EÖTVÖS LORÁND TUDOMÁNYEGYETEM

TERMÉSZETTUDOMÁNYI KAR

HELYKÖZI BUSZJÁRATOK INTERAKTÍV VIZUALIZÁCIÓJA MENETRENDI ADATOK ALAPJÁN

SZAKDOLGOZAT

FÖLDTUDOMÁNYI ALAPSZAK

KÉSZÍTETTE:

Eigner Péter

térképész és geoinformatikus szakirányú hallgató

TÉMAVEZETŐ:

dr. Gede Mátyás

adjunktus

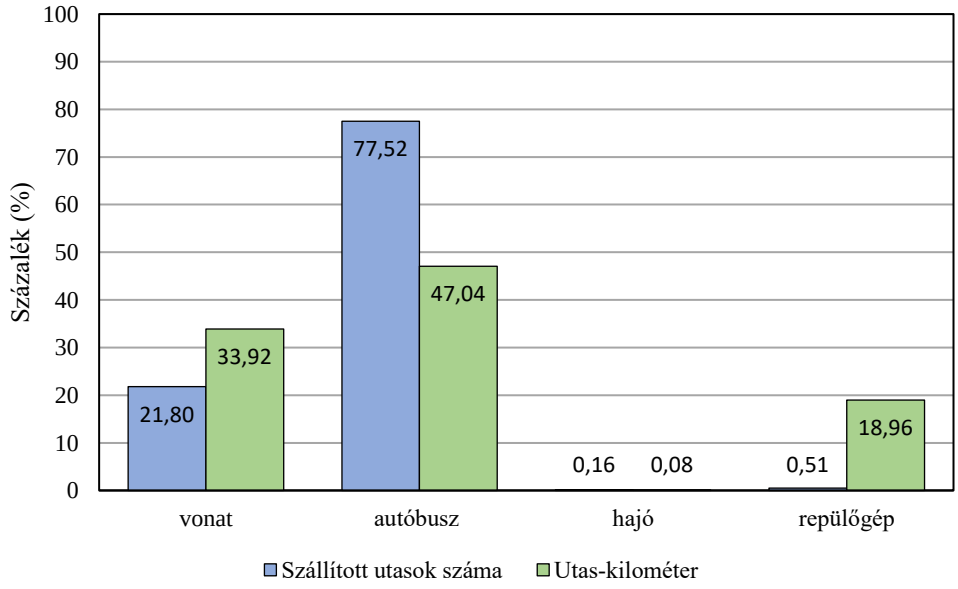
ELTE Térképtudományi és Geoinformatikai Tanszék

Budapest, 2017

Tartalomjegyzék

1. Bevezetés

Napjainkban hazánk helyközi közösségi közlekedését döntően regionális busztársaságok bonyolítják le. Helyközi közlekedésnek tekintjük a belföldön, települések között végzett személyszállítási szolgáltatást, ide értve az annak részeként egyes települések közigazgatási határán belül – helyközi díjszabás alapján – végzett személyszállítást is. [Volánbusz Zrt. Üzletszabályzat] A Központi Statisztikai Hivatal adatai alapján a 2001-2016-ig tartó időszakban a helyközi személyszállításban részt vevő utasok 77,52%-a vette igénybe a közösségi autóbusz-közlekedést, ugyanez az érték a vasúti közlekedés esetében 21,8%, a vízi közlekedés esetében 0,16%, míg légi közlekedés esetében 0,5%. (*1.ábra*)



1. ábra: A helyközi személyszállítás megoszlása az egyes közlekedési ágazatok között (KSH)

A magyarországi Volán társaságok teljes körű szervezeti átalakítása a dolgozat megírását megelőző években zajlott le. A busztársaságok régiós szintű összeolvasztásának eredményeképpen, a korábbi 24 állami tulajdonban lévő vállalatot 7 új regionális társaságba szervezték. Az összevonások elsődleges célja a cégek működési hatékonyságának növelése, ami egyúttal színvonalasabb utaskiszolgálást is eredményez. A szolgáltatási színvonal növelése keretében kiemelt jelentősége van az utastájékoztatási rendszerek átfogó megújításának. Ezeknek az utastájékoztató és forgalomirányító rendszereknek szerves részét képezik azok a web alapú térképek, amelyek segítségével az utasok nem csak a menetrendi információk

között igazodhatnak el könnyebben, de a menetrendekhez kapcsolódó földrajzi adatok megjelenítésével utazásuk térképen is könnyen megtervezhetővé és nyomon követhetővé válik. Hasonló fejlesztések zajlanak az ország több településén a helyi tömegközlekedésben, és országos szinten a vasúti személyszállításban egyaránt [Világgazdaság, 2014].

Azonban a tömegközlekedési cégek közül ma még nem mindegyik rendelkezik ehhez hasonló térképes felülettel, illetve a busztársaságok jelenleg használatban lévő alkalmazásai sem készültek azonos irányelvek szerint. A térképek tartalmi elemei és azok ábrázolásmódjai különbözőek, megjelenítésük bizonyos esetekben kirívóan hibás, továbbá a menetrendi adatok térképi vizualizációjában rejlő lehetőségek sokszor kiaknázatlanok. Ezen okok hatására született meg dolgozatom témája.

A dolgozat célja tehát, a fent említett tapasztalatokból kiindulva, egy olyan interaktív webtérkép megalkotása, amely átfogó képet nyújt egy vizsgált terület (Somogy megye) buszközlekedéséről, illetve a rajta ábrázolt tematikus tartalom áttekinthetősége és a megjelenített objektumok összehasonlíthatósága segíti a térkép alapú utazástervezést.

2. Interaktív utastájékoztató térképek a hazai tömegközlekedésben

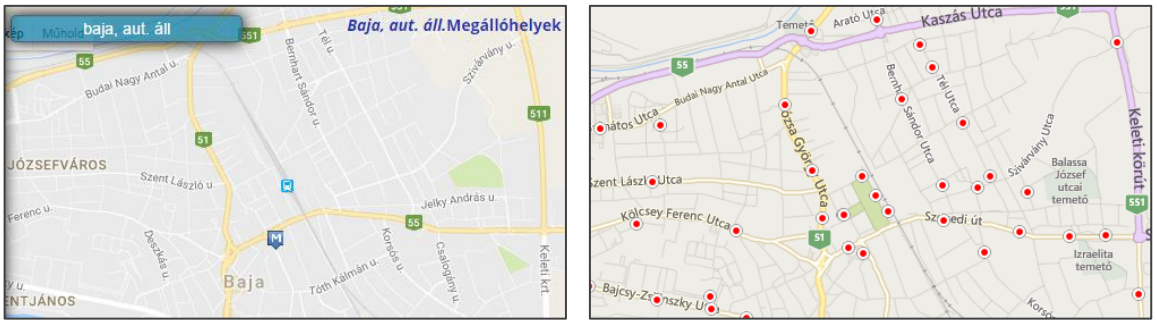
Ahogy az a bevezetésben említésre került, a hazai személyszállító cégek egy része már rendelkezik olyan online utastájékoztató alkalmazással, amelyek térképes felületen is szolgáltatnak információt a felhasználók részére. Ezek a webtérképek azonban jelentős különbségeket mutatnak a megjelenített térképi tartalom, a megjelenítés módja, és hozzájuk kapcsolódó további lehetőségek tekintetében. Ebben a fejezetben szeretném példákkal és ellenpéldákkal illusztrálva bemutatni és összehasonlítani ezeket a térképeket.

Az webes utastájékoztató térképek három alapvető funkcióval rendelkeznek:

- a megállóhelyek elhelyezkedéséről és a járatok vonalvezetéséről, illetve ezek menetrendjéről adnak információt,
- az előbbi adatokat felhasználva utazástervező (útvonaltervező) feladatokat látnak el,
- valós idejű járatkövetést biztosítanak, amennyiben a járat műholdas járműkövető rendszerrel van ellátva.

Ezek közül legtöbb esetben az utazástervező funkció dominál. A megállóhelyek és a hozzájuk kapcsolódó adatok általános ábrázolása háttérbe szorul.

Az alapvetően utazástervező alkalmazások általában sokszor csak egy adott lekérdezés eredményét jelenítik meg, nincs mód a vonalhálózat és a megállóhelyek áttekintésére, vagy egy adott megállót érintő vonalvezetések megjelenítésére. Ugyanez igaz az éppen közlekedő járatok ábrázolására is. Egy adott járat megjelenítésén túl, az összes úton lévő járat megtekintésére ugyanazon a felületen általában nincs lehetőség.

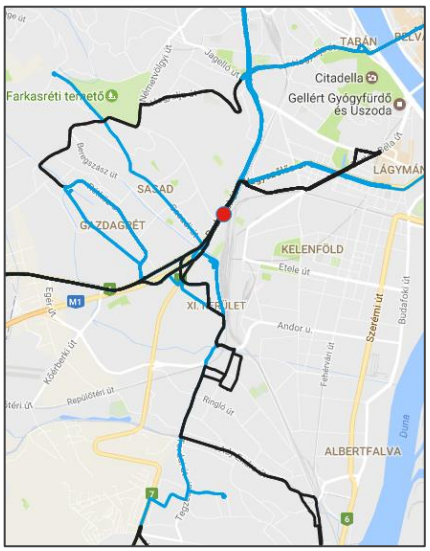


2/a *2/b*

2. ábra: Baja város megállóinak megjelenítése két különböző cég (2/a-2/b) utastájékoztató felületén

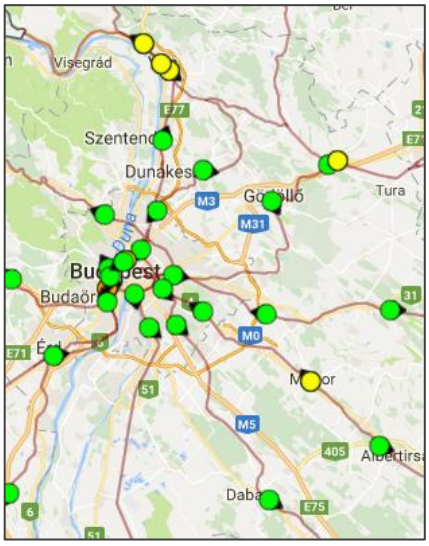
Mivel legtöbbször a megállóhelyek ábrázolása is csak lekérdezés útján érhető el (*2/a ábra*), nehézségbe ütközhetünk, ha olyan megállók adatait szeretnénk megtekinteni, amelyeknek a nevéről pontos információink nincsenek (de elhelyezkedését esetleg ismerjünk), vagy ha egy bizonyos terület, település összes megállóját szeretnénk megjeleníteni. (*2. ábra*)

A megállóhelyekhez kapcsolódó adatok ábrázolásában több olyan lehetőség is rejlik, amelyek jelenleg csak egy-két felületen érhetők el. Ilyen hasznos funkció az egy adott megállóhelyet érintő járatok vonalvezetésének megjelenítése. (*3. ábra*) Ezzel az funkcióval lehetőségünk nyílik térképen is információt szerezni arról, hogy egy megállóhelyről milyen útvonalon, milyen célállomások felé közlekednek járatok. A helyközi közlekedés témakörére vonatkoztatva ez azt jelenti, hogy tájkozódhatunk arról, hogy egy településről mely másik települések közelíthetőek meg közvetlen járatokkal.



*3. ábra: A Dayka Gábor utca megállóhelyet érintő **nappali** és **éjszakai** buszjáratok a BKK Futár térképén*

Az alapvetően járműkövetési profilú alkalmazások már rendelkeznek a fent említett komplex megjelenítési lehetőségekkel a közlekedő járatokra vonatkozóan, viszont csak a többi megjeleníthető tartalom terhére. A pillanatnyilag közlekedő összes (vagy csak több) járat vizualizációja hasznos lehet, ha utazásunkat több járaton, átszállással végezzük (*4. ábra*). Ilyenkor fontos, hogy ne csak egy keresett járat, hanem mind a két járat aktuális helyzetét legyen lehetőségünk látni, amelyek között az átszállás történik. A valós idejű járműkövetés pedig azt is lehetővé teszi, hogy a járatok késését szemléltessük, így megkönnyítve a csatlakozás nyomon követését.



4. ábra: A pillanatnyilag közlekedő vonatok ábrázolása a vonatinfo.hu térképén

3. Az alapanyag feldolgozása

A dolgozat alapjául a Dél-dunántúli Közlekedési Központ Zrt. (továbbiakban: DDKK) által szerkesztett 2016. évi Somogy megyei autóbusz menetrend digitális adatbázisa szolgál, amelyet a Társaság Somogy megyei Szolgáltatási Központja bocsátott rendelkezésemre. A menetrendi adatbázis a Somogy megyében közlekedő belföldi országos és regionális közforgalmú autóbuszjáratok menetrendjét és vonatkozó adatait tartalmazza. Ezen túlmenően tartalmazza a megye területén áthaladó –esetlegesen más személyszállítási szolgáltató által üzemeltetett– belföldi országos autóbuszvonalak menetrendjét is.

A DDKK adatbázisából exportált adatokat a cég **CSV** állományok formájában juttatta el hozzám. A CSV (Comma-Separated Values, magyarul: vesszővel tagolt értékek) állomány egy olyan szöveges fájl, amely táblázatos adatok speciális tárolására és megjelenítésére alkalmas. A táblázat celláinak tartalmát egy jól meghatározott szimbólum választja el egymástól (*5. ábra*). Az elválasztó szimbólum általában vessző vagy pontosvessző [GEVAPC, 2011]. A CSV formátumú fájlok nagy előnye, hogy könnyen importálhatók –az általam is alkalmazott– SQL alapú adatbáziskezelőkbe.

mr_akt_megallo.csv

```
1  MEGALLOKOD;ERVKEZD;MEGALLONEV;ROVIDNEV;HONOSSAGI;TIPUS;TELEPULESKOD;ELHELYEZKED;MEGJEGYZES;WGSX;WGSY
2  6203;;Szenyér, felső;Szenyér, felső;1;1;8717;1;;17.37422;46.464245
3  6204;;Szenyér, alsó;Szenyér, alsó;1;1;8717;1;;17.371933;46.457108
4  6205;;Szenyéri elág.;Szenyéri elág.;1;1;87171;0;;17.373957;46.447624
5  6206;;Tapsony, alsó;Tapsony, alsó;1;1;8718;1;;17.340384;46.451824
6  6207;;Tapsony, ib.;Tapsony,ib.;1;1;8718;1;;17.332638;46.457291
7  6208;;Nagyszakácsi, temető;Nszakácsi,tem.;1;1;8739;1;;17.322777;46.47958
8  6209;;Nagyszakácsi, ib.;N.szakácsi,ib.;1;1;8739;1;;17.321001;46.48687
9  6210;;Nagyszakácsi, felső;Nszakácsi,felső;1;1;8739;1;;17.318977;46.495106
10 6211;;Nagyszakácsi, Cserhát;Nszakácsi,Cserh;1;1;8739;1;;17.316114;46.509289
11 6212;;Nemeskisfalud, bej. út;Nemeskisfalud,;1;1;87170;1;;17.376892;46.43914
12 6213;;Böhönye, felső;Böhönye, felső;1;1;8719;1;;17.383667;46.414059
13 6214;;Böhönye, közp. aut. vt.;Böhönye, közp.;1;1;8719;1;;17.392984;46.407375
14 62141;;Böhönye, nagyatádi elág.;Böhönye, nagyat;1;1;8719;0;technikai;;
15 6215;;Böhönye, vá. bej. út;Böhönye, vá. be;1;1;8719;1;;17.405609;46.407803
```

5. ábra: Részlet a megállóhelyek adatait tartalmazó CSV fájlból

3.1. Saját adatbázis kialakítása

A webtérkép létrehozásához szükséges egy önálló adatbázis kialakítása, a rendelkezésre álló CSV állományokból.

Az újonnan létrehozott adatbázis egy relációs adatbázis, vagyis az adatok több adattáblában kerülnek tárolásra, viszont a táblák között egyértelmű kapcsolat van. Egy adattáb-

lának összesen két dimenziója van, a sorok és az oszlopok. A sorokat szakszerűen rekordoknak, az oszlopokat pedig a tábla mezőinek nevezzük. Ebből értelemszerűen következik, hogy minden rekord ugyanannyi számú mezőt tartalmaz. Minden mező meghatározott típusú lehet (string, integer, date stb.) A rekordok és mezők keresztmetszete az attribútum [ELEK, 2011].

Saját adatbázisomat **phpMyAdmin** felületen készítem és szerkesztem. A phpMyAdmin, egy olyan PHP nyelven írt, nyílt forráskódú, grafikus felületű eszköz, amelyet MySQL adatbázisok internetes menedzselésére alkottak. Az eszköz teljesen platformfüggetlen, és grafikus böngészővel futtatható.

A **MySQL** –ahogy nevéből is következik– egy SQL-alapú adatbázis-rendszer. Az SQL a Structured Query Language kifejezés rövidítése, amelynek jelentése strukturált lekérdezőnyelv. Ennek használatával végezhetünk lekérdezéseket a MySQL-hez hasonló relációs adatbázis-kezelő rendszereken.

A CSV fájlok közül azok kerülnek importálásra az új adatbázisba, amelyek a térképi megjelenítés szempontjából releváns adatokat tartalmazzák. Importálás előtt azonban nagyon fontos az importálandó CSV-k karakterkészletének pontos beállítása, azaz az adatbázis-kiszolgáló karakterkódolásához (utf8) való illesztés. A fájlok Notepad++ szövegszerkesztőben könnyen átalakíthatók UTF-8 kódolásra. Ezután kerülhet sor az adatok beolvasására.

Nélkülözhetetlen azonban az így létrejött „nyers” adatbázis pontosítása. A folyamat során igyekszem kiszűrni és korrigálni az importálás, vagy a kezdeti exportálás okozta hibákat, pótolni az egyébként kulcsfontosságú hiányzó adatokat, és további szűréseket végezek a táblák mezői és rekordjai között, hogy végül olyan adat ne szerepeljen az adatbázisban, amely később a térképi tartalom kialakítása szempontjából szükségtelen.

3.1.1. Az adatok korrekciója

Az adatbeolvasás során olyan hibák merülhetnek fel, amelyek az adatok helytelen importálását idézik elő. Ilyen kedvezőtlen helyzet áll elő akkor, ha olyan számokat, vagy idő adatokat importálunk egy táblába, amelyek kezdő vagy záró helyi értékén nulla számjegy áll. Ilyenkor a MySQL a beolvasott kezdő nulla helyi értékeket levágja a számról. Ez komoly problémát okoz például a megállók adatait tartalmazó tábla (*megallok*) importálásakor.

Ebben a táblában a megállók egyedi azonosítóját tartalmazó mező az *mkod* (megállókód). A megállókód egy kizárólag számjegyekből álló karaktersor. Számként való importálásuk viszont éppen a fent leírt probléma miatt nem lehetséges, mivel sok esetben a megállókódok között egy kezdő nulla tesz csak különbséget. (*6.ábra*)

mkod ▲ 1	mnev	rnev	telepuleskod	lon	lat
01000	Budapest, Sasadi út	Budapest, Sasadi út	1810	19.016947	47.467262
1000	Kaposvár, autóbusz-állomás	Kaposvár, aut. áll.	7400	17.790956	46.354343

6. ábra: Példa a kezdő nulla érték jelentőségére a megállókódban

Ezért a megállókódokat szöveg típusú adatként importáltam, így az importálás során a kezdő nullák megmaradtak. A szövegként való importálásra azért van lehetőség, mert bár az *mkod* kizárólag számokból áll, de nem számként funkcionál, csak egy azonosító karaktersor a táblában.

Nem ilyen könnyen orvosolható a probléma az idő adatok beolvasása estében, ahol már a kezdeti exportálás során levágásra kerültek a kezdő és záró nulla értékek. A buszjáratok menetrendjét leíró táblában (*menetrend*) két idő típusú adatot tartalmazó mező van, amelyet ez a probléma érint. Ezek egy olyan CSV-ből kerültek beolvasásra, amely

- levágott záró nulla értékekkel rendelkezik, vagyis:
  - nem tartalmazza a 00 perc értékeket, amennyiben az idő adat egész óra (pl.: 13:00 a CSV-ben: 13),
  - levágja az egyéb 0-ra végződő perc értékeket (pl.: 13:20 a CSV-ben: 13.2)
- továbbá egy számjeggyel írja le a 10-nél kisebb óra értékeket (pl.: 07:11 a CSV-ben: 7.11)

Az idő adatok korrigálásához fontos először beállítani egy egységes karaktert az időegységek elválasztására. Mivel ez korábban a CSV-kben a pont karakter volt, azonban a MySQL által alapértelmezett karakter a kettős pont, ezért az elválasztó karaktereket az adatbázisban az alábbi lekérdezéssel módosítottam:

```
UPDATE jarat SET indulas=(REPLACE(indulas, '.',':')
```

Ezután pótlom a hiányzó nulla karaktereket az alábbi módon:

```
UPDATE jarat SET indulas=(REPLACE(indulas,':1',':10')

WHERE indulas LIKE '__:1'
```

A példa a :10 -re végződő értékek módosítását mutatja, a további adatok módosítása ugyanerre a sémára épül.

3.2. Az adatbázis felépítése

Az adatbázist a *terkeptar.elte.hu*-n futó MySQL 5.5.54 adatbázis-szerveren helyeztem el.

Az adatbázist felépítő táblák és azok mezői:

- **ceg** – Az buszjáratokat közlekedtető személyszállítási szolgáltatók adatai
  - *cegkod* – a cég egyedi azonosítója
  - *cegnev* – a cég neve
  - *megjegyzes*
- **megallok** – Az autóbusz-megállók adatai
  - *mkod* – a megállóhely egyedi azonosítója
  - *mnev* – a megállóhely teljes neve
  - *rnev* – a megállóhely rövid neve
  - *honossagi* – 1, amennyiben a DDKK Zrt. által üzemeltetett megállóhely. 0, ha más cég üzemelteti.
  - *tipus*
  - *telepuleskod* – annak a településnek az egyedi azonosítója, amelyik területén a megállóhely elhelyezkedik
  - *elhelyezked* – 1, amennyiben a település belterületén helyezkedik el. 0, ha azon kívül.
  - *megjegyzes*

- *lon* – a megállóhely földrajzi koordinátái (hosszúság)
- *lat* – a megállóhely földrajzi koordinátái (szélesség)

- **menetrend** – A járatok menetrendjét leíró tábla

  - *vonalkod* – az autóbusz-vonal egyedi azonosítója
  - *jaratszam* – a vonalon közlekedő járat azonosítója
  - *sorszam* – a járat valamely megállóhelyének sorszáma
  - *megallokod* – a megállóhely egyedi azonosítója
  - *kocsiallas* – a kocsiállás száma
  - *gongyolt_km* – az induló állomástól megtett út hossza (km)
  - *elozotol_km* – az előző megállótól megtett út hossza (km)
  - *erkezes* – a járat megállóba érkezésének időpontja
  - *indulas* – a járat megállóból (tovább)indulásának időpontja
  - *beterojel*
  - *felszallo_leszallo* – F, amennyiben a járat az adott megállóban csak felszállók részére áll meg. L, amennyiben a járat az adott megállóban csak leszállók részére áll meg. Az attribútum egy pont karakter, ha felszállásra és leszállásra egyaránt van lehetőség.
  - *poggyasz_kez* – I, amennyiben poggyászkezelés igénybe vehető
  - *hatarallomas*
  - *hangbemondas*

- **jarat** – a járatokra vonatkozó általános információk

  - *vkod* – a buszvonal egyedi azonosítója
  - *jszam* – a járat egyedi azonosítója
  - *vvkod* – a vonalvezetés egyedi azonosítója
  - *kkorlat* – a járatra vonatkozó közlekedési korlát (mely napokon közlekedik a járat?) azonosítója
  - *cegkod* – a járatot üzemeltető cég azonosítója
  - *irany* – 0 vagy 1. (oda/vissza)
  - *vastag*
  - *jtipus* – jegytípus

- *megjegyzes*
- *kiegjegy* – 1, ha felár megfizetésével igénybe vehető autóbusz-járat
- *csak_elovetel* – 1, ha a járat csak elővételben váltott menetjeggyel vehető igénybe
- *kot_ulohely* – 1, ha a járat csak külön díj ellenében történő ülőhely-foglalással vehető igénybe
- *eloallat_nem*, 1, ha a járaton élőállat nem szállítható
- *alacsonp* – 1, ha a járatot alacsonypadlós autóbusz végzi
- *wifi* – 1, ha a járaton WiFi rendelkezésre áll
- *inet_jegy* – 1, ha a járatra elővételben internetes jegyváltás lehetséges
- *sebes* – 1, ha nagy sebességű járat

- **kkorlat** – a tábla azt írja le, hogy adott napra, mely közlekedési korlátok vannak érvényben

  - *datum* – a 2017. évi naptári napok
  - *kkorl* – a közlekedési korlát azonosítója
  - *korlnev* – a közlekedési korlát teljes megnevezése
  - *kozlekedik* – I, amennyiben az adott közlekedési korlát, az év adott napjára érvényben van. N, ha nincs.

- **szakaszok** – a buszjáratok útvonalának elemi szakaszai

  - *id* – az elemi szakasz azonosítója
  - *mkod1* – a kiinduló megálló kódja
  - *lon1* – a kiinduló megállóhely földrajzi koordinátái (hosszúság)
  - *lat1* – a kiinduló megállóhely földrajzi koordinátái (szélesség)
  - *mkod2* – az érkezési megálló kódja
  - *lon2* – az érkezési megállóhely földrajzi koordinátái (hosszúság)
  - *lat2* – a érkezési megállóhely földrajzi koordinátái (szélesség)
  - *geom* – a vonalszakasz geometriája

4. A weblap felépítése

A dolgozat végeredménye egy weboldal, amelynek legjelentősebb egysége az a térképi felület, amelyen az adatbázis információi megjelenítésre kerülnek.

A weboldal **HTML**-nyelv használatával íródott. A HTML a HyperText Markup Language angol kifejezésből alkotott mozaikszó, jelentése hiperszöveges jelölőnyelv. A hiperszöveg nem más, mint különböző szövegek, képek, videók stb. kombinációjából álló dokumentumoknak az együttese. A HTML dokumentum pedig egy egyszerű szövegfájl amibe meghatározott szabályok szerint parancsokat (HTML kódokat) illesztünk, és ezekkel írjuk le a weblap szerkezetét. A HTML tehát nem programozási nyelv, hanem egy kódnyelv, amivel egyszerű szövegszerkesztő programban készíthetünk weblapot.

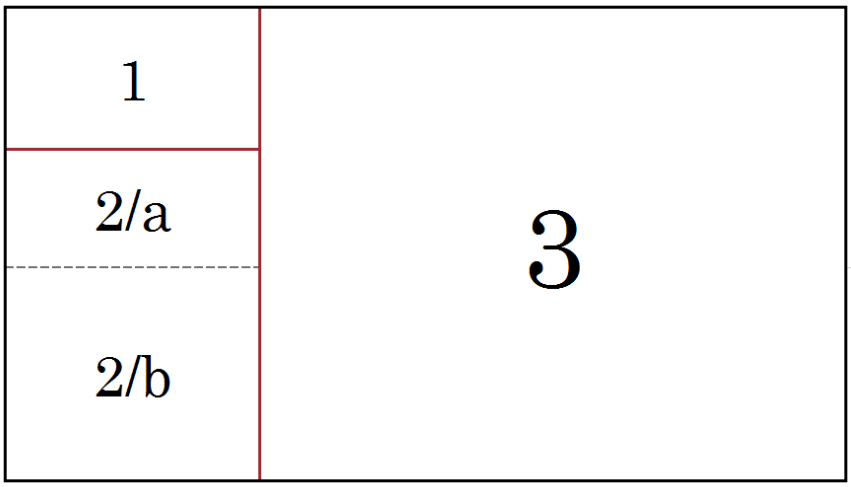
A weblap formai kinézetét **CSS** segítségével alakítottam ki. A CSS (*Cascadium Style Sheets*) egy stílusleíró nyelv, amely a HTML szegényes formázási lehetőségeit orvosolja. Segítségével olyan stíluslapokat gyárthatunk, amelyek megkönnyítik oldalaink formázását, és a strukturálásban is szerepet játszanak. Ezekkel a stíluslapokkal a szövegformázástól kezdve a szövegigazításon, sorközön, kereten, margón át egészen a színekig gyakorlatilag minden formázási kívánságot teljesíthetünk [PACZONA, 2001].

Mivel a HTML egy egyszerű szöveges leírónyelv, képtelen válaszolni a felhasználónak, döntéseket hozni, vagy automatikusan végrehajtani különböző feladatokat. Az ilyen jellegű feladatok elvégzéséhez egy összetettebb nyelvre, azaz egy parancsnyelvre vagy programozási nyelvre van szükség [MONCUR, 2002]. A **JavaScript** az első és máig legnépszerűbb webes parancsnyelv, így a weboldalam összetettebb funkciói is ezen a nyelven íródtak.

A weboldal működése a kiszolgáló (szerver), ill. a kiszolgálón tárolt adatbázis és a felhasználó (kliens) folyamatos interakcióján alapszik. A tisztán HTML-ből álló weboldalak csak tartalommegjelenítésre alkalmasak. Ahhoz, hogy egy webhely interaktív webalkalmazássá változhasson, a webkiszolgálónak új, dinamikusabb szerepet kell betöltenie – ezt pedig a **PHP** (*PHP: Hypertext Preprocessor*) teszi lehetővé. A PHP parancsfájlokat (szkripteket) a webkiszolgáló tárolja és futtatja, az eredményt pedig HTML-oldalként küldi el a böngészőnek. A PHP lehetővé teszi, hogy egy a kiszolgálón futó parancsfájl kódjában SQL-utasításukat adjunk ki [BEIGHLEY, MORRISON, 2009]. Ezt az elvet követve végezhet a fel-

használó is lekérdezéseket az adatbázisból (pl. adott megálló vagy járat menetrendjére vonatkozóan), de ilyen SQL-lekérdezés eredményeképpen válnak megjeleníthetővé azok a dinamikus térképi elemek is, mint a pillanatnyilag közlekedő buszjáratok.

4.1. A weboldal szerkezeti egységei



7. ábra: A weboldalt felépítő egységek

A weboldal három jól elkülöníthető egységből tevődik össze (*7. ábra*):

- **1** – Általános tájékoztató elem: tartalmazza a weblap címét, az aktuális dátumot, időpontot. Az egyetlen elem, amelynek tartamlát a felhasználó nem képes befolyásolni.
- **2** – A térképi elemek attribútumait megjelenítő elem, amely két alegységből áll:
    - **2/a** – Az általános információkat tartalmazó elem (pl. megálló neve vagy egy járat száma, és az azt üzemeltető cég neve stb.)
    - **2/b** – Menetrendi elem: a megállóhely vagy a közlekedő járat menetrendjét tartalmazza táblázatos formában
- **3** – A térképfelület: méretéből és tartalmából adódóan is a legjelentősebb elem. Ezen a felületen zajlik a megállók, járatok és útvonalaik megjelenítése.

5. Az adatok vizualizációja

A weblap térképi felületét **OpenLayers** használatával hoztam létre. Az OpenLayers egy JavaScript nyelven készült programkönyvtár és felhasználása szintén JavaScript környezetben lehetséges. Segítségével többféle forrásból érkező térképeket jeleníthetünk meg dinamikus térképként a weboldalunkon. Számos szabványos formátumot (gml, kml, stb.) és szolgáltatást (WMS, WFS, stb.) támogat.

Az OpenLayers-en keresztül elérhető alaptérképre három különböző típusú objektum, hat különálló rétegen kerül ábrázolásra. Egy objektumtípus két rétegen kerülhet megjelenítésre. A megjelenítés elve mindkét réteg esetében ugyanaz, csak az ábrázolt objektumok számát tekintve van különbség. A rétegek láthatósága szabadon változtatható. Az objektumtípusok és a hozzájuk tartozó rétegek a következők:

- a megállók***:*** pont típusú statikus objektumok
  - **Megállók**: az összes megállót tartalmazó réteg, az oldal megnyitásakor egyszer kerül betöltésre a szerverről, és a továbbiakban változatlan
  - **Választott járat megállói**: nem tartalmazza az összes elemet, csak egy kiválasztott járat által érintett megállóhelyeket
- járatok útvonala*:* vonal típusú réteg, amely a kliens igényeit figyelembe véve kerül megjelenítésre
  - **Választott járat útvonala**: csak egy kiválasztott járat útvonalát tartalmazza
  - **Választott megálló útvonalai**: csak egy kiválasztott megállón áthaladó járatok útvonalát mutatja
- a pillanatnyilag közlekedő autóbuszok: pont típusú dinamikus objektumok, amelyek helyzete automatikusan változik, így szimulálva a buszok közlekedését.
  - **Járatok**: az összes úton lévő járatot mutatja
  - **Választott járat**: nem tartalmazza az összes elemet, csak egy kiválasztott járatot

Az objektumok térképi elhelyezéséhez szükséges adatok minden esetben az adatbázisból kerülnek lekérdezésre PHP parancsfájlokon keresztül. A PHP szkript tartalmazza

azt az SQL-lekérdezést, amivel az adatbázisból a szükséges információk kinyerhetők, illetve azt az utsítássort, amivel a lekérdezés eredményét meghatározott formátumban továbbítja az OpenLayers felé.

Ez a meghatározott formátum a **GeoJSON**, ami egy olyan kódolási nyelv, amely alkalmas a különböző földrajzi adatok strukturált leírására. A GeoJSON emberek által is könnyen értelmezhető, és általában a szerver és a kliens számítógép közötti adatátvitelre használják. A teljes GeoJSON adatok szerkezete megad egy, vagy több geometriai objektumot. A GeoJSON tartalmi leírása név / értékpárok strukturált halmazával tagolt információt hordoz [GeoJSON Dokumentáció]. Az alábbi példa egy megállóhely leírását tartalmazza GeoJSON kódolásban:

{"type":"Feature","geometry":{"type":"Point","coordinates":["17.567924","46.572769"]},"properties":
{"mkod":"96135","mnev":"Tibolamajor","rnev":"Tibolamajor","honossagi":"1","tipus":"1",
"telepuleskod":"87061","elhelyezked":"0","megjegyzes":"","lon":"17.567924","lat":"46.572769"}}

5.1. Az alaptérkép

A térképi felület kialakításához – amely a dolgozat korábbi részében 3-as számmal jelölt weblap-egységben zajlik – szükség van egy alaptérképre, amin a tematikus tartalom ábrázolásra kerül.

Az OpenLayers lehetőséget biztosít arra, hogy külső térképszerverek (Google Maps, OpenStreetMap, Bing stb.) térképeit integráljuk saját webes térképünkbe.

Webtérképemhez az OpenStreetMap (OSM) nevű térkép-szolgáltató biztosítja a háttértérkép adatait raszteres kép formájában. Az OSM egy önkéntes munkán alapuló, szabadon felhasználható, világméretű térképi adatbázis és internetes térképszolgáltatás. A 2004 nyarán indult projekt alapelve, hogy kizárólag szabad információforrások használhatók fel a munka során, és a keletkezett adatbázis is teljesen szabadon használható. Mára az OpenStreetMap információtartalma összemérhető bármelyik nagy kommersz internetes térképszolgáltatással [GEDE, 2012].

Az OSM alapréteg hozzáadását a térképhez a következő kódsor írja le:

```
...
<script src="http://openlayers.org/api/OpenLayers.js"></script>
<body>
<div style="width:75%; height:100%; position: absolute; right: 0; top:0"
id="terkep_helye"></div>
<script type="text/javascript">
// térkép létrehozása
 var map = new OpenLayers.Map('terkep_helye',
  {projection: "EPSG:3857", // EPSG:3857 = Spherical Mercator
   maxExtent: new OpenLayers.Bounds (1850000, 5720000, 2040000, 5960000)});

    map.addLayer(new OpenLayers.Layer.OSM()); //OSM réteg hozzáadása
...
```

5.2. A megállók megjelenítése

A megállóhelyek adatait az adatbázis *megallok* nevű táblája tartalmazza. A tábla *lon* és *lat* mezőiben találhatók a megálló hosszúság- és szélesség koordinátái, amik a megálló földrajzi helyzetét egyértelműen meghatározzák. Ezek az adatok, továbbá a megállók összes attribútuma az alábbi PHP szkripttel nyerhetők ki az adatbázisból:

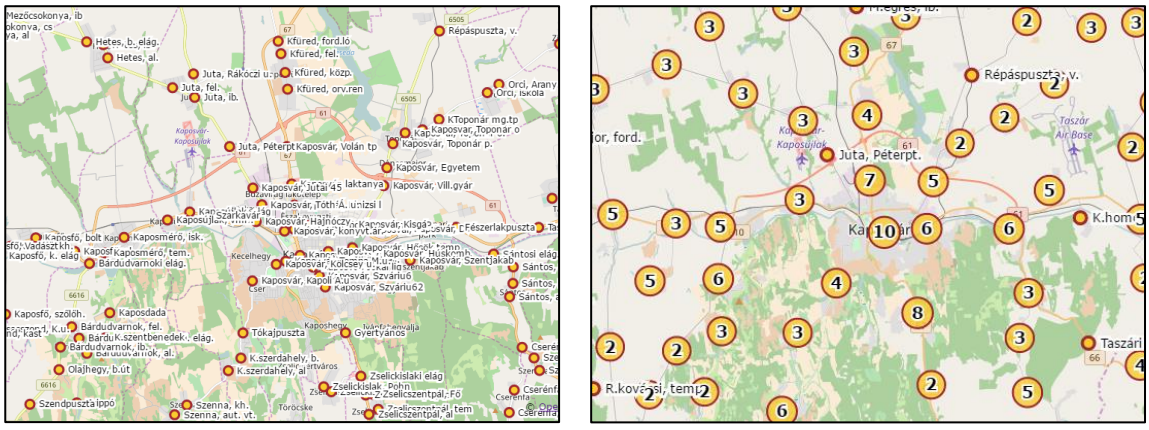
```
<?php
// json tömbben visszaadja az összes megálló adatát az adott táblában

require('db.php');

$db=mysql_connect('localhost',$dbuser,$dbpwd);
mysql_select_db($dbname,$db);
mysql_query('set names utf8');
$r=mysql_query("SELECT * FROM megallok WHERE mkod IN ".
              (SELECT megallokod FROM menetrend)");

if (mysql_error())
  die("{error:'".mysql_error()."'}");

while ($l=mysql_fetch_assoc($r))
  $p[]=array("type"=>"Feature","geometry"=>array("type"=>"Point", ".
"coordinates"=>array($l['lon'],$l['lat'])),"properties"=>$l);
print '{"type": "FeatureCollection","features":'.json_encode($p)."}";
mysql_close();
?>
```

A szkript a lekérdezés eredményét JSON kódolásúvá alakítja, amit az OpenLayers később értelmezni képes. A végeredményben csak azok a megállóhelyek szerepelnek, melyeket a járatok menetrendjét leíró táblába is tartalmaz, azaz létezik olyan járat amely a megállót érinti. Az átmenetileg nem üzemelő, végleg megszűnt, vagy utasforgalmat le nem bonyolító, technikai jellegű megállók nem kerülnek megjelenítésre. Az adatok térképi megjelenítését a dolgozat következő (A Cluster stratégia) alegységében fejtem ki.

5.2.1. A Cluster stratégia

Az adatbázisban 1603 db olyan megálló szerepel, melyekhez menetrend társítható. Ezeknek döntő többsége Somogy megye területén koncentrálódik. Éppen ezért megjelenítésük bizonyos nagyításokban (kis méretarányokon) a tematikus tartalmat túlzsúfolttá teszi. Erre a problémámra kínál megoldást az OpenLayers Cluster stratégiája.

A OpenLayers-ben definiált stratégiák olyan szabályrendszerek, amely a térkép adott rétegéhez (vagy rétegeihez) rendelhetők, és a rétegek objektumainak megjelenítésmódját határozzák meg. A Cluster stratégia összevonja a réteg azon objektumait, melyek között a távolság egy meghatározott értéknél kisebb, és egy önálló objektumként jeleníti meg őket (*8. ábra*).



8/a *8/b*

8. ábra: Ugyanannak a területnek az ábrázolása Cluster stratégia alkalmazása előtt (8/a) és után (8/b)

Megjelenítéskor különböző stílusszabályok szerint ábrázolom a különálló elemeket és a clusterezett objektumokat. A két objektumtípust szűrő használatával különböztetem meg:

```
...
// cluster szűrő

              var clusterFilter=new OpenLayers.Filter.Comparison({
                  type: OpenLayers.Filter.Comparison.GREATER_THAN,
                  property: 'count',
                  value: 1

              });
...
```

A filter kiszűri az elemek közül a clusterezett objektumokat, amennyiben azok több mint egy elemből állnak össze.

Ezután stílusszabályokat definiálok, először a clusterekre (clusternek tekintjük azokat az objektumokat, amelyekre az előbbi szűrő teljesül):

```
  ...
  // cluster stílusszabálya
              var clusterRule=new OpenLayers.Rule({
                  filter: clusterFilter, // csak arra vonatkozik, ami cluster
                  symbolizer: {
                      graphicName: 'circle',
                      label: '${count}',
                      ...
                  }});
  ...
```

Majd meghatározom a különálló elemek (minden, amire az előző szabály nem vonatkozik) stílusszabályát:

```
  ...
  // különálló elem stílusszabálya
              var elemRule=new OpenLayers.Rule({
                  elseFilter: true, // arra vonatk., amire a másik szab. nem
                  symbolizer: {
                      graphicName:'circle',
                                pointRadius: 6,
                                strokeColor:'#a12930',
                                ...
                  }});
  ...
```

Ezután létrehozom azt a rétegstílust, amire az előbbi szabályok érvényesek és egyúttal meghatározom az elemek ikonméretét, amely annál nagyobb minél több elem kerül összevonásra:

```
...
// az elemekre vonatkozó style
            var clusterStyle=new OpenLayers.Style(null,{
                context: {
                    // az ikonméretet kiszámító függvény
                    getSize: function(feature) {
                           return 12+Math.sqrt(feature.attributes.count); }
                    },
                rules: [elemRule,clusterRule]
            });
...
```

Végül létrehozom a megállók rétegét, amely a fenti rétegstílus szerint ábrázolja a megállókat, és a *megallok.php*-t dekódolva határozza meg azok helyzetét, a réteget pedig a térképhez adom:

```
...

// létrehozzuk a megállók réteget

     var megallok = new OpenLayers.Layer.Vector( "Megállók", {
            isBaseLayer: false,
            projection: 'EPSG:4326',
            strategies: [new OpenLayers.Strategy.Fixed(),
                         new OpenLayers.Strategy.Cluster(
                             {distance: 40,threshold: 2})],
            protocol: new OpenLayers.Protocol.HTTP(
              {url: "megallok.php",
                format: new OpenLayers.Format.GeoJSON()}),
            styleMap: new OpenLayers.StyleMap(clusterStyle),
                 labelSymbolizer: {maxResolution: 200}
        });

// hozzáadjuk a réteget a térképhez
     map.addLayer(megallok);
...
```

Az adott járat által érintett megállók ábrázolása ehhez képest annyiban különbözik, hogy a feldolgozásra kerülő PHP szkript SQL szintaxisában WHERE feltételek szűkítik a lekérdezés eredményét.

Az összes megálló megjelenítése az *1. sz. mellékleten* látható, az egy kiválasztott megálló adatainak ábrázolását pedig a *2. sz. melléklet* szemlélteti.

5.3. Az autóbusz-vonalak megjelenítése

Az adatbázisban csak a megállóhelyekre vonatkozóan rendelkezünk pontos földrajzi adatokkal. Egy adott buszjárat által érintett megállók és azok sorrendje az menetrend táblában került leírásra, viszont a megállók közötti útvonal geometriájáról konkrét információink nincsenek, így ezeket az adatokat a megállók elhelyezkedéséből kiindulva kell meghatározni.

5.3.1. Az elemi útvonalszakaszok

Első lépésként, a járatok útvonalát (és egyben a teljes vonalhálózatot) elemi szakaszkra bontom fel. Elemi szakasznak tekintem a menetrendi tábla szerint két egymást követő megálló közötti vonalszakaszt, amelyek geometriáját később külső útvonaltervező alkalmazás segítségével határozom meg. Az elemi szakaszokat, azok kezdő és végpontját (vagyis a két megállót) az alábbi SQL-parancs állapítja meg és képez belőlük új táblát az adatbázisban:

```
CREATE TABLE szakaszok (
    SELECT DISTINCT CONCAT (m1.mkod,'_', m2.mkod) as id,
        m1.mkod as mkod1, m1.lon as lon1, m1.lat as lat1,
        m2.mkod as mkod2, m2.lon as lon2, m2.lat as lat2
    FROM `menetrend` j1, `menetrend` j2,
        megallok m1, megallok m2
    WHERE j1.megallokod=m1.mkod
        AND j2.megallokod=m2.mkod
        AND j2.sorszam=j1.sorszam+1
        AND j1.vonalkod=j2.vonalkod
        AND j1.jaratszam=j2.jaratszam
        AND m1.lon IS NOT NULL
        AND m2.lon IS NOT NULL)
```

A létrejött táblát –amely 4555 db rekordot tartalmaz– egy további linestring típusú mezővel (*geom*) bővítem, amely a szakaszok geometriáját foglalja majd magában:

```
ALTER TABLE `szakaszok` ADD `geom` LINESTRING
```

A geometriákat az **Open Source Routing Machine** (OSRM) nevű alkalmazással hozom létre. Az OSRM egy C++ nyelven írt, nyílt forráskódú útvonaltervező motor, amelyet olyan módon terveztek, hogy az az OpenStreetMap Project-ből származó térképadatokkal optimálisan működjön. A OSRM nagy előnye, hogy rendkívül gyors útvonal-lekérést biztosít. A lekérdezés gyorsasága abban rejlik, hogy az OSRM nem az A* algoritmust használja, mint a legtöbb útvonal-tervező szolgáltatás, hanem az annál sokkal gyorsabb úgynevezett Összehúzási Hierarchiák (Contraction Hierarchies) módszerét alkalmazza. Az Összehúzási Hierarchiák módszere egy olyan alkalmazott matematikai eljárás, amely lehetővé teszi, hogy az OSRM-en futtatott útvonal-lekérdezések időtartama 1 ms alá csökkenjen [WIKIPEDIA].

Az elemi szakaszokra történő útvonaltervezés PHP parancsfájlon (*utratesz.php*) keresztül fut le. A szkript először kinyeri az adatbázisból azoknak a szakaszoknak minden adatát, amelyekhez nem tartozik geometria:

```
...
<?php
require('db.php');
$db=mysql_connect('localhost',$dbuser,$dbpwd);
mysql_select_db($dbname,$db);
mysql_query('set names utf8');
$r=mysql_query("SELECT * FROM szakaszok WHERE geom IS NULL");
while ($l=mysql_fetch_assoc($r)) $p[]=$l;
print "szakaszok=".json_encode($p).";";
mysql_close();
?>
...
```

A lekérdezés eredményét felhasználva a szkript ezután egy olyan függvényt (*kodol*) futtat, amely a szakasz kezdő- és végpontjaira útvonaltervezést végez az OSRM alkalmazással.

Az OSRM a lekérdezés eredményét JSON formátumban adja vissza, amelyet egy a függvénybe ágyazott külső PHP parancsfájl (*szakaszgeom.php*) feltölt a *szakaszok* tábla megfelelő mezőjébe.

```
<?php
// geometria feltöltése az adatbázisba
require('db.php');

$db=mysql_connect('localhost',$dbuser,$dbpwd);
mysql_select_db($dbname,$db);
mysql_query('set names utf8');
$q="UPDATE szakaszok SET ".
   "geom=GeomFromText('{$_GET['g']}') ".
   "WHERE id='{$_GET['id']}'";
print "$q\n";
$r=mysql_query($q);
print mysql_error();
mysql_close();
?>
```

(szakaszgeom.php)

A teljes *kodol* függvény pedig:

```
...
function kodol() {
  fi1=szakaszok[i].lat1;la1=szakaszok[i].lon1;
  fi2=szakaszok[i].lat2;la2=szakaszok[i].lon2;
   var url="http://router.project-osrm.org/route/v1/".
       url+="driving/"+la1+","+fi1+";"+la2+","+fi2+"?geometries=geojson";
  x.open('get',url,false);
  x.send();
  var res=JSON.parse(x.responseText);
  console.log(res);
  if (res.code=="Ok") {
      // rendben van, lett eredmény
      var ls="";
      var cr=res.routes[0].geometry.coordinates;
      for (var j=0;j<cr.length;j++)
          ls+=(ls!=""?",":"")+cr[j][0]+" "+cr[j][1];
      ls="LINESTRING("+ls+")";
      console.log(ls);
       x.open("get","szakaszgeom.php?id="+szakaszok[i].id+"+&g="+ls,false);
      x.send();
      console.log(x.responseText);
      i++;
      if (i<n)
          setTimeout(kodol,2000);
  }
```

Az útvonalak egy adott járatra vagy buszmegállóra vonatkozóan jeleníthetők meg. Mivel az összes elemi szakasz együttes ábrázolása Somogy megye közel teljes úthálózatát lefedné, ezért az összes útvonal egy rétegen történő megjelenítésének (mint vonalhálózati térkép) gyakorlati haszna nincs.

A megjelenítést a következő forráskód az egy járathoz tartozó útvonal példáján keresztül mutatja be:

```
<?php
// egy adott járat útvonala, vonalkód és járatszám alapján

...
//járat elemi szakaszainak lekérdezése
$r=mysql_query("SELECT astext(sz.geom) as g FROM ...");
if (mysql_error())
  die("{error:'".mysql_error()."'}");
$c="";
while ($l=mysql_fetch_assoc($r)) {
    if ($c!="") $c.=",";
    $c.=substr($l['g'],11,strlen($l['g'])-12);
    }
$c=str_replace(",","],[",$c);
$c="[".str_replace(" ",",",$c)."]";
print '{"type":"Feature","geometry":{"type":"LineString", ".
""coordinates":['.$c.']}}';
mysql_close();
?>
```

(jarat_utvonal.php)

A *jarat_utvonal.php*-ba ágyazott SQL szintaxis lekérdezi az adtabázisból a útvonal összeállításához szükséges elemi szakaszokat a megfelelő menetrendi sorrendben, majd az elemi szakaszok töréspontjainak koordinátáiból egy új linestringet állít elő. Mivel a MySQL más módon tárolja a görbe geometriáját, mint ahogy az GeoJSON kódolás szerint értelmes lenne ezért a PHP karakterlánc-függvényeinek alkalmazásával a geometriákat a szkriptben a megfelelő formátummá alakítom.

Példa az adatbázisban tárolt formátumra:

```
('LINESTRING(17.790949 46.354327,17.7849 46.355696,17.779683 46.345351,... )',0)
```

Példa a GeoJSON linestring formátumra:

```
({„type”:”Feature”,”geometry”:{„type”:”LineString”,”coordinates”:
    [[17.790949,46.354327],[17.7849,46.355696],[17.779683 46.345351],,...]}})
```

A PHP eredményeként kapott GeoJSON kódot az OpenLayers dolgozza fel.

Egy útvonal kirajzolása mindig kliens oldali kezdeményezésre zajlik le. Vagyis szükséges valamilyen tevékenység a felhasználó részéről, amely konkretizálja, hogy milyen paraméterekkel rendelkező objektum jelenjen meg és egyúttal el is indítja a folyamatot. A közlekedő járatok példájánál maradva, ez a felhasználói aktivitás, a járat ikonjára való kattintás. Az ilyen, és ehhez hasonló tevékenységeket (objektumra kattintás, kurzor objektum fölé vitele, objektumon kívüli kattintás stb.) az OpenLayers eseménykezelői értelmezik. Eseménykezelőket az alábbi módon hozhatunk létre:

```
jaratok.events.register('featureclick',map,klikk2);
jaratok.events.register('featureover',map,over);
jaratok.events.register('featureout',map,out);
```

Az eseménykezelő érzékeli a felhasználó tevékenységét, és hatására a hozzárendelt funkciót lefuttattja. A közlekedő járat ikonjára való kattintás egy olyan függvény futását indítja el, ami egyebek mellett a járathoz tartozó útvonal vizualizálását is végzi. Az vonal kirajzolásáért az alábbi függvényrészlet felel:

```
...
OpenLayers.Request.GET({
   url: 'http://terkeptar.elte.hu/~volan/jarat_utvonal.php
         ?vkod='+attr2.vonalkod+'&jszam='+attr2.jaratszam,
   async: true,
   success: function(e) {
       var features = new OpenLayers.Format.GeoJSON().read(e.responseText);
       for(var i= 0; i < features.length; i++){
         features[i].geometry.transform("EPSG:4326","EPSG:3857");
       }
       vonal.addFeatures(features);
       vonal.refresh();
...
```

A *jarat_utvonal.php*-vel ábrázolt görbe a vonal nevű rétegen válik láthatóvá.

A megállókon átmenő vonalak ábrázolásának elve ugyanez, ahol a folyamatot a megálló ikonjára való kattintás indítja el, eredménye pedig több útvonal egy új rétegen.

A kiválasztott megállót érintő járatokat egy PHP válogatja le az adatbázisból, majd egy JavaScriptben írt ciklus „adagolja” a járatok adatait (vonal- és járatszám) a *jarat_utvonal.php*-nek, amely mindegyik járat útvonalát megjeleníti.

5.4. A pillanatnyilag közlekedő járatok megjelenítése

A harmadik rétegcsoport, amely a térképen elhelyezésre kerül a pillanatnyilag közlekedő járatokat ábrázoló rétegek, ami a webtérkép járműkövetési funkcióját hivatott szimulálni. Szimulációról azért beszélünk, mert –ahogyan az a dolgozat címében is szerepel– a járatok vizualizációja a mentrendi adatokat alapul véve történik interaktív módon. Vagyis feltételezzük, hogy a járatok a menetrendet pontosan betartva, késés nélkül közlekednek, ami viszont a valós idejű járműkövetés elveinek nem tesz eleget.

Egy buszjárat minden esetben a hozzá tartozó útvonal mentén közlekedik, amiből következik, hogy adott pillanatban az útvonalat felépítő elemi szakaszok valamely pontján helyezkedik el. Az elemi szakaszok kezdő- és végpontjai a járat megállói, amelyekhez a menetrendet leíró táblában indulási és érkezési idők tartoznak. Ezekből az időadatokból és a vonalszakaszok geometriájából a járat helyzete pontosan kiszámítható.

A számításokat az adatbázis-szerver végzi. A számítás elvégzéséhez új függvények definiálása szükséges a phpMyAdmin felületén.

5.4.1. A jármű helyzetét kiszámító függvények

Az elemi vonalszakasz egy olyan linestring típusú görbe, amit az azt felépítő töréspontok koordinátája határoz meg egyértelműen. A görbe két töréspontját egy egyenes vonal köti össze, tehát a görbét még kisebb egyenes vonaldarabok alkotják. Hogy megállapíthassuk, hogy a jármű az elemi vonalszakasz melyik pontján tartózkodik, ismernünk kell a görbét felépítő egyenesek hosszát. Ennek kiszámításhoz definiálom az első függvényt (*PDISTANCE*), amely bemeneti paraméterként megkapja a vonalszakasz két töréspontját, mint pont típusú adatot, a függvény kimeneti értéke pedig egy double típusú szám, mint az egyenes vonaldarab hossza:

```
DELIMITER $$
CREATE FUNCTION PDISTANCE(p1 point, p2 point)
    RETURNS double DETERMINISTIC
    BEGIN
        declare dx double;
        declare dy double;
        set dx=X(p1)-X(p2);
        set dy=Y(p1)-Y(p2);
        RETURN sqrt(dx*dx+dy*dy);
    END$$
```

Egy további függvény pedig –felhasználva az előző *PDISTANCE*-t– megállapítja, hogy egy elemi görbe bizonyos arányú felosztásánál, melyik pont található:

```
DROP FUNCTION PointAlongLine;
DELIMITER $$
CREATE FUNCTION PointAlongLine(ls linestring, p double)
    RETURNS point DETERMINISTIC
    BEGIN
        declare l double;
        declare sl double default 0;
        declare dl double default 0;
        declare i int default 1;
        declare x double;
        declare y double;
        declare r double;
        set l=glength(ls);
        while i<numpoints(ls) and sl+dl<=l*p do
            set sl=sl+dl;
            set dl=pdistance(pointn(ls,i),pointn(ls,i+1));
            set i=i+1;
        end while;
        set r=(l*p-sl)/dl;
        set x=x(pointn(ls,i-1))*(1-r)+x(pointn(ls,i))*r;
        set y=y(pointn(ls,i-1))*(1-r)+y(pointn(ls,i))*r;
        return geomfromtext(concat("point(",x," ",y,")"));
    END$$
DELIMITER ;
```

A függvény bementeni paraméterei egy linestring, és egy double típusú arányszám. Ez az arányszám majd a két megállóhoz (linestring kezdő- és végpontja) tartozó érkezési és

indulási idők továbbá az aktuális időpont aránya lesz. A függvény kimenti értéke pedig az a pont, ahol a jármű tartózkodik.

5.4.2. Interaktív vizualizáció a webtérképen

Mivel a közlekedő buszok pont típusú objektumként jelennek meg a térképen, ábrázolásuk a megállókhoz hasonló módon történik (Cluster stratégia nélkül). Helyzetüket és minden egyéb attribútumukat egy PHP fájlba (*kozlekedo_jaratok.php*) ágyazott SQL-lekérdezés adja. Az lekérdezés felhasználja a 5.4.1. alfejezetben ismertetett függvényeket, a *PointAlongLine* függvényhez pedig ezúttal meghatározásra kerül az időadatokból alkotott arányszám:

```
...
$r=mysql_query(  "SELECT j1.vonalkod as vonalkod, ".
                 "j1.jaratszam as jaratszam, jr.kkorlat, ".
                 "m1.mnev honnan, m2.mnev hova, j1.indulas, j2.erkezes, ".
                 "Y(PointAlongLine(geom,((CURTIME()-j1.indulas) ".
                       "/(j2.erkezes-j1.indulas))) as lat, ".
                 "X(PointAlongLine(geom,((CURTIME()-j1.indulas)/ ".
                       "(j2.erkezes-j1.indulas))) as lon ".
    "FROM  `szakaszok`, `menetrend` j1, ".
           "`menetrend` j2, `megallok` m1, ".
           "`megallok` m2, `jarat` jr ".
    "WHERE j1.vonalkod=j2.vonalkod ".
           "AND j1.jaratszam=j2.jaratszam AND j2.sorszam=j1.sorszam+1 ".
           "AND j1.indulas<=(CURTIME()) AND j2.erkezes>=(CURTIME()) ".
           "AND j1.megallokod=m1.mkod AND j2.megallokod=m2.mkod ".
           "AND j1.vonalkod=jr.vkod AND j1.jaratszam=jr.jszam ".
           "AND id=CONCAT(m1.mkod,'_',m2.mkod) ".
```

A *kozlekedo_jaratok.php* térképi ábrázolása a már ismertetett módon történik:

```
...
// létrehozzuk a járatok rétegét
     var jaratok = new OpenLayers.Layer.Vector( "Járatok",
           {isBaseLayer: false,
            projection: 'EPSG:4326',
            strategies: [new OpenLayers.Strategy.Fixed()],
            protocol: new OpenLayers.Protocol.HTTP(
              {url: "kozlekedo_jaratok.php",
                format: new OpenLayers.Format.GeoJSON()}),
            styleMap: new OpenLayers.StyleMap({
                externalGraphic:'bus_icon.png',
                pointRadius: 12,
                        opacity: 1,
                        hoverPointRadius: 20,
                        cursor: "pointer",
            })
        });
...
```

A réteg objektumai dinamikus térképi elemek, helyzetük pillanatról pillanatra változik. Ez a változás könnyen szemléltethető, ha a réteg bizonyos időközönként automatikusan frissül. A réteg újratöltésére 10 mp-es intervallumot állítok be:

```
...
setInterval(function() {
        jaratok.refresh();
        console.log('na');
    },10000);
...
```

Egy pillanatnyilag közlekedő járat megjelenítése annak minden adatával együtt a *3 sz. mellékleten* tekinthető meg.

6. Továbbfejlesztési lehetőségek

Az elkészült webtérkép számos olyan lehetőséget rejt magában, ami különböző okok miatt még nem került megvalósításra. Az egyik legfőbb ok, ami gátat szabott munkám során az a megfelelő pontosságú adatok hiánya.

Az adatbázisban a megállóhelyek helyzetének meghatározása csupán egy koordinátapárral történik. Egy megállóhely (logikai megállóhely) viszont a legtöbb esetben több kocsiállásból (fizikai megállóhelyből) áll, melyek között jelentősebb távolság is lehet. Az adatbázis kocsiállás alapúvá tétele a webtérkép egészét pontosabbá tenné, mivel minden más objektum is a megállók helyzetéből kerül levezetésre. Pontosabbá tenné a járatok útvonalának meghatározását, és ezzel kiküszöbölné az útvonaltervezés automatikus jellegéből fakadó hibákat. Az utas számára is egyszerűbbé tenné a megfelelő megállóhely terepi azonosítását, tekintve, hogy a kocsiállásokról jelenleg csak szöveges formában kap tájékoztatást a felhasználó.

Nem kerültek kialakításra a felületen utazástervezési lehetőségek. Bár a dolgozat témájához szorosan nem kapcsolódik ezeknek a funkcióknak a tárgyalása, de a weboldalnak –amennyiben utastájékoztató jellegét növelni akarjuk– ezt a feladatot is el kell látnia.

A térkép jelenleg csak az aktuális (mai) napra vonatkozó információkat képes megjeleníteni. Ugyancsak a felhasználók kényelmét segítheti, ha pl. egy megállóhelynek szabadon választott napra is megtekinthetjük a menetrendjét.

7. Összegzés

Dolgozatom célja egy olyan webtérkép megalkotása volt, amely átfogó képet nyújt Somogy megye helyközi autóbusz-közlekedéséről, interaktív módon, pusztán menetrendi adatok felhasználásával.

A dolgozat bevezetésében igyekeztem rávilágítani az utastájékoztató webtérképek jelentőségére, továbbá arra, hogy miért kaptak különös figyelmet az ezeket korszerűsítő fejlesztések a hazai személyszállító vállalatoknál.

Munkám első részében röviden bemutattam a hazánkban jelenleg is üzemben lévő online utastájékoztató térképeket, hogy az így szerzett tapasztalataimat felhasználva alakíthassam ki saját térképem tematikus tartalmát.

A következő fejezetben azzal foglalkoztam, hogy a DDKK Zrt.-től kézhez kapott adatokból hogyan hoztam létre azt az adatbázist, ami később a térképi megjelenítést alapját képezte.

Ezután bemutattam a térképet tartalmazó weboldal megalkotásának eszközeit, majd a következő fejezetekben részletesen leírtam a térkép tematikus tartalmának kialakítását, az adatok interaktív ábrázolásának módját, és további fejlesztési lehetőségeit.

A dolgozat eredményeként létrejött weboldal a ***http://terkeptar.elte.hu/volan*** címen érhető el.

8. Irodalomjegyzék

Lynn BEIGHLEY, Michael MORRISON: (2009): *Agyhullám: PHP & MySQL*, Kiskapu Kiadó, Budapest, pp. 37-38.

ELEK István (2011): *Adatbázisok, térképek, információs rendszerek*, ELTE Eötvös Kiadó, Budapest, pp. 9-11.

GEDE Mátyás (2012): *Az OpenStreetMap*, Eötvös Loránd Tudományegyetem, Informatikai Kar, http://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011_0056_IK_osmap/lecke1_lap1.html

GEOJSON (2008): *Dokumentáció*, http://geojson.org/geojson-spec.html (utolsó letöltés: 2017.05.11.)

GEVAPC Felhasználói Tudástár (2011): *Excel – CSV fájlok*, 2011.03.21, http://gevapc.hu/story/excel-csv-fajlok (utolsó letöltés: 2017.05.11.)

Michael MONCUR (2002): *Tanuljuk meg a JavaScript használatát 24 óra alatt*, Kiskapu Kiadó, Budapest, p. 13.

PACZONA Zoltán (2001): *HTML technikák a gyakorlatban*, Computer Panoráma Kiadó, Budapest, p. 40.

VILÁGGAZDASÁG (2014): *Volán: összevonás idén, ezer új busz a következő években*, Világgazdaság (online), http://www.vg.hu/vallalatok/kozlekedes/volan-osszevonas-iden-ezer-uj-busz-a-kovetkezo-evekben-429808

VOLÁNBUSZ Zrt.: *Üzletszabályzat*, https://www.volanbusz.hu/hu/tarsasagunkrol/uzletszabalyzat/00 (utolsó letöltés: 2017.05.11.)

WIKIPEDIA: *Open Source Routing Machine*, https://en.wikipedia.org/wiki/Open_Source_Routing_Machine (utolsó letöltés: 2017.05.11.)

9. Ábrajegyzék

Az ábrajegyzékben forrás nélkül feltüntetett ábrák saját képek és képernyőmentések.

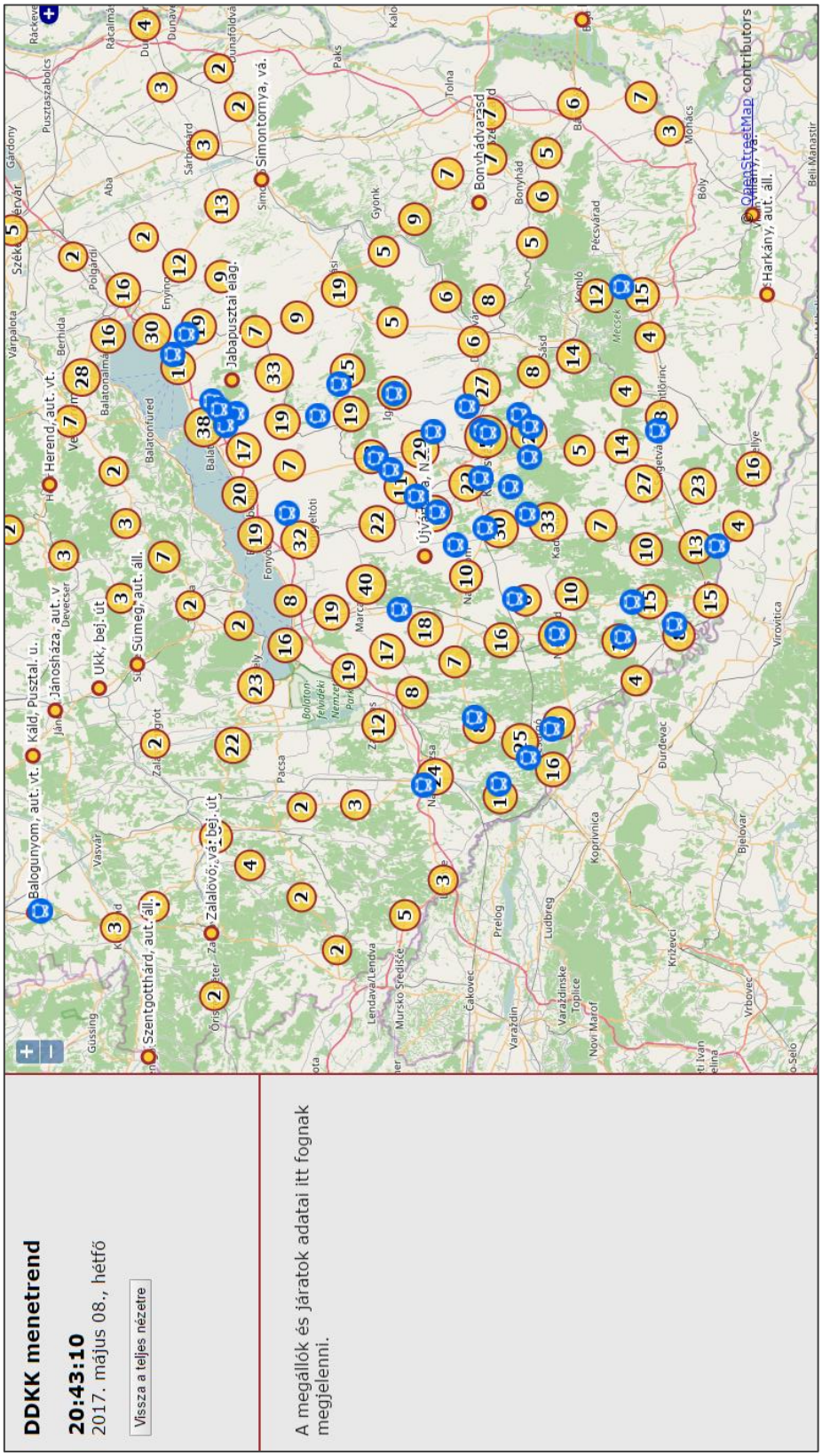
10. Köszönetnyilvánítás

Ezúton szeretném megköszönni témavezetőmnek, Gede Mátyásnak, hogy rám áldozott idejével, hasznos tanácsaival és útmutatásával segítette a dolgozat létrejöttét.

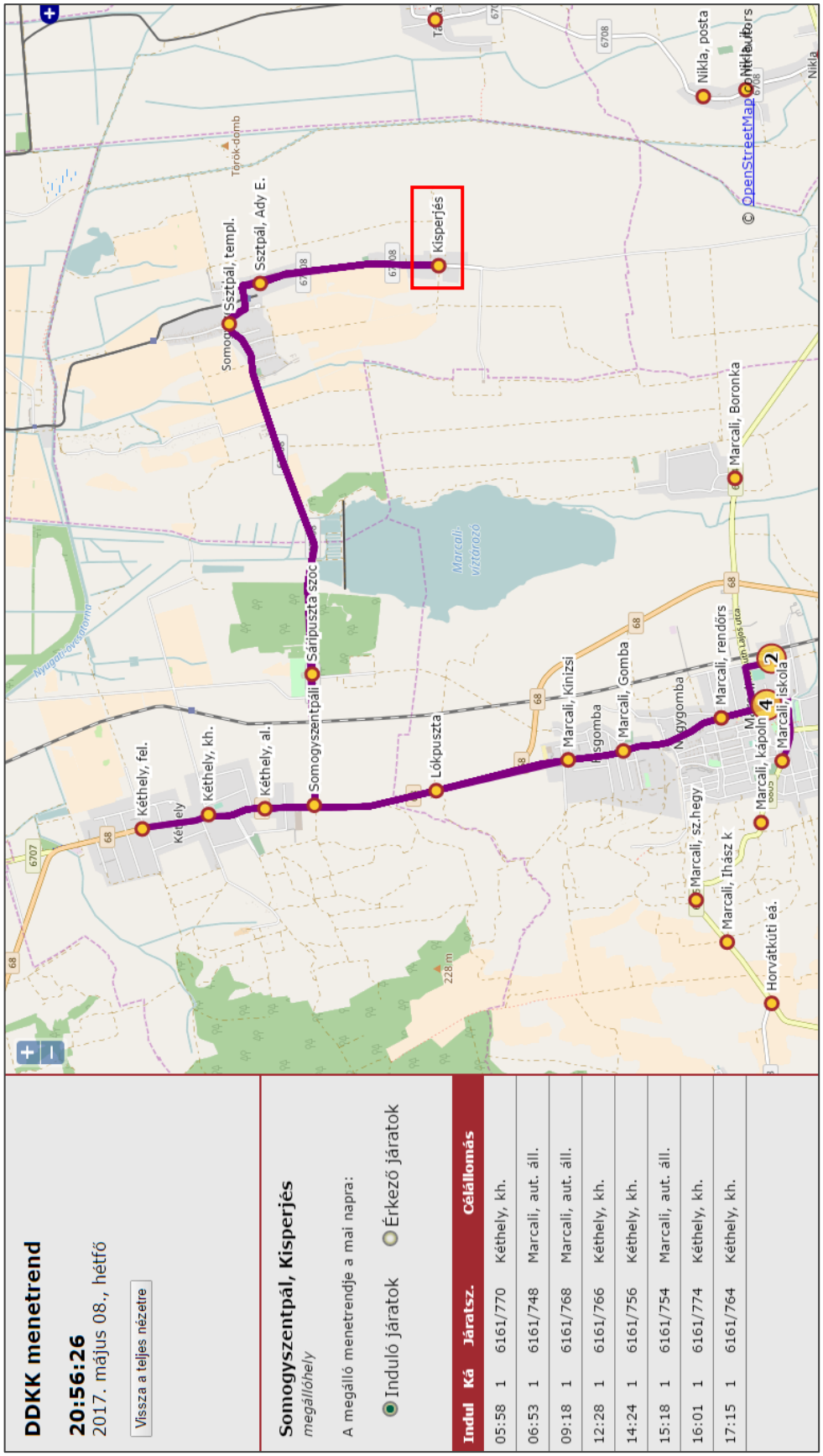
Köszönettel tartozom továbbá a Dél-dunántúli Közlekedési Központ Zrt. Somogy megyei Szolgáltatási Központjának és különös tekintettel édesapámnak, Eigner Jánosnak, a szolgáltatási központ vezetőjének, aki a dolgozat alapjául szolgáló adatokat a rendelkezésemre bocsátotta, és közlekedésszakmai kérdésekben nyújtott értékes segítséget.

11. Mellékletek

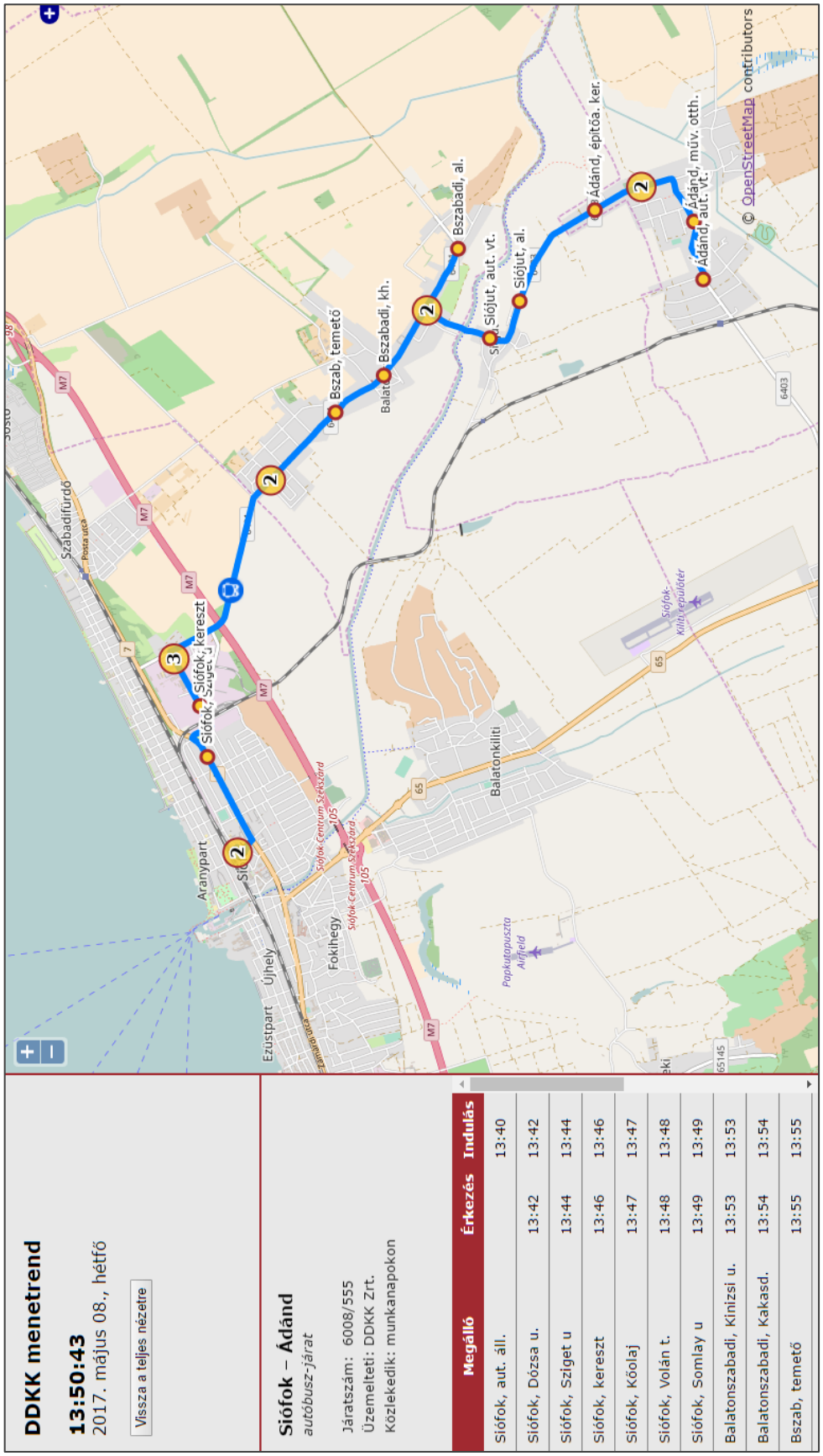
1. melléklet: A megállóhelyek és a közlekedő járatok áttekintő nézete (alapnézet).



2. sz. melléklet: Megállóhely (Somogyszentpál, Kisperjés) ábrázolása, annak mentrendjével, és a megállót érintő járatok útvonalával.



3. sz. melléklet: Járat (Siófok–Ádánd) megjelenítése a hozzátartozó útvonallal, megállóhelyekkel és egyéb általános információkkal a weboldalon.



Nyilatkozat

Alulírott, Eigner Péter nyilatkozom, hogy jelen szakdolgozatom teljes egészében saját, önálló szellemi termékem. A szakdolgozatot sem részben, sem egészében semmilyen más felsőfokú oktatási vagy egyéb intézménybe nem nyújtottam be. A szakdolgozatomban felhasznált, szerzői joggal védett anyagokra vonatkozó engedély a mellékletben megtalálható.

A témavezető által benyújtásra elfogadott szakdolgozat PDF formátumban való elektronikus publikálásához a tanszéki honlapon

HOZZÁJÁRULOK NEM JÁRULOK HOZZÁ

Budapest, 2017. május 15.

…….
a hallgató aláírása