EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR

Az openrouteservice.org lehetőségeinek bemutatása egy komplex webes útvonaltervező alkalmazással

DIPLOMAMUNKA
TÉRKÉPÉSZ MESTERSZAK

Készítette:
Balla Dániel

Témavezető:
Dr. Gede Mátyás
egyetemi docens
ELTE Térképtudományi és Geoinformatikai Intézet

Budapest, 2022

EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR
TÉRKÉPTUDOMÁNYI ÉS GEOINFORMATIKAI TANSZÉK

DIPLOMAMUNKA TÉMABEJELENTŐ

Hallgató adatai:
Név: Balla Dániel
Neptun kód: G3WSRX

Képzési adatok:
Szak: térképész, mesterképzés (MA/MSc)
Tagozat: Nappali

Belső témavezetővel rendelkezem

Témavezető neve: *Dr. Gede Mátyás*
munkahelyének neve: ELTE IK Térképtudományi és Geoinformatikai Tanszék
munkahelyének címe: 1117, Budapest, Pázmány Péter sétány 1/A.
beosztás és iskolai végzettsége: egyetemi docens, PhD

A diplomamunka címe: Az openrouteservice.org lehetőségeinek bemutatása egy komplex webes útvonaltervező alkalmazással

A diplomamunka témája:
(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben diplomamunka témájának leírását)

Az openrouteservice.org az OpenStreetMap adatbázisára alapozva számos hasznos geoinformatikai szolgáltatást kínál, ingyenesen. A dolgozat célja egy olyan komplex útvonaltervező alkalmazás készítése, mely kombinálja az útvonaltervező, a magasságiadat- és az izokrónkészítő szolgáltatásokat, elsősorban gyalogos és kerékpáros útvonalak tervezésére. Az elkészülő alkalmazásnak mobil és asztali gépes környezetben is jól használhatónak kell lennie.

Budapest, 2021. 11. 29.

Tartalomjegyzék

Abstract

Demonstration of features provided by service openrouteservice.org, using a complex, web-based route planning application

Openrouteservice.org (ORS), based on the spatial database of OpenStreetMap, provides numerous spatial services for free, which can be implemented in third-party applications. This thesis aims to present some of the features Openrouteservice has to offer, implemented in a complex, web-based route planning application, whose development is also discussed. ORS offers various services: route planning between waypoints, generating isochrone polygons, geocoding, elevation data and POI-retrieval, generating time/distance matrices, vehicle optimization (using VROOM). The developed application, which is based on JavaScript (besides HTML for UI and CSS for styling), constructs and sends requests to ORS, interprets received (mainly GeoJSON) data and visualizes these data on the user interface (UI) and on an embedded Leaflet map object, in an interactive way. Features of the application include: car-, bicycle- or walking-based route planning between up to three waypoints (using recommended, fastest or shortest route planning approaches), with an itinerary containing navigation steps of the route; displaying elevation profile using elevation data that is applicable for the route; retrieving and display of isochrones and isodistance polygons; map measurements of distance and bearing; geolocation; geocoding and reverse geocoding. Some geoprocessing was done in JavaScript using the open-source Turf.js library. The accompanying interactive UI was created with responsive web design in mind. Some feature ideas are also discussed, which could be implemented in the future (e.g., defining avoidable areas while route planning, or support of GPX/KML import/export).

The developed web application can be accessed online at:

https://balladaniel.github.io/ors/

Note: the web version might differ compared to the version produced at the time of publishing this thesis (15 May 2022)

Keywords: web mapping, interactive map, isochrone, isodistance, elevation profile, geocoding, geolocation, route planning, responsive web design, GeoJSON, API, Openrouteservice, OpenStreetMap, Leaflet, Turf.js, JavaScript

1. Bevezetés

Az OpenStreetMap egy 2004-ben indult, nyíltan hozzáférhető és szerkeszthető, térbeli adatokat gyűjtő projekt, mely adatait letölthető, API-on keresztül lekérhető és térképcsempék formátumában is elérhetővé teszi. Az OSM kollaboratív természetéből adódó, potenciálisan kérdéses adatminőséget több kutatás is vizsgálta, pl. (Mooney & Corcoran, 2012), (Neis, Zielstra, & Zipf, 2012) és (Zielstra & H. Hochmair, 2012). Ezen kutatások alapján az OSM adatbázisa meglehetően teljes és naprakész, általában ugyanolyan részletességű vagy részletesebb, mint az akkor elérhető üzleti alapú adatkészletek. A friss, naprakész adatbázis vélhetően az OSM népszerűségének és főként az azt szerkesztő felhasználók számának köszönhető, mely a 2010-es évekre jelentősen megnőtt: 2011 decemberében félmillió, 2022 márciusában több, mint 8 millió regisztrált felhasználóval rendelkezett (OpenStreetMap stats, 2022). Az OSM adatbázisát napjainkban számos weboldal, térképszolgáltatás, útvonaltervező és mobilos alkalmazás használja térképi alapként.

Az Openrouteservice (ORS) az OpenStreetMap adatbázisára alapozva számos hasznos geoinformatikai szolgáltatást kínál, melyek ingyenesen elérhetőek és használhatóak. Ilyen szolgáltatás például az útvonaltervező, vagy az izokrónkészítő szolgáltatás. Az ORS által kínált lehetőségek közé tartozik a magasságiadat-lekérés, idő- és távolságmátrix generálása, geokódolás és POI-k kigyűjtése is. Ezek az ORS által fejlesztett és üzemeltetett szolgáltatások feldolgozzák az érkező, megfelelő szintaxisú lekéréseket, az OSM adatbázisán futtatják algoritmusaikat, majd a szolgáltatott adatot válaszként küldik vissza (pl. útvonal polyline formátumban vagy fordított geokódolás esetén a megadott pont környékén lévő elemek pontként). (Openrouteservice.org, 2022)

A dolgozatom célja egy olyan komplex, webes útvonaltervező alkalmazás készítése, mely az Openrouteservice által szolgáltatott adatokat dolgozza fel, és kombinálja az útvonaltervező, a magasságiadat- és az izokrónkészítő szolgáltatásokat, elsősorban gyalogos és kerékpáros, valamint autós útvonalak tervezésére. Dokumentálom az alkalmazás felépítését, a legfontosabb (technikai és geoinformatikai) folyamatok és függvények szerkezetét és működését, valamint a fejlesztés alatt megnyilvánuló problémákat, azok potenciális megoldásait. A HTML, CSS és JavaScript nyelveken írt alkalmazás tervezésekor kitűzött cél volt, hogy felületének kialakítása reszponzív legyen (annak érdekében, hogy mobil és asztali gépes környezetben is jól használható legyen).

2. OpenStreetMap

Az OpenStreetMap (https://www.openstreetmap.org/) egy nyíltan hozzáférhető és szerkeszthető, térbeli adatokat gyűjtő és megjelenítő térképszolgáltatás és projekt, melyet Steve Coast alapított 2004-ben. A projekt célja, hogy egy olyan térbeli adatbázist hozzon létre, melyet bárki szabadon szerkeszthet és módosíthat, és amely szabadon, ingyenesen elérhető bárki számára, bármilyen felhasználásra. A projekt által kínált adatok "nyílt adatok" (*open data*), melynek licenszét az OSM az Open Data Commons Open Database License (ODbL) licensszel a következőképp határozta meg: bárki, bármilyen célra felhasználhatja az adatokat, feltéve, hogy feltünteti és hivatkozza az OpenStreetMap-et és annak szerkesztőit (OpenStreetMap contributors). Az OSM adatbázisát számos weboldal, térképszolgáltatás, útvonaltervező és mobilos alkalmazás használja térképi alapként (OpenStreetMap, 2022). Amellett, hogy az OSM API-hozzáférést is kínál az adatokhoz, a projekt által tárolt heti frissességű adatokat teljes egészében is le lehet tölteni tömörített XML és PBF formátumban a Planet OSM weboldalról (https://planet.openstreetmap.org/), viszont ez rendkívüli méretű anyag (2022. március elején a tömörített XML 113 GB (kicsomagolva 1552 GB), a PBF fájl 62 GB méretű volt). Számos weboldal kínál lehetőséget arra, hogy a teljes OSM adatbázis csak egy részét, kivonatát töltsük le. Ezek a weboldalak perces, órás, napos vagy heti rendszerességgel frissítenek, hogy mindig az OSM lehető legfrissebb állapotát kínálják (OpenStreetMap wiki, 2022).

Az adatok eléréséhez nem szükséges regisztráció, a szerkesztéshez viszont igen. Az önkéntes szerkesztők kézi GPS-egységek által generált GPX fájlokat tölthetnek fel (pl. egy útvonal térképezésére), de akár helyismeretükre vagy megfigyelésre is hagyatkozhatnak, és az alapján is szerkeszthetnek (pl. egy új POI elhelyezése, vagy egy meglévő attribútumainak frissítése). A szerkesztéshez légi- vagy műholdfelvételeket is lehet referenciaként használni, ilyenek például az ESRI által szolgáltatott, 15 méteres felbontású TerraColor, a 2,5 méteres felbontású SPOT, vagy egyéb forrásból származó, akár 5-20 centiméteres felbontású műholdképek (Gravois, 2017). A pontok, útvonalak vagy poligonok megrajzolása után a szerkesztőnek lehetősége adódik arra, hogy attribútumokat rendeljen ezekhez, esetleg más térképi elemekhez való kapcsolatokat definiáljon. A szerkesztés mentése után az új térképi adatok bekerülnek az OSM adatbázisába, és egy kis idő elteltével az OSM által újonnan generált térképcsempéken is megjelennek.

Mivel az adatbázist bárki szerkesztheti, felmerül a kérdés, milyen az adatok minősége, helyessége. A projekt nyitott, kollaboratív természetéből adódóan (hasonlóan a Wikipedia-hoz) nyilvántartja a változtatásokat, és lehetővé teszi, hogy ezeket a változtatásokat vissza lehessen

fordítani (ennek következtében a vandalizmus és a hibás adatok is könnyen kiküszöbölhetőek). Az OSM adatainak minőségét több kutatás vizsgálta, pl. (Mooney & Corcoran, 2012). Egy 2011-ben végzett kutatás alapján az OSM útadatbázisa Németország területén 27%-kal több információt tartalmazott, mint a TomTom akkori adatkészlete (Neis, Zielstra, & Zipf, 2012). Zielstra és Hochmair 2012-es kutatása azt igazolta, hogy az USA és Németország városainak gyalogoshálózatát elemezve, kb. 1000 legrövidebb út legenerálása után, az OSM adatbázisán rövidebb útvonalakat sikerült találni, mint két kereskedelmi adatkészleten, vélhetően a teljesebb gyalogoshálózat megléte miatt (Zielstra & H. Hochmair, 2012). Az OSM adatkészlete nagyon heterogén minőséget mutat, mivel a teljessége és minősége országról országra különbözik. Városi, beépített területeken (főleg az Egyesült Királyság, Németország, Ausztria és Svájc területén) az OSM teljessége hasonló a kereskedelmi és hivatalos szervek forrásaihoz. A vidéki területekre vonatkozó adatok koncentrációja alacsonyabb. (Neis & Zipf, 2012)

A regisztrált felhasználók száma a 2010-es évekre jelentősen megnőtt: 2011 decemberében elérte a félmilliót (ekkor ennek 38%-a szerkesztett is legalább egyszer (Neis & Zipf, 2012)), 2022 márciusának elején pedig már 8,1 millió regisztrált felhasználója volt a projektnek (OpenStreetMap stats, 2022).

A dolgozatban fejlesztett webes útvonaltervező alkalmazásban használt Openrouteservice szolgáltatás főként az OpenStreetMap adatbázisával dolgozik, ezen futtatja útvonalkereső és egyéb algoritmusait (ORS Data sources, 2022).

2.1 Adatszerkezet

Az OpenStreetMap topologikus adatszerkezetet használ. Ez az adatbázis három alapegységgel rendelkezik:

- Node (csomópont)
- Way (út/vonal)
- Relation (kapcsolat)

Node-ként tárolódik minden pont, mely szélesség és hosszúság koordinátapárral rendelkezik (WGS84). Ezek jelölhetnek egyedülálló pontszerű elemeket, “méret nélküli” pontokat (pl. POI-k), vagy részei lehetnek egy vagy több útvonalnak is (*way*). 2022 márciusában az OSM adatbázisa 7,5 milliárd *node*-ot tartalmazott.

A *way* típusú elem egy vonalas térképi elem (pl. út, kerítés, patak, villanyvezeték), melynek csomópontjait sorrendbe definiált *node*-ok adják meg. Egy út lehet nyitott és zárt: nyitott, ha az első és utolsó *node* nem egyezik meg; zárt, ha ezek megegyeznek. A zárt típusú

way lehet egy egyszerű zárt, magában végződő polyline (pl. körforgalom), vagy az `area=yes` paraméter megléte esetében terület is – ebben az esetben már poligonról beszélünk (annak ellenére, hogy poligon típusú elemfajta nem létezik az OSM adatszerkezetében), és kitöltést is kap. Amennyiben két *way* típusú elem azonos magasságban keresztezi egymást, a keresztezés helyén meg kell osztaniuk egy *node*-ot, ha más magasságban teszik ezt (pl. két út keresztezésénél, az egyiken híddal), akkor nem. A magassági elkülönítésért a `layer=` vagy a `level=` elemhez rendelt paraméterek felelnek. 2022 márciusában az OSM adatbázisa 840 millió *way*-t tartalmazott.

A kapcsolatokat leíró *relation* típusú alapegység egy vagy több *node*, *way* vagy *relation* elem rendezett csoportja, mely az elemek között valamilyen logikai kapcsolatot teremt. Ilyen pl. egy multipoligon kialakítása egy ország exklávéja esetén, vagy egy elfordulást tiltó közlekedési szabály megléte, amikor egy útról nem kanyarodhatunk a másikra. 2022 márciusában az OSM adatbázisa 9,7 millió *relation*-t tartalmazott.

A három alapegységen kívül a rendszer fontos részei a leíró *tag*-ek, melyek megszabják az alapelemek attribútumait. Ezek `kulcs=érték` formátumú párok, pl. egy útszakasz legnagyobb megengedett sebességét (50 km/h) a `maxspeed=50` *tag* attribútum szabná meg. (OpenStreetMap wiki, 2022)

Az OSM térképi adatokat tároló és kezelő rendszere PostgreSQL adatbáziskezelő rendszert és annak PostGIS kiegészítőjét használja (OpenStreetMap wiki, 2022).

3. Webes útvonaltervezők

Az útvonaltervezés két alapvető kérdése, hogy honnan és hová tervezünk, viszont fontos az is, milyen közlekedési eszközzel, milyen potenciális útvonalakon tudunk eljutni A-ból B-be. Célja a legoptimálisabb útvonal megtalálása két vagy több pont között. Az optimális útvonal kiválasztásakor az útvonal hossza és időtartama a két fő befolyásoló tényező. Rothkrantz szerint a meglévő útvonaltervezők négy típusba sorolhatóak: statikus (az úthálózat egy gráfként reprezentálódik (melyen az útszakaszok megtételének költsége ismert, pl. az utazási idő), és ezen a gráfon futtat egy legrövidebb utat kereső Dijkstra algoritmust), dinamikus (az útszakaszok átutazási ideje dinamikusan változik a úton meglévő dugók, balesetek függvényében), probabilisztikus (az útszakasz ideje nem valós idejű, hanem múltbeli, rögzített adatra hagyatkozik) és hibrid útvonaltervező (Rothkrantz, 2018).

A webes útvonaltervezők megjelenésére nagy hatással volt az internet elterjedése, a térbeli adatok egyre nagyobb elérhetősége és a nyilvános tömegközlekedés fejlődése, azok bonyolultságával és modern igényeivel együtt. A 2000-es évek elején különféle webes útvonaltervező szolgáltatások jelentek meg (pl. Google Directions vagy a Transport for London közlekedési szervezet útvonaltervezője), melyek főként a tömegközlekedésben való, minél effektívebb útvonaltervezést segítették elő. Cheung és Sengupta 2016-os elemzésében összehasonlított 20 meglévő útvonaltervező alkalmazást funkciók (multimodalitás, valós idejű navigáció és információk, testreszabhatóság), használhatóság (pontlerakás lehetőségei, POI-k, geokódolás, útvonalparaméterek, útvonalak összehasonlítása) és népszerűség szempontjából. Vizsgálatuk alapján funkciók szempontjából a Google Maps és a TripGo, használhatóság szerint a TripGo és a Citymapper, népszerűség szempontjából pedig szintén a Google Maps és a MAPS.ME alkalmazások végeztek az élen. (Cheung & Sengupta, 2016)

A tömegközlekedésben való egyén útvonaltervezésében (mely menetrendekre, pontos időbeosztásra és időtartamokra támaszkodik) fő szempont lehet, hogy az utazó minél kevesebbet várjon egy-egy átszállásnál, vagy hogy az átszállások száma a lehető legkevesebb legyen. Egy 2004-es kutatásban nagyjából a vizsgált alanyok fele nem tudta sikeresen megtervezi utazását nyomtatott anyagok (pl. menetrend a megállóban) alapján, egy menetrend-alapú közlekedési rendszerben (Alasdair, 2004). A Google 2005-ben implementálta útvonaltervező funkcióit a Google Transit alkalmazásba, mely az Oregon állambeli Portland tömegközlekedési adatait is használni tudta (az adatok szolgáltatója a TriMet volt, a helyi közlekedési szervezet) (McHugh, 2013). Ez a kollaboráció indította el a *Google* (később *General*) *Transit Feed Specification (GTFS)* kialakulását, mely később a közlekedési adatok

világszerte használt, nyílt formátuma, szabványa lett. Ez a szabvány eredetileg statikusnak számító, ritkábban frissülő adatok (pl. menetrend) tárolására és továbbítására létrehozott szabvány volt, de a későbbi GTFS-realtime kiegészítés lehetővé tette, hogy késéseket, forgalmi változásokat vagy az adott közlekedési eszköz (busz, vonat, villamos stb.) jelenlegi helyzetét is nyilvánossá tegyék a szervezetek (ez alapján az alkalmazás pl. érkezési időt jósoljon) (GTFS Realtime, 2022). 2017-ben közel 800 tömegközlekedési vállalat használta ezt a formátumot adatai továbbadásához (Zipper, 2017). Az OpenMobilityData nyilvános GTFS-adatokat ("GTFS-feed") gyűjtő weboldal szerint 2022 májusában több, mint 1300 szervezet osztotta meg adatait a nyilvánossággal, kb. 670 helyről világszerte (OpenMobilityData, 2022). A Budapesti Közlekedési Központ szintén elérhetővé teszi a teljes budapesti közösségi közlekedési menetrendi adatbázist GTFS-formátumban, valamint valós idejű adatokat is szolgáltat GTFS-realtime formátumban (Budapesti Közlekedési Központ, 2022). A nagy mennyiségű, hozzáférhető GTFS-adatok lehetővé teszik, hogy bárki saját alkalmazásába közlekedési adatokat integráljon (Antrim & Barbeau, 2013).

4. Az Openrouteservice szolgáltatás és lehetőségei

Az Openrouteservice (https://openrouteservice.org/) egy összetett, ingyenes, nyílt forráskódú webes (lokálisan is futtatható) szolgáltatás, mely a fő profilján, az útvonaltervező szolgáltatáson kívül más alszolgáltatásokat is kínál (pl. geokódoló, izokrónkészítő). Az ORS projekt 2008 áprilisában indult, Pascal Neis és Alexander Zipf alapötlete nyomán (OpenStreetMap wiki: Openrouteservice, 2022). Az ORS-t a Heidelberg Institute for Geoinformation Technology (HeiGIT) csoport fejleszti és tartja karban, mely a Heidelbergi Ruprecht Karl Egyetem alá tartozó GIScience Research Group (Institute of Geography) része (Meet the Team - ORS, 2022). Az ORS által kínált adatokra a CC BY 4.0 típusú licensz érvényes. A Java-ban írt projekt teljes forráskódja elérhető GitHub-on. Kínál egy átfogó webes API-t is, mely a beérkező lekérések alapján szolgáltat adatot a felhasználónak (pl. egy külön alkalmazás esetében, mely az ORS szolgáltatásból érkező útvonaladatokat használja). Ez az ingyenes API egy gyors regisztráció után kapott ún. API-kulccsal (API-key) használható, viszont lekérési korlátozásokkal rendelkezik (pl. 2000 útvonalat lehet csak lekérni naponta; 40 darabot percenként). Ezeket a korlátozásokat vélhetően a weboldal túlterheltségének megakadályozásaképpen vezették be. Elérhető egy szintén ingyenes “Collaborative” használati csomag is, mely akadémiai, humanitárius, kormányzati és non-profit célú felhasználásra van kihegyezve, és nagyobb lekérési limitekkel rendelkezik. A webes, API-alapú elérhetőségen kívül az Openrouteservice szoftvercsomagja ingyenesen letölthető és futtatható bármilyen saját szerveren is – ebben az esetben a lekérések száma nincs korlátozva, valamint az üzemeltető a konfigurációban is szabad kezet kap.

Az Openrouteservice az OpenStreetMap adatbázisát használja térképi adatbázisként, és ezen futtatja algoritmusait. A magassági adatok adatforrásai a Shuttle Radar Topography Mission (SRTM) és a Global Multi-resolution Terrain Elevation Data (GMTED2010) felmérések (ORS Data sources, 2022).

Az ORS hét fő végponttal (endpoint) rendelkezik, ezeken keresztül érhetőek el az egyes alszolgáltatások: útvonaltervező, izokrónkészítő, idő- és távolságmátrix, geokódoló, POI, magassági adat és útvonal-optimalizáló szolgáltatás.

A dolgozatban fejlesztett webes útvonaltervező alkalmazás az Openrouteservice-től kéri le a webes API-on keresztül az adatokat, és jeleníti meg azokat a felhasználói felületen és a térképen. Az ORS bizonyos végpontokon *GET*, de többségében *POST* típusú HTTP kéréseket vár, melyeket, ha megfelelő szintaxisúak, feldolgoz. A *GET* kérést egyszerűen, webcím-szerűen

lehet indítani (itt megadva az API-key-t is), míg a *POST* típusú lekérésnél az autorizációs fejlécben (Header) kell meg adni az ORS-től kapott API-kulcsot (ezzel azonosítjuk a felhasználónkat), valamint a lekérés üzenettestében (Body) JSON formátumban kell megadni az összes paramétert, mely alapján az ORS végrehajtja a lekérést. A legtöbb szolgáltatás lekérési paramétereiben megadható, milyen kimeneti formátumot szeretnénk majd kapni (pl. JSON, GeoJSON, GPX). Az ORS-től kapott válasz mindig tartalmaz státuszkódokat (három- vagy négyjegyű számokat), melyek egyértelműen azonosítják, ha a feldolgozás során hiba lépett fel, de azt is, ha sikeres volt. A szolgáltatások elérési címében kötelező az útvonalprofil megadása, mely a következők egyike kell, hogy legyen: `driving-car`, `driving-hgv`, `cycling-regular`, `cycling-road`, `cycling-mountain`, `cycling-electric`, `foot-walking`, `foot-hiking`, `wheelchair`. Az API-ról való lekérések esetében minden végpont rendelkezik előre meghatározott limitációkkal, hogy biztosítani tudják az ORS webes szolgáltatásának stabilitását. Ilyen pl. az útvonaltervezés teljes útvonalhossza (max. 6000 km), vagy a polyline node-jainak száma vonalra vonatkoztatott magassági lekérés esetében (max. 2000) (ORS API Restrictions, 2022). Ezeket a limitációkat megfelelően le kell kezelnie az alkalmazásnak, hogy a felhasználó ne tudjon ilyen paraméterekkel keresést indítani, és az esetleges hiba esetén mindenképpen értesüljön a problémáról, miközben az alkalmazás továbbra is használható marad.

Az API-dokumentáció az ORS oldalán olvasható (https://openrouteservice.org/dev/#/api-docs), a 4.1-4.7 fejezetekben leírt dokumentáció az ORS Core 6.7.0-s változatára vonatkozik. (Openrouteservice.org, 2022)

4.1 Útvonaltervező szolgáltatás

Az útvonaltervező végpont az ORS projekt egyik fő szolgáltatása. A két pont közötti szimpla útvonaltervezést az egyszerűbb, *GET* típusú lekéréssel is el lehet indítani (ez GeoJSON formátumban adja vissza az eredményt). Ha kettőnél több pontot szeretnénk érinteni az útvonal során, vagy egyéb paramétereket is szeretnénk használni, muszáj a *POST* típusú szolgáltatások valamelyikét használni. Ebből három létezik: JSON, GeoJSON és GPX formátumot képesek visszaadni. A dolgozatban tárgyalt webes alkalmazás útvonaltervező funkciója GeoJSON formátumban kéri le az útvonalat az ORS-től. Az ORS-től érkező válaszban a teljes útvonal egy darab *feature* objektumként érkezik meg a ‘*features’* objektumon belül. Ha 3 vagy több útvonalpontot érintünk, az útvonalpontok közötti szakaszok instrukciói a *feature.properties.segments* tömbben vannak felsorolva (pl. 3 útvonalpont esetében 2 szegmenst kapunk).

A lekérés üzenettestében (Body) a következő paraméterek adhatók meg:

- *coordinates*: hosszúság/szélesség sorrendben megadott koordinátapárokból álló tömb, melyek mindegyike érintendő útvonalpontokat fejez ki; értelemszerűen két vagy több koordinátapár szükséges
- *alternative_routes*: egy JSON objektumot vár, melyben a következő elemek létezhetnek:
    - *target_count*: megszabja, hány útvonalat generáljon le; egész szám
    - *share_factor*: a legnagyobb megengedett aránya egy alternatív útvonal szegmensei által és az optimális útvonal által megosztott szegmenseknek; 0-1 közötti lebegőpontos szám
    - *weight_factor*: az alternatív útvonalak eltérésének legnagyobb megengedett aránya; lebegőpontos szám (pl. 1.3 azt jelentené, hogy az alternatív útvonalak legfeljebb 1.3-szor lehetnek hosszabbak az optimális úthoz képest)
- *attributes*: extra, egy útvonalszegmenst (útvonalponttól útvonalpontig tartó szakasz) leíró attribútumok lekérése (pl.: *avgspeed*, *detourfactor*, *percentage*); tömb
- *continue_straight*: ha aktív, egy közbeeső útvonalpont elérése esetén egyenesen halad tovább, és nem fordul meg helyben akkor sem, ha az gyorsabb lenne; boolean (true/false)
- *elevation*: ha aktív, a teljes útvonal töréspontjainak koordinátái mellé magassági értékeket is rendel; boolean (true/false)
- *extra_info*: extra attribútumok lekérése (pl.: meredekség, burkolat, útvonaltípus) (*steepness, suitability, surface, waycategory, waytype, tollways, traildifficulty, osmid, roadaccessrestrictions, countryinfo, green, noise*); tömb
- *geometry_simplify*: az útvonal simítása; boolean (true/false)
- *instructions*: lépésenkénti instrukciók lekérése; boolean (true/false)
- *instructions_format*: a kapott instrukciók formátuma; szöveg (text/html)
- *language*: a szöveges instrukciók nyelve; szöveg (pl.: en, hu)
- *maneuvers*: a lépésekhez rendelt manőver hozzácsatolása hellyel, előtti és utáni azimuttal; boolean (true/false)
- *options*: egy JSON objektumot vár, melyben a következő elemek létezhetnek:
    - *avoid_borders*: országhatárok elkerülése; szöveg (all/controlled/none)

    - *avoid_countries*: az elkerülendő országok listája tömbbe rendezve, az országokhoz rendel kódok alapján
    - *avoid_features*: autópályák, útdíjak és kompok elkerülése; tömb (*highways*, *tollways*, *ferries*)
    - *avoid_polygons*: GeoJSON-ként formázott poligon vagy multipoligon objektumot vár – a poligon által lefedett területet az útvonal mindenképp elkerüli
- *preference*: útvonaltervezési módszer; szöveg (*fastest*/*shortest*/*recommended*)
- *radiuses*: minden egyes útvonalpont úthoz való csatolásának toleranciája, méterben kifejezve (prioritási sorrendben halad a tömbben; a -1 a végtelen hosszú toleranciát jelenti); számokból álló tömb (pl.: [200, -1])
- *roundabout_exits*: a lépések objektumához hozzácsatolja a körforgalmak kimeneti irányát is; boolean (true/false)
- *skip_segments*: két szegmens közötti útvonal átugrása az útvonaltervezésnél; egész számokból álló tömb (pl.: [2, 5])
- *suppress_warnings*: figyelmeztetések mellőzése a válaszban; boolean (true/false)
- *units*: a válaszban használt hossz-mértékegységek; szöveg (m/km/mi)
- *geometry*: útvonalgeometria visszaadása a válaszban; boolean (true/false)
- *maximum_speed*: a megszabott maximális sebesség; szám (pl.: 90)

4.2 Izokrónkészítő szolgáltatás

Az izokrónkészítő szolgáltatás képes az egy vagy több pontra végzett idő- és távolságalapú analízisre. Figyelembe veszi a megadott útvonalprofilt is (az idő alapú analízisnél fontos figyelembe venni, hogy autóval, vagy gyalog tervezünk). Képes egy azon pontra más és más távolságú izokrónok/izodisztansok legyártására is. Mindegyik visszaadott elem poligon típusú, és a válaszban a ’*features’* objektum tartalmazza ezeket, a geometria töréspontjaival és egyéb adatokkal együtt.

Az ORS poligon-generálási algoritmusa Duckham, Kulik, Worboys és Galton: “Efficient generation of simple polygons for characterizing the shape of a set of points in the plane” (Duckham, Kulik, Worboys, & Galton, 2008) publikációján alapszik (Openrouteservice.org, 2022).

A lekérés üzenettestében (Body) a következő paraméterek adhatók meg:

- *locations*: hosszúság/szélesség sorrendben megadott koordinátapárokból álló tömb, melyek mindegyike különálló pontot fejez ki (az izokrón(ok) középpontjait, kiindulópontjait); tömb
- *range*: a kívánt poligonok távolságai (azt, hogy időnek (mp), vagy távolságnak (m) értelmezi, a későbbi *range_type* paraméter szabja majd meg); tömb (pl.: [400, 200])
- *attributes*: extra, egy izokrón poligont leíró attribútumok lekérése (*area, reachfactor, total_pop*); tömb
- *intersections*: egymást metsző poligonok visszaadása; boolean (true/false)
- *interval*: amennyiben csak egy darab *range* érték alapján generált izokrónt kérünk le az lesz a felső határ, és ez a szám adja meg a 0-tól való osztásközt, egészen a megadott *range* értékig – így range/interval darabszámú poligont kapunk egy adott pontra; szám (pl. 50)
- *location_type*: a megadott pont kiinduló- vagy célpontként való meghatározása; szöveg (*start*/*destination*)
- *options*: egy JSON objektumot vár, melyben az útvonaltervező szolgáltatásnál is említett elemek létezhetnek (lásd 4.1 fejezet)
- *range_type*: a poligonok típusa (izokrón/izodisztans); szöveg (*time*/*distance*)
- *smoothing*: poligon simításának szorzója; szám (0-100 között)
- *area_units*: a válaszban használt terület-mértékegységek; szöveg (m/km/mi)
- *units*: a válaszban használt hossz-mértékegységek; szöveg (m/km/mi)

4.3 Idő- és távolságmátrix szolgáltatás

Az ORS képes több kiindulópont és több célpont közötti, 1:n, n:1 vagy n:m típusú idő- és távolságmátrix legenerálására. Alaphelyzetben minden megadott pontból minden megadott pontba számítja az eredményeket. A lekérés *POST* típusú.

- *locations*: a pontokat jelölő koordinátapárok tömbje
- *destinations*: az összes helyett csak a megadott pontok intervallumát értelmezi célpontként; tömb (pl.: [0,3] az első, második, harmadik és negyedik megadott pontot veszi célpontként)
- *metrics*: a mátrix típusa (idő, és/vagy távolság alapú); tömb (distance/duration)

- *resolve_locations*: a pontok fordított geokódolása, melynél a ponthoz legközelebbi utcanév visszaérkezik a válaszban az azonosítást segítendően; boolean (true/false)
- *units*: a távolság mértékegysége; szöveg (pl.: m)

4.4 Geokódoló szolgáltatás

Az Openrouteservice geokódoló szolgáltatása a Pelias (https://pelias.io/) nyílt forráskódú geokódoló motort használja. Lehetőség adódik geokódolásra (amikor egy szöveges kereséshez rendelünk hozzá koordinátákat), valamint fordított geokódolásra is (egy koordinátához rendeljük hozzá a legvalószínűbb, környékén lévő térképi elemeket).

Az ORS geokódoló szolgáltatásai *GET* típusú kéréseket várnak, melynél az API-kulcson kívül a következő paramétereket lehet megadni:

- *text*: a geokódolandó szöveg
- *focus.point.lon* és *focus.point.lat*: a találatokat az ezen ponttól számított lináris távolság alapján rendezi sorrendbe; hosszúság és szélesség számmal
- *boundary.rect.min_lon*, *boundary.rect.min_lat*, *boundary.rect.min_lon*, *boundary.rect.min_lat*: egy téglalap alakú területre szűkíthetőek a találatok
- *boundary.circle.lon*, *boundary.circle.lat* és *boundary.circle.radius*: egy kör alakú területre szűkíthetőek a találatok (a középpont és a sugár megadásával)
- *boundary.country*: a találatok egy konkrét országra való szűkítése (2- vagy 3-karakteres ISO 3166-1 alpha-2 és alpha-3 országkódokkal); szöveg (pl.: DE)
- *sources*: az adatbázisok, melyben a geokódoló keres (ha nincs megadva, mindegyikben keres); tömb (*openstreetmap*, *openaddresses*, *geonames*, *whosonfirst*)
- *layer*: találatok szűkítése térképi elemek típusaira; tömb (pl.: region, country, address)
- *size*: találatok száma; szám (pl.: 10)

A fordított geokódoló amellett, hogy értelemszerűen szöveg helyett egy koordinátapárt vár és a találatok attribútumait adja vissza, hasonló lekérési paraméterekkel rendelkezik (kivéve, hogy jelen esetben nem lehet szűrő téglalapot megszabni, mivel egy pontot adunk meg).

4.5 POI-szolgáltatás

A POI-szolgáltatás visszaad egy befoglaló téglalap, egy poligon, egy vonal vagy egy pont bufferje alá eső POI-listát. A visszakapott POI-elemek típusa, OSM-beli azonosítója, kategóriája és tag-listája is helyet kap a válaszban. A lekérés *POST* típusú.

A lekérés üzenettestében (Body) a következő paraméterek adhatók meg:

- *request*: a válasz típusa - *pois* esetében a tényleges térképi elemek, a POI-k jönnek vissza, *stats* esetében a POI-k száma kategóriákra bontva, *list* esetében pedig egy kategórialista; szöveg (pois/stats/list)
- *geometry*: a keresési terület geometriájára vonatkozó adatok; JSON objektum
    - *bbox*: két koordinátapár, melyek a befoglaló téglalap északkeleti, majd délnyugati sarokpontját adják meg
    - *geojson*: pont, vonal vagy poligon geometriáját leíró GeoJSON objektum
    - *buffer*: a geometria köré számított buffer szélessége; egész szám (pl.: 500)
- *filters*: OSM-tagekre való tetszőleges szűrés (*osm_tags*); JSON objektum
- *limit*: a visszakapott találatok száma; egész szám (pl.: 1000)
- *sortby*: a találatok kategória vagy távolság szerinti rendezése; szöveg (category/distance)

4.6 Magasságiadat-szolgáltatás

Az ORS kínál magasságiadat-szolgáltatást is, mely képes a kapott geometriához (legyen az pont, vagy vonal) magassági értékeket rendelni. A lekérésben megadott geometriai elem minden töréspontjának koordinátapárjához rendel egy harmadik értéket, mely a magasság értéke, méterben. A geometria bemeneti és kimeneti formátuma négyfajta lehet: GeoJSON, polyline, vagy a Google által használt, veszteséges polyline-kódoló formátuma 5- és 6-tizedes koordináta-pontossággal. A magassági adatok forrásai az SRTM és a GMTED2010. A ponthoz magasságot rendelő szolgáltatás *POST* és *GET*, a vonalhoz magasságot rendelő szolgáltatás pedig *POST* típusú.

A lekérés üzenettestében (Body) a következő paraméterek adhatók meg:

- *format_in*: a geometria bemeneti formátuma; szöveg (geojson, polyline, encodedpolyline5, encodedpolyline6)
- *format_out*: a geometria bemeneti formátuma; szöveg (geojson, polyline, encodedpolyline5, encodedpolyline6)
- *dataset*: a magassági adatok forrása (a dolgozat írásakor csak az SRTM volt elérhető); szöveg (srtm)
- *geometry*: a geometria, mely lehet pont vagy vonal

4.7 Útvonal-optimalizáló szolgáltatás

Az ORS kínál egy útvonal-optimalizáló szolgáltatást is, mely képes több közlekedési eszköz (pl. teherautó) egyidejű útvonalának legoptimálisabb megtervezésére, miközben ezek több célpontot (*job*-ot) érintenek, időablakokat, kapacitást és képességeket is figyelembe véve. A szolgáltatás a C++-ban írt, nyílt forráskódú VROOM (Vehicle Routing Open-source Optimization Machine) projekt algoritmusait használja, melynek a legoptimálisabb útvonalak megtalálása mellett az is egyik fő célja, hogy a számítás a lehető leggyorsabban fusson le. A VROOM projekt számos benchmark tesztet is kínál, mely különféle útvonaltervezési problémákat old meg (VRPTW, MDVRP, PDPTW, CVRP, TSP) bizonyos adatkészleteken (pl. az MDVRP (Multiple Depot Vehicle Routing Problem) probléma megoldását Cordeau, Gendreau és Laporte 1997-es adatkészletén futtatja (Cordeau, Gendreau, & Laporte, 1997)) (VROOM-Project Benchmarks, 2022).

A szolgáltatás lekérése *POST* típusú. A lekérés üzenettestében (Body) a következő paraméterek mindenképp szükségesek:

- *jobs*: az elérendő célpontokat és azok paramétereit (*id*: azonosító; *service*: az ott töltött idő; *location*: a célpont helye, koordinátapár; *skills*: szükséges képességek, melyekre az érkező járműnek képesnek kell lennie; *time_window*: időablak; stb.) gyűjtő tömb
- *vehicles*: a járművek és azok paramétereit (*start*: kiindulópont, koordinátapár; *end*: végső érkezési pont, koordinátapár; *capacity*: a jármű kapacitása; *skills*: a jármű képességei; *time_window*: az időablak, melyben a jármű aktív; stb.) gyűjtő tömb

5. A webes tartalmak alkotóelemei: HTML, CSS és JavaScript

A HTML, a CSS és a JavaScript (JS) a három alapvető pillére a világhálón elérhető webes tartalmaknak: HTML a weboldal tartalmáért, CSS a megjelenésért, a JS pedig az interaktivitásért felel. (Flanagan, 2011)

A HTML (Hypertext Markup Language) egy, az internetes oldalak leírására szolgáló nyelv. Az ilyen anyagok többnyire .html vagy .htm kiterjesztésű fájlokban rögzülnek. A fájlban vezérlő elemekkel találkozunk, melyek megjelölése a `<` és `>` jelek közé van zárva (pl. a félkövér tartalmat jelölő `<b>` elem). Az ilyen vezérlő elemeknek többnyire van egy nyitó és egy záró része is: a záró részt egy `/` jel indikálja, pl. `</b>`). Ilyen elemek között helyezkedik el a tartalom, melyre az adott elem vonatkozik (pl. `<b>Cím</b>`).

Egy HTML dokumentum kezdetét és végét a `<html>` elem definiálja. Ez a tartalom két fő részre tagolható – egy fejlécre (`<head>`), valamint egy törzsre (`<body>`):

- `<head>`: a dokumentum fejléce, melyben különféle technikai információkat és metaadatokat definiálhatunk (pl. az oldal címe, mely megjelenik a böngészőben (`<title>`), a fájl karakterkódolása (pl. `<meta charset=”utf-8”/>`), stb.). Ebben a fejlécben szabhatjuk meg a teljes stílusdefiníciót a `<style>` elem segítségével, valamint a `<script>` elemmel itt ágyazhatunk be külső, egy másik fájlban tárolt programkódot is, hogy a HTML dokumentumból hivatkozni tudjunk rá, és igénybe tudjuk venni a külső függvénykönyvtár lehetőségeit.
- `<body>`: a dokumentum törzse, mely a böngészőben megjelenő teljes tartalmat foglalja magába. A törzsben lévő `<script>` elemben közvetlenül a dokumentumban írhatjuk meg a JavaScript nyelvű programkódot, mely a weboldal böngészése közben futni fog.

A törzsben alkotjuk meg a tényleges tartalmat, melyet a felhasználó látni fog, esetleg, amivel interakcióba lép. Ez magába foglalja a teljes szöveges tartalmat, gombokat, képeket, valamint a szöveg tördelését, tagolását is (melyet a vezérlő elemek definiálnak). A tagolást segítő vezérlő elemek sokfélék, pl.: bekezdést `<p>...</p>` közé, félkövér szöveget `<b>...</b>` közé, dőltet `<i>...</i>` közé zárunk, sortörést a `<br />` elemmel szúrhatunk be, sorszámozott listát az `<ol>...</ol>` segítségével hozunk létre (melyben a lista elemei `<li>…</li>` típusúak), táblázatokat hozhatunk létre a `<table>...</table>` elemmel, linkeket szúrhatunk be az `<a href=”linkURL”>...</a>`, képeket szúrhatunk be az `<img`

`src=”minta.jpg”/>` elemmel, a gombokat a `<button>` elem, a bemeneti mezőket (melybe a felhasználó beírhat valamit) az `<input>` elem valamely típusa hozza létre. A dokumentumban logikai egységeket képezhetünk a `<div>...</div>` elem segítségével, melyek egy komplexebb, erősen stilizált, interaktív weboldal esetében kulcsfontosságúak. Az összes elemhez hozzárendelhető egy azonosító (`id=”xy”`) – ezekre hivatkozva dinamikusan változtathatjuk (JS programkóddal), vagy stilizálhatjuk a tartalmat a CSS stílusleíró nyelv segítségével. (Gede, HTML összefoglaló, 2022)

A Cascading Style Sheets (CSS) egy stílusleíró nyelv, mely az érintett HTML weboldal tartalmának stílusdefiníciójáért felel. A CSS stíluslapok segítségével egyedi, de egységes megjelenést tudunk biztosítani HTML oldalunknak. A JavaScript fájlokhoz hasonlóan ezt sem kell feltétlenül külön fájlban tárolni és arra hivatkozni (csak abban az esetben, ha ugyanazt a fájl több oldal is használja) – beszúrhatjuk közvetlenül a HTML kódba is, az annak fejlécében lévő `<style>` elem segítségével. A CSS-ben megírt stílusszabályok a HTML elemek megjelenését definiálják, pl. helyzetet, négy oldali margót/paddinget, méretet, színt, háttérszínt, betűtípust, betűméretet, keretet, elrendezésbeli magasságot (z-index) stb. A szabályban egy tulajdonság időbeli változtatására (animálására) is van lehetőség, ezek használatával a weboldal kifinomultabbá, „élettel telibbé” tehető. Az animációk finomhangolásához használhatunk a videószerkesztés világából ismert keyframe-eket, melyekkel megszabható, hogy az animáció alatt melyik időpillanatban mennyi legyen egy tulajdonság értéke. Egy általános stílusszabály a következőképp épül fel:

Konkrét azonosítóra hivatkozva:

```
#elemazonosító {
    tulajdonság: érték;
    tulajdonság: érték
    ...
}
```

Osztály definíciója:

```
.osztálynév {
    tulajdonság: érték;
    tulajdonság: érték
    ...
}
```

Ha a szabály konkrét azonosítóra hivatkozik, értelemszerűen csak arra az elemre lesz érvényes, mely az adott azonosítóval rendelkezik (id). Ha osztályt definiálunk, a HTML elem `class=”osztálynév”` paraméterével használhatjuk az adott osztályt.

A stílusszabályok létrehozhatók HTML elem-típusra is (pl. `<button>` elemekre), viszont így az összes ilyen típusú elemre érvényes lesz. Ez utóbbi hasznos viszont, ha például az egész `<html>` vagy `<body>` elem szélességét/magasságát szeretnénk megszabni (a képernyő teljes kihasználása érdekében), vagy egy konkrét betűtípust/betűméretet beállítani, hogy

a weboldalon egységes legyen a betűtípus. Így nem kell minden egyes elemnek külön beállítani a betűtípust, elég a dokumentumtörzs (`<body>`) számára definiálni. (Gede, Script nyelvek alkalmazása a web-kartográfiában, 2022)

A JavaScript egy script nyelv (futás közben értelmezett = interpretált), mely gyengén típusos, szintaxisa C nyelv-szerű, hasonlít a Java-ra is. Nagyon elterjedt: a weboldalak mintegy 97,9%-a használja a JS-et kliens oldalon (W3Techs, 2022). Szintaxisát tekintve:

- gyengén típusos: a változók típusát nem lehet előre definiálni, hozzárendeléssel változtathatjuk menet közben is,
- a változók típusai: boolean (logikai), number (bármilyen típusú szám), string (szöveg), array (tömb), object (objektum) valamint meghívható function (függvény),
- érzékeny a kis- és nagybetűk közti különbségre,
- az utasításokat pontosvessző (`;`) vagy újsor jel határolja,
- utásítás-blokkokat `{` és `}` jelek között hozunk létre,
- elérhetőek a megszokott vezérlési szerkezetek, pl. elöl (`while`), hátul tesztelős ciklus (`do`), `for` ciklus, `if-else` elágazások.

A JS számára a HTML dokumentumot és annak hierarchikus rendbe szervezett elemeit a DOM (Document Object Model) reprezentálja. Script kód segítségével a DOM elemeit futás közben dinamikusan módosíthatjuk, ezáltal a HTML dokumentum tartalma is megváltozik (pl. átírunk egy adott szöveges elemet, megjelenítünk egy képet stb.). A HTML elemeire az ott megadott `id=”xy”` azonosítóval tudunk hivatkozni. (Gede, Script nyelvek alkalmazása a web-kartográfiában, 2022)

6. A webes alkalmazás fejlesztése

A webes alkalmazás fejlesztéséhez – mely a felhasználói bemenetet feldolgozza, majd az Openrouteservice szolgáltatástól a megfelelő módon lekéri, feldolgozza és megjeleníti az adatokat egy térképen –, JavaScript nyelvet használtam. A programkódot a HTML fájlba ágyaztam, a weboldal a betöltés után máris használható. A webes alkalmazás felhasználói felületének kinézetét és interaktivitását (részben) a CSS stílusleíró nyelvben definiáltam, mely szintén a fő HTML fájlban kapott helyet.

Az általam fejlesztett alkalmazás térképi alapként az OpenStreetMap (© *OpenStreetMap contributors*), az OpenTopoMap (© *OpenTopoMap CC-BY-SA*) és a Stamen Terrain (© *Stamen Design, CC BY 3.0*) térképcsempéit használja a megjelenítéshez, közvetett módon pedig az OSM adatbázisát használja az útvonaltervezéshez – ezt már az Openrouteservice szolgáltatás (© *openrouteservice.org by HeiGIT | Map data © OpenStreetMap contributors*) végzi a saját szerverein. Az Openrouteservice által visszaadott magassági adatokra való hivatkozás dinamikusan jelenik meg az alkalmazás oldalpaneljében és a Leaflet attribúcióra fenntartott paneljén (ezek forrása SRTM vagy GMTED2010 lehet).

Az alkalmazás számos külső forrásból származó függvénykönyvtárt használ, melyek nélkül nem, vagy csak részben valósulhatott volna meg az alkalmazás. A teljes hivatkozáslista a fő HTML dokumentum első soraiban is olvasható. A következő, nyílt forráskódú csomagokat használtam:

Leaflet.js

Vladimir Agafonkin - https://leafletjs.com/ (BSD 2-Clause "Simplified" License: https://github.com/Leaflet/Leaflet/blob/main/LICENSE)

JavaScript nyelven írt, ingyenesen és szabadon használható függvénykönyvtár, mely segítségével térbeli adatok jeleníthetőek meg egy weboldalba ágyazható, mobilos környezetben is használható interaktív térképen. (Agafonkin, 2022)

Leaflet.contextmenu

Adam Ratcliffe - https://github.com/aratcliffe/Leaflet.contextmenu (MIT License: http://www.opensource.org/licenses/mit-license.php)

A Leaflet térképén belüli jobbegérgomb-kattintást egy kis menü megjelenítésével lekezelő függvénycsomag.

Leaflet.PolylineMeasure

"ppete2" (GitHub), Jürgen Treml - https://github.com/ppete2/Leaflet.PolylineMeasure (*BSD 2-Clause "Simplified" License: https://github.com/ppete2/Leaflet.PolylineMeasure/blob/master/LICENSE.txt*)

A contextmenu csomaghoz hasonlóan ez is egy Leaflet-ben használható kiegészítő. Feladata az egyszerű vonalon, vagy akár komplexebb polyline-on való távolságmérés. A mérést ortodróma mentén végzi (tehát gömbi távolságot számol, mely a gömbfelületen lévő két pont közötti távolság, legrövidebb út). A kettőnél több csomópontos polyline elemek teljes hossza mellett információt ad a pontokba érkező és a pontokból kimenő azimutról, valamint a csomópontok közötti szakaszhosszról is.

Chart.js

Downie, N., Perkins, D., Timberg, E., Sylvestre, J., Panzarino, Z., Brunel, S., & Linsley, T. - https://www.chartjs.org (MIT License: http://www.opensource.org/licenses/mit-license.php)

Adatvizualizációs függvénykönyvtár, mely különféle formátumú adatokat tud megjeleníteni 8 féle, modern, letisztult, és testre szabható grafikonon és diagramon, reszponzív módon (Downie, és mtsai., 2022). Az alkalmazásban a feldolgozott magassági adatok megjelenítője.

Turf.js

Turf.js: Copyright (c) 2019 Morgan Herlocker - https://turfjs.org/ (MIT License: https://github.com/Turfjs/turf/blob/master/LICENSE)

Térinformatikai feladatok elvégzésére képes, kis modulokból álló függvénycsomag. A turf.js az alkalmazásban a magassági adatok lekérése előtti polyline-átszámítást végzi (hogy az ORS-nek elküldött polyline csomópontjainak száma ne lépje át a 2000 db-os határt).

Az alábbi alfejezetekben részletezem a webes alkalmazás szerkezetét és működését. A programkód teljes terjedelme miatt csak egyes részeire térek ki, egyes kulcsfunkciókat mutatok be. A teljes weboldal CSS-kódja megközelítőleg ~880, a HTML elemek definiálása ~330, a JavaScript programkód pedig ~2200 sorból áll (közel 50, sokszor egymásba fűződő függvénnyel).

6.1 Leaflet térképobjektum

Az alkalmazás egyik, ha nem a legfontosabb kulcseleme a térképobjektum, mely a Leaflet függvénykönyvtáron alapszik. A HTML dokumentumban létrehoztam a `<div id="mapdiv"></div>` elemet, melynek egyedi azonosítójára hivatkozva a JavaScript

programkódból inicializálni tudom a Leaflet térképet (mely a kód többi részéből elérhető majd a `map` változó segítségével), a következő beállításokkal:

```
var map = L.map('mapdiv', { zoomControl: false,
                            contextmenu: true,
                            contextmenuWidth: 165,
                            contextmenuItems: [{...}]
                          }).setView([47.19531,19.54467], 8);
L.control.zoom({position: 'bottomright'}).addTo(map);
L.control.scale({position: 'bottomleft'}).addTo(map);
L.control.polylineMeasure({...}).addTo(map);
```

A `zoomControl` paraméterrel kikapcsoltam az alapértelmezetten bal felső sarokba kerülő nagyító/kicsinyítő gombokat. A `contextmenu` paraméter `true`-ra állításával aktiváltam a Leaflet.contextmenu függvénycsomag jobbegérgomb-kattintásra megnyíló menüjét, majd ugyanitt meg is adtam az ott megjelenő funkciók listáját (`contextmenuItems` tömb; ezek mindegyike egy-egy saját függvényt hív meg, pl. az "Útvonal innen" funkció a `naviFrom()` függvényt). A Leaflet térkép létrehozása után a kezdőnézetet (térképközéppont-koordinátapár és zoom szint) Magyarország középpontja környékére állítottam. Az alap zoom-kezelőt létrehoztam a jobb alsó, a mértéklécet pedig a bal alsó sarokban. A térképobjektumhoz hozzáadtam még a `Leaflet.PolylineMeasure` controlt is, melynek segítségével távolságokat mérhetünk a térképen (ennek kezelője a jobb alsó sarokban kapott helyet).

Háttértérképként az OpenStreetMap XYZ térképcsempéit használom, mivel az ORS is az OSM adatbázisán fut, és ennek függvényében az eredményeket is érdemes ezen a térképen megjeleníteni. Az OSM tileLayert, a hivatkozást, és a legkisebb és legnagyobb zoomszintet a következő kódrészlet szabja meg, majd hozzáadja a már létrehozott Leaflet térképobjektumhoz:

```
L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
                attribution: '&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors',
                minZoom: 2,
                maxZoom: 19
            }).addTo(map);
```

A térképobjektumhoz kapcsolódnak még az egérkurzor térkép feletti mozgatását lekezelő függvények is (`map.on('mousemove', ...)` és `map.on('mouseout', ...)`), melyeknek köszönhetően megjelenik az ahhoz a helyhez kapcsolódó koordinátapár szélesség/hosszúság sorrendben, a jobb alsó sarokban a térképhivatkozások mellett.

6.2 Útvonaltervezés

Az útvonaltervezés a legösszetettebb folyamat az alkalmazásban. A futás során sor kerül többfajta beolvasásra és ellenőrzésre, pontok geokódolására, útvonalterv lekérésére,

útvonalszakaszok és pontok megjelenítésére és interaktívvá tételére, egyéb információk és eredmények felhasználói felületbe való dinamikus betöltésére, valamint a polyline egyszerűsítésére és magassági adatok lekérésére, azok megjelenítésére is.

Az útvonaltervezést többfajta módon lehet indítani, melyek mindegyikének feltétele a kiindulási, cél, és az esetleges közbeeső pont megléte (vagy szöveges formában, vagy már geokódolva, tömbben). Ha ezen pontok valamelyike hiányzik, az útvonaltervezés és a hozzá kapcsolódó folyamatok nem futnak le annak érdekében, hogy az ORS-hez ne jusson el a hibás, hiányos lekérés. Ebben az esetben a hiányzó adatok mezői pirosra váltanak, jelezve a felhasználónak, hogy valami hiányzik, valamint a `showMsgBox(szoveg, tipus)` függvénynek köszönhetően egy hibaüzenet is megjelenik (ez teljeskörűen érvényes az alkalmazásra), és ha a `hiba` boolean változó értéke `true`, a futás megáll. A mezők 1,5 másodpercig pirosra váltásának animációjáért ehhez hasonló kódrészletek felelnek:

```
if (waypoint_0_text == ""){
    hiba = true;
    w0.classList.add("ABfield_error");
    setTimeout(function(){w0.classList.remove("ABfield_error");}, 1500);
};
```

Az útvonalpont-mezők támogatják a szöveges bevitelt (pl. “`Pázmány Péter sétány 1/A, Budapest`”), a koordinátamegadást (szélesség, hosszúság sorrendben, vesszővel elválasztva, pl. “`47.48841, 19.07046`”), de a mezőbe kattintás után a térképre kattintva, vagy a térképre jobb gombbal kattintva, az „Útvonal innen/ide” funkcióval való pontleszúrást is.

Az útvonaltervezés folyamatát az útvonalpont-mezők szerkesztése közbeni Enter gomb lenyomásával, az indító gomb megnyomásával, az útvonalpontok irányának megfordításával (amit a `reverseWaypoints()` függvény kezel), vagy bármilyen útvonaltervezési paraméter (elkerülendő útelemek, határok, útvonalprofil) megváltoztatásával lehet kezdeményezni. Az útvonaltervezés akkor is indul, ha jobbegérgombbal adjuk meg a kiinduló- és célpontot („Útvonal innen/ide”). Minden esetben a `directionsCheck()` függvény kezdi a folyamatot.

A `directionsCheck()` függvény fő feladata, hogy leellenőrizze a megadott útvonalpont-mezőket, és ha mindent rendben talál, tovább haladjon az útvonaltervezésre (esetlegesen geokódolásra). Először azt vizsgálja, üresen maradt-e valamelyik mező: ha igen, meghívja a korábban említett `showMsgBox(szoveg, tipus)` függvényt és leállítja az útvonaltervezési folyamatot. Ha mindegyik mezőben van valamilyen tartalom, továbbhalad - azt ellenőrzi mindegyik mezőre (egy `for` ciklus segítségével), hogy a megadott tartalom koordináta, vagy pedig szöveges megadás. Ezt az ellenőrzést egy RegEx (Regular Expressions) objektum segítségével végzem. Ez képes egy szabály alapján alkotott minta vagy

karakterkombináció felismerésére egy adott szövegben (Regular expressions, 2022). A regex, ha talált a szabályoknak megfelelő mintát, egy tömböt ad vissza a részekre bontott találatokkal, ha pedig nem, értéke `null` lesz. A szövegen használt regex szabály a következő:

```
var regex = text.match(/^(-?[0-9]+(\.[0-9]+)?),\s*(-?[0-9]+(\.[0-9]+)?)$/);
```

Egy olyan koordinátapárt keres a `text` változó szövegében (ami jelen esetben a mező tartalma), melyben bármilyen, akár negatív jellel (`-`) megelőzött szám szerepelhet, az adott koordinátában a tizedesvessző karaktere pont (`.`), a két koordinátát egy vessző (`,`) mindenképp és akárhány tagoló jel választhatja el (pl. szóköz), majd következik a második koordináta, melyben szintén pont a tizedesvessző karaktere. A visszaadott tömbből külön elemként kiszedhető a két koordináta, ezeket mentem egy globális, tömb típusú változóba (`navipontok[]`), majd egy Leaflet marker pontra vonatkozó jelet is elhelyezek a térképen a megfelelő koordinátákon. Ennek jelét a `markerIcon()` függvény határozza meg, a globális változóként nyilvántartott útvonalpontok száma alapján (ha ez kettő, A és B jelet ad vissza, ha három, akkor a közbeeső útvonalpont jelét is visszaadja).

Ha a regex segítségével az derült ki, hogy a mezőben nem koordináta, hanem szöveges megadás van, tartalmát mentjük a `navipontok[]` tömbben, “keres“ nevű objektumként. Ezzel a szöveggel hívjuk meg a geokódolót (`pontKeres_geokod[szoveg]`), feltéve, hogy van internetkapcsolat a készüléken (ezt az `internetConnectionCheck()` függvény dönti el).

A geokódoló függvény egy objektumhoz tartozó szöveget vár egyetlen bemeneti paramétereként (pl. `navipontok[0].keres`), és futása végén az ORS geokódolójától visszakapott választ az adott pont “valasz” objektumaként adja vissza és menti (pl. `navipontok[0].valasz`). Ebben a válaszban van minden, az adott találatra vonatkozó információ, pl. a hely neve, település, megye, régió, ország, kontinens stb., és persze a pont koordinátapárja, melyet később fel tudunk használni az útvonaltervezéshez. A geokódoló függvény, mivel az ORS-től kéri le az adatokat, nyilván tartja a geokódolandó pontok számát, és csak akkor lép tovább, ha mindegyikkel végzett (a geokódolandó pontok száma: `geocoding_queue = 0`). Erre azért van szükség, hogy ne történhessen meg az, hogy az ORS geokódolójától még nem érkezett válasz, viszont az alkalmazás már a következő pontra lép, mely az útvonaltervezés és megjelenítés függvénye lesz (`getDirections()`). Ha mindennel végzett, felhelyezi az útvonal kezdő- és célpontjának (esetleg közbeeső pontjának) markerjét a térképre, valamint beírja az oldalpanel útvonal-információkat tartalmazó paneljébe a kiinduló

és végpont geokódolt, OSM-en (vagy ritkább esetben egyéb adatbázisokban, pl. whosonfirst, openaddresses, geonames) szereplő helyes nevét.

Az útvonaltervezés és útvonal-megjelenítés fő függvénye a `getDirections()`. Ez a függvény egy *POST* típusú, aszinkron lekérést indít az ORS Directions Service GeoJSON szolgáltatás felé, a teljes GeoJSON formátumú választ pedig feldolgozza, és meg is jeleníti a térképen. A lekérés üzenettestébe (Body) kerül az összes, útvonaltervezést érintő paraméter, melyeket globális objektumként tárolok, majd a lekérés indítása előtt a `body` változóban szintaxisnak megfelelően betöltök és összefűzök. Az érkező választ csak akkor dolgozza fel, amikor a lekérés megérkezik és státuszkódja megfelelő (tehát sikeres a lekérés). Aszinkron jellegéből adódóan a lekérés egy független szálon fut, így a teljes alkalmazás futása nem akad meg (pl. amíg egy útvonaltervre várunk az ORS felől, a felhasználó nyugodtan interakcióba léphet az alkalmazás felületével). A függvény minden, sikeres érkező válasz feldolgozása előtt kiolvassa a fejlécben található adatokat, és:

- leellenőrzi, hány lekérés kérhető le még az adott napon (ha elfogyott a lekérések száma, értelemszerűen nem haladhat tovább a függvény és le kell kezelni az eseményt, erről értesíteni a felhasználót egy üzenetablakban)
- leellenőrzi, az ORS-től érkező státuszkód is megfelelő-e (a `statusCodeHandler()` függvény segítségével).

Ha minden rendben, jöhet a tényleges adatfeldolgozás. A válaszban minden teljes útvonal a *features* tömbben van külön elemként (ha az alternatív útvonalak száma 2, értelemszerűen három darab *feature* objektumot kapunk vissza). Minden ilyen *feature* objektumban van egy *geometry* elem, mely az útvonal töréspontjainak koordinátapárjait tartalmazza, valamint egy *properties* elem, mely egyéb információt hordoz az útvonalszegmensekről (minden útvonalpont közötti megtervezett szakasz egy szegmens), a navigáció lépéseiről, valamint az útvonal hosszáról és az út becsült időtartamáról. Ezen adatokat felhasználva a lépéseket tartalmazó tömbön iterálok, melyben az adott lépésszakasz töréspontjainak koordinátapárjait egy ideiglenes `reszlet` tömbbe mentem, majd az így felépített polyline-t hozzáadom a Leaflet térképrétegéhez – ez által megjelenik az adott lépésszakasz a térképen. A térképre rakásnál minden lépésszakaszhoz egy tooltip-et (a szakaszra mutató, felugró ablak) rendelek a `.bindTooltip()` metódussal, melynek tartalmát rövid HTML-kódok és az információt hordozó mezők összefűzésével határozom meg. Ezen adatok közé tartozik a lépésszakasz:

- instrukciója és annak ikonja, pl.: ↰ Forduljon be balra erre: Kéri út
- a `distanceFormatter()` függvény által formázott hossza, pl.: 822 m

- a `timeFormatter()` függvény által formázott időtartama, pl.: 1 perc 27 mp

A tooltip polyline-hoz való rendelésével az egérkurzor szakasz fölé mozgatásával megjelenik a tooltip, viszont alaphelyzetben a szakasz közepére mutat. Mivel a tooltip a lépést is leírja (pl. forduljon balra, hajtson be a körforgalomba, tartson jobbra stb.), navigációs szempontból érthetőbb, ha ez a tooltip a szakasz elejére mutat, ahol a tényleges lépést meg kell tenni. Ezt az adott polyline szakaszra beállított `mousemove` eseménykezelő függvény végzi el, mely a tooltipet a szakasz legelső csomópontjára helyezi:

```
t.on('mousemove',e=>{
    e.target.getTooltip().setLatLng(e.target.getLatLngs()[0]);
    });
t.on('mouseover',e=>{
    e.target.setStyle({color: "red", weight: 5});
    });
t.on('mouseout',e=>{
    e.target.setStyle({color: "#3388ff", weight: 3});
    });
```

A `mouseover` és `mouseout` eseménykezelők a vonal színét és vastagságát változtatják az egérkurzor szakaszra fel- és szakaszról lemozgatásánál.

Az adott lépésszakasz térképen való megjelenítése után az itiner táblázat konkrét sorainak betöltése következik. Ebben a táblázatban az útvonaltervezés teljes lefutása végére az útvonal minden egyes lépése, instrukciója helyet kap. Az itiner táblázat törzsére a `table_directions_body` azonosítóval hivatkozok. Ezt az elemet minden útvonaltervezés elején lenullázom, hogy ne tartalmazzon egy sort sem. Az aktuális lépésszakasz adatait új sorként adom hozzá az itiner táblázathoz, az `.insertAdjacentHTML()` függvénnyel:

```
document.getElementById('table_directions_body').insertAdjacentHTML('before
end','...tartalom...');
```

A táblázat egy sorában helyet kap a lépés sorszáma, az instrukció (ikonnal), a szakasz hossza és a becsült időtartam (az adatok formázásáért függvények felelnek, hasonlóan a tooltip tartalmának legenerálásánál). A célpontba megérkezés sora egy feltétel segítségével külön formázást kap (félkövér betűtípust, valamint eltűnnek a hossz/idő oszlopok cellái, mivel ezek 0 m és 0 mp értékűek az útvonal végpontja miatt). A táblázat sorai (`<tr>` elemek) megkapják az `onmouseover=` és `onmouseout=` paramétereket: ezek segítségével megadható egy JavaScript kódrészlet vagy függvényhívás, mely mindig lefut, ha a felhasználó a kurzort a sor bármely része fölé helyezi, vagy éppen a kurzorral arról lefelé tart. Az ilyen esetben meghívandó folyamatként a `highlightSegment(i)` és `unHighlightSegment(i)` függvényeket adom

meg, dinamikusan (az `i` változó mindig a sor/lépés száma). Ennek köszönhetően az adott szakasz kiemelődik a térképen (vonalvastagság és vonalszín ideiglenes megváltozásával).

Az itiner táblázat betöltése után a felhasználói felületen megjelenik az útvonal kiinduló és végpontja, a teljes hossz, a teljes időtartam, az útvonaltervezés hivatkozása, attribúciója is (ORS, OSM), valamint a kérés ideje is. Végső lépésként a Leaflet térképet a felhelyezett útvonalhoz mozgatjuk, méghozzá az útvonalt befoglaló téglalap sarokpontjai alapján. Ez az információ a válaszban érkezik meg, méghozzá a *bbox* tömbbel. Ez a tömb négy elemmel rendelkezik: délnyugati sarokpont hosszúsága és szélessége, északkeleti sarokpont hosszúsága és szélessége. A térkép befoglaló téglalapra helyezése a következőképp történik (melyet bármilyen, térképen megjelenítendő válasz érkezésénél ugyanígy használok):

```
var bb1=L.latLng([r.bbox[1],r.bbox[0]]); //lat, lon
var bb2=L.latLng([r.bbox[3],r.bbox[2]]);
map.fitBounds(L.latLngBounds(bb1, bb2));
```

Ezen a ponton véget ér a függvény szigorúan az útvonaltervezésre és annak megjelenítésére szolgáló szakasza és következik a magassági lekérés függvényének meghívása, melynek segítségével megtudhatjuk majd, hogy az útvonal során milyen magasságváltozást kell leküzdenünk. Még mielőtt a magassági lekérést indítom, át kell számítanom az útvonal polyline-ját úgy, hogy csomópontjainak száma ne lépje túl a 2000-ret (az ORS vonalas magasságiadat-szolgáltatása elutasítja a kérést, ha ennél több csomóponttal rendelkezik a polyline). Az átszámítás szükséges a lineáris X-osztásközű magassági profil megjelenítéséhez is. A nyers, ORS által generált útvonal csomópontjai nem azonos távolságokban helyezkednek el az útvonal mentén, és ennek köszönhetően a megjelenített grafikon az X tengely mentén (távolság) változó torzulású lenne. E két probléma megoldásaképp az útvonalat átszámítom/egyszerűsítem a Turf.js `turf.along()` függvényével. Ez a függvény visszaad egy pontot (melyet egy koordinátapár definiál), mely a megadott polyline-on végighaladva, a kezdőponttól megadott távolságra helyezkedik el (Turf.js Docs, 2022). Ezt egy `for` ciklusba ágyazva végigfuttatom úgy, hogy a távolság egy osztásköznyivel növekszik, minden futásnál. Az `osztaskoz` változót egy külön feltételrendszer számolja ki az útvonal teljes hosszának függvényében úgy, hogy végeredményben az útvonal darabjainak száma sose lépje túl a 2000-ret, és hogy a grafikonon megjelenítendő magassági adatok részletessége ésszerű legyen. A ciklus futása közben a megkapott pont koordinátapárját egy új, globálisan elérhető tömb változóban mentem, melyben így felépítek egy új, átszámított útvonal polyline-t. Az első és utolsó csomópontot manuálisan adom hozzá a felépítendő polyline tömbhöz az átszámítás előtt

és után (az első pontot ismerjük, ezért nem kell kiszámolni; az utolsó pont pedig a maradék nélküli távolság alapú felosztás miatt eltűnne a polyline végéről).

```
elsopont = turf.along(turfline, 0, turfoptions);
current_polyline_osztott[0] = [elsopont.geometry.coordinates[0],
elsopont.geometry.coordinates[1]];
for (var i = osztaskoz, db = 1; i<tav/1000; i = i+osztaskoz) {
    var along = turf.along(turfline, i, turfoptions);
    current_polyline_osztott[db] = [along.geometry.coordinates[0],
    along.geometry.coordinates[1]];
    db = db+1;
}
utolsopont = turf.along(turfline, tav, turfoptions);
current_polyline_osztott[db]=[utolsopont.geometry.coordinates[0],
utolsopont.geometry.coordinates[1]]
```

Az így kapott polyline az átszámításnak köszönhetően már biztosan kevesebb, mint 2000 csomóponttal rendelkezik, így alkalmas arra, hogy lekérjem a magassági adatokat az ORS szolgáltatásától. Ezért a folyamatért a `getElevation_line()` függvény felel. Ez a függvény a `getDirections()` függvényhez hasonlóan *POST* típusú, a lekérés alapvető működése hasonló. A válasz megérkezésekor először lenullázom és frissítem a magassági profil grafikonját (`elevChart.data=null;` `elevChart.update()`), majd következik az adatbetöltés.

Az ORS az útvonal minden csomópontjára vonatkozó magassági adatot ad vissza, mely a *geometry*, azon belül a *coordinates* tömbben helyezkedik el. Egy `for` ciklus segítségével végigmegyek ezen a tömbön, miközben a magassági adatot és a kezdőponttól számított távolságot mentem a Chart.js grafikonjának adattömbjeiben. Az `elevChart` grafikon megjelenítési beállításaiban definiáltam, hogyan szeretném az útvonal címkézni távolságát (X, vízszintes tengely), a tengely mentén: a távolság rövid (< 3 km hosszú) utak esetében métert, minden egyéb esetben pedig kilométer értékeket használ. Ez véleményem szerint megkönnyíti az adatok értelmezését rövid utak esetében (pl. egy 1 km-es útvonalnál nem 0,1 km, 0,2 km stb., hanem 100 m, 200 m stb. címkéket lát a felhasználó). Kényelmi, de szintén hasznos beállítások a grafikon interakcióját segítő beállítások, melyeknek köszönhetően a felhasználó a grafikon X tengelye mentén bárhol mozgathatja az egérkurzort (a magassági vonalon, az felett vagy az alatt is), a távolsághoz rendelt magasság felugró tooltipje mindig megjelenik a grafikonon és a térképen egy jellel együtt az útvonal mentén, ahová vonatkozik az adott magassági adat. A betöltés után a felhasználói felületen megjelenik a magassági adatok forrásának hivatkozása is.

6.3 Izokrónkészítés

Az izokrónkészítés folyamata az útvonaltervezéshez hasonló ellenőrző függvényeket futtat, geokódol, lekéri az izokrónkészítő-szolgáltatástól az adatokat, feldolgozza és megjeleníti azokat. Globális változókat is manipulál, a legfontosabbak: a poligongenerálás módszere (izókrón- vagy izodisztanskészítés, idő vagy távolság alapján), az izokrón-középpontokat tartalmazó `isopontok[]`, és az izokrón-poligonokat gyűjtő `isopolygons[]` tömbök.

Elsőként az `isochroneCheck()` függvény fut le, amiben az izokrónközéppont beolvasása, ha a bemenet szöveg, a geokódolás zajlik.

Az izokrónkészítés fő függvénye a `getIsochrones()`. Ez a függvény egy *POST* típusú, aszinkron lekérést indít az ORS Isochrones Service szolgáltatása felé, a GeoJSON formátumú választ pedig feldolgozza, és meg is jeleníti a térképen. A lekérés üzenettestét (Body) a korábbiakban részletezett módon hozom létre, természetesen a megfelelő paraméterekkel (kiindulópont, módszer és a dinamikusan számolt intervallumok). Még mielőtt a lekérés megtörténne, mindenképp lefut a `refreshIsochronesOptions()` függvény, mely kiolvassa a felhasználó által beállított paramétereket (módszer, maximális idő vagy távolság és az izokrónok száma), valamint lineáris intervallumokat számol a maximális idő vagy távolság és az izokrónok száma alapján, majd menti azt az `isoranges[]` globális tömbben (ami a lekérés üