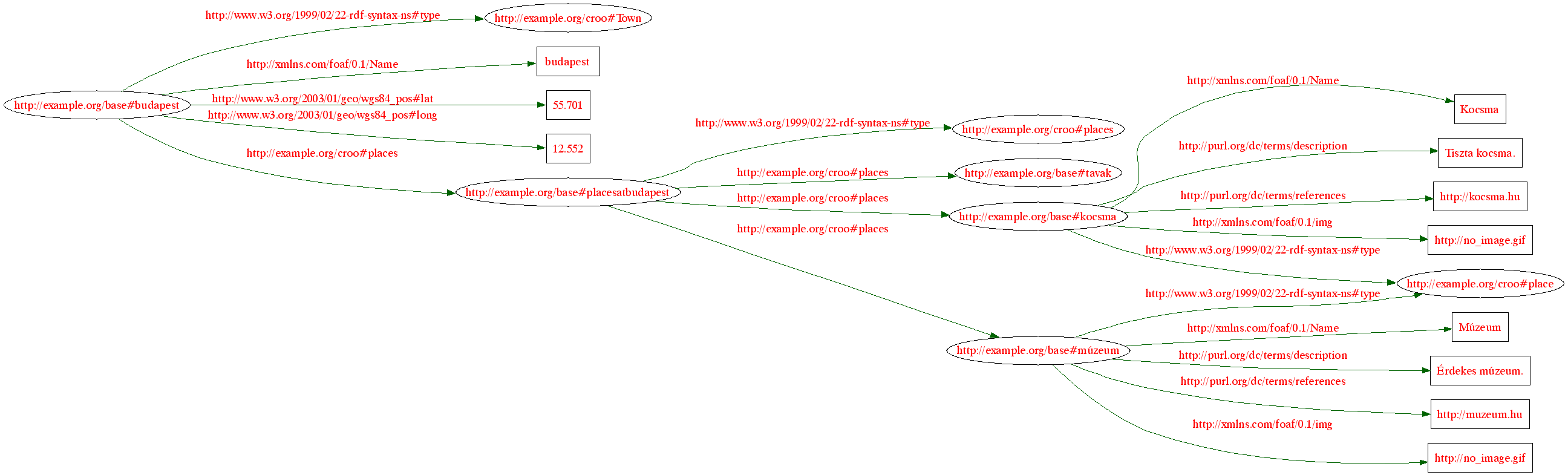
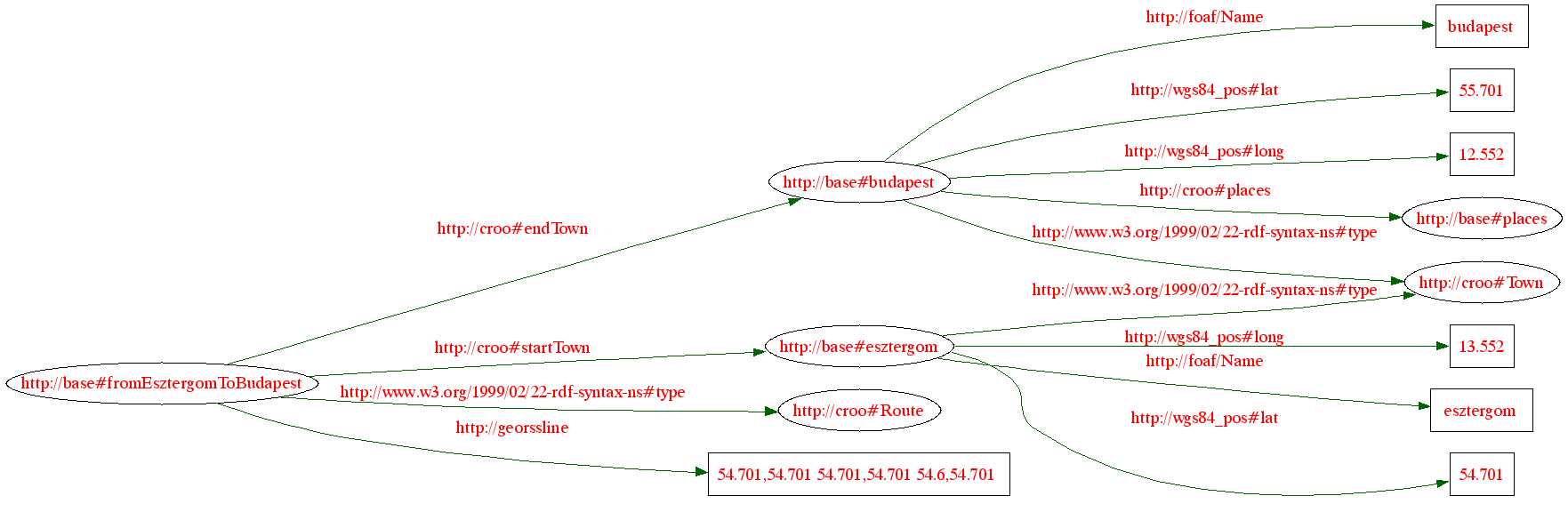
névterek és azok elemei segítségével írható le az egyes elemek közötti reláció.

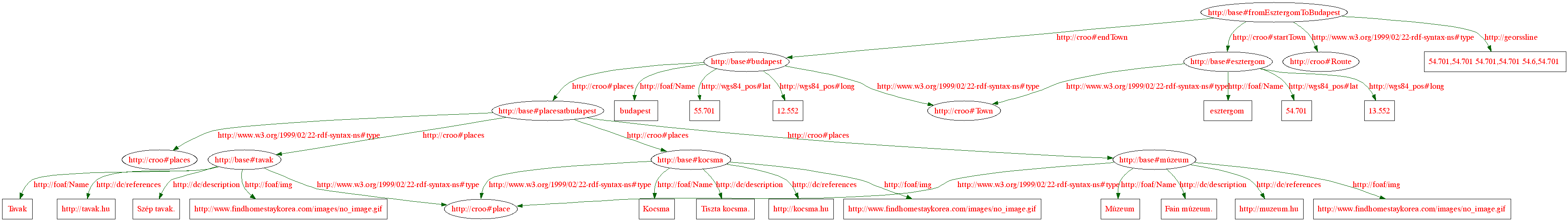
Az adatbázis a következő információkat tartalmazza:

* **rdf**=http://www.w3.org/1999/02/22-rdf-syntax-ns#
  + Az rdf adatbázis struktúrájában használt elemek névtere.(például ID,about) Minden RDF adatbázisban megtalálható.
* **dc**=http://purl.org/dc/terms/
  + A Dublin Core névtere. A honlap(**references**) és a leírás(**description**) relációkat használja belőle az adatbázis.
* **geo**=http://www.w3.org/2003/01/geo/wgs84\_pos#
  + A WGS84[[9]](#endnote-9) specifikációnak megfelelően földrajzi helyek leírására alkalmas névtér. A szélesség(**lat**) és a hosszúság(**long**) koordináták leírását használja belőle az adatbázis.
* **georss**=http://www.georss.org/georss
  + Földrajzi koordináták, útvonalak tárolására alkalmas névtér. Az adatbázisban az útvonal leírására(**line**) van használva.
* **foaf**=http://xmlns.com/foaf/0.1/
  + foaf azaz „friend of a friend” nevű névtér, emberi kapcsolatok leírására. A nevek(**name**) és kép-urlek(**img**) tárolására.
* **croo**=http://example.org/croo#
  + Ez az adatbázis saját névtere amiben az adatbázisra egyedi elemek találhatóak. Azaz a városok(**town**) az útvonalak(**route**) és az érdekes helyek(place).

Az adatbázis elemei

**Város(Town):**- A város neve  
- A város pontos helyrajzi koordinátája  
- Latitude  
- Longitude  
- A városban található érdekes helyek listája  
  
**Érdekes Hely(Place):**- A hely neve  
- A hely leírása  
- A hely honlapja  
- Egy, a helyhez kapcsolódó kép netes linkje  
  
**Útvonal(Route):**- Az útvonal kezdőállomása  
- Az útvonal végállomása  
- Az útvonalat leíró koordináták listája  
  
Ezek a köztük lévő relációk alapján egy gráfot alkotnak:



### Felhasználói Felület

A felhasználói felület – feltéve, hogy van – tervét (a képernyő- és listaterveket, valamint a menütervet). Legyen egy áttekintő ábra, amely mutatja a képernyők (ablakok, weblapok) közti navigálási lehetőségeket, irányokat. Ki kell emelni a fontosabb felhasználói eseménykezeléseket.

## Megvalósítás

A fejlesztői leírásnak a megvalósításról szóló része bemutatja, hogy milyen döntéseket kellett hozni a terv megvalósítása során (adatábrázolás, felhasznált komponensek, kódban alkalmazott nyelvi elemek, stb). A dokumentáció ne tartalmazza a forrásprogramot (legfeljebb csak fontosnak ítélt részleteit), elég azt a mellékelt adathordozón elhelyezni. A megvalósítás a fentieken kívül tartalmazza a komponens tervet (az alkalmazás fizikai komponenseinek kapcsolatrendszerét) és azok telepítésének módját.

### #Ez mi a fene, cleancode-ról meséljek egy egész fejezetet? Oké...

A forráskód tartalma, szerkezete megfelel-e a tervnek?

 Mennyire ismeri a hallgató az adott fejlesztő eszközt (pl. korszerű, hatékony nyelvi elemek vannak-e túlsúlyban, vagy ehelyett bonyolult, nehézkes, körülményes és leginkább terjengős forráskódot eredményező nyelvi elemek jellemzik a kódot)? Indokoltak-e a választott nyelvi elemek használata?

 Milyen a forráskód külalakja, mennyire áttekinthető (strukturáltság, bekezdések, tagolások, kommentezés stb.)?

 Mennyire módosítható a kód. Alkalmazza-e a hallgató a kód-újrafelhasználás nyelvi eszközeit (függvények, származtatás, generikus elemek)?

 Törekszik-e a hatékony adatábrázolásra?

 Mennyire öndokumentáló a kód, vagyis a választott azonosítók (pl. változónevek) mennyire beszédesek, konvencionálisak, a megjegyzések mennyire segítik a kódértést?

 Tartalmazza a szükséges ellenőrzési, hibakezelési funkciókat, általában megoldott-e a kivételkezelés?

 Mennyire gazdálkodik jól az emberi és gépi erőforrásokkal, így például a felhasználó idejével és türelmével, a lemezkapacitással és a memóriakapacitással?

## Tesztelés

Ez is a fejlesztői leírás része, amelynek a tesztelési szempontokat kell bemutatnia, és a tesztelés során szerzett tapasztalatokat összegeznie valamint a szoftver skálázhatóságáról készített elemzést kell tartalmaznia.

Az értékelésnél vegyük figyelembe, hogy a dokumentáció:

### Junit

 Tartalmaz-e tesztelési terveket, teszteseteket (Ezeket csoportosíthatja rendszerteszt és modultesztek szerint illetve fekete és fehérdoboz megközelítéssel)?

### MVP / Database interface / Sycron vs Asyncron hívások

 Beszámol-e olyan tanulságokról, amelyek alapján meg kellett változtatni a korábbi implementációs döntéseket, esetleg a terv egyes elemeit (az ilyen tapasztalatok nem rontják a dolgozat értékét)?

### RDF Adatbázis mérete

 Tartalmazza-e nagy adattömeg melletti futtatások értékelését?

### Eeh...? Eredményhelyesség - kihagyós

 Elemzi-e a program által adott eredmény helyességét (különösen olyan optimalizációs feladatok esetén, ahol több helyes megoldást valamilyen célfüggvénnyel lehet rangsorolni)?

### Hatékonyság elemzés: GWT Speed Tracer

 Elemzi-e a program futásának hatékonyságát?

## 

# Irodalomjegyzék

GWT dokumentáció

Robert C. Martin – Clean Code

Martin Flower – Refactoring

1. GWT: Google Web Toolkit – Keretrendszer webes alkalmazások fejlesztéséhez [↑](#endnote-ref-1)
2. RPC :Remote Procedure Call – Elosztott rendszerek kommunikációjához használt technológia [↑](#endnote-ref-2)
3. Robert C. Martin – Clean Code – Könyv illetve előadássorozat a jól olvasható kód írásáról, lásd irodalomjegyzék [↑](#endnote-ref-3)
4. Martin Flower – Refactoring – Köny a kód „refaktorálásáról”, struktúrájának javításáról, lásd irodalomjegyzék [↑](#endnote-ref-4)
5. WAS – Web Application Server – Speciális Java alkalmazások futtatási környezetét biztosító alkalmazásszerver [↑](#endnote-ref-5)
6. JVM – Java Virtual Machine – virtuális futtatókörnyezet, minden Java alkalmazás egy JVMben fut [↑](#endnote-ref-6)
7. warfile – .war kiterjesztésű fájl. A web application archive rövidítése - Javaban írt webes program [↑](#endnote-ref-7)
8. properties fájl - .properties kiterjesztésű fájl ami kulcs érték párokat tartalmaz egyenlőségjellel elválasztva : key=value [↑](#endnote-ref-8)
9. WGS84 – Földrajzi fogalom: vonatkoztatási rendszer, a Földet globálisan közelítő ellipszoid-modell. [↑](#endnote-ref-9)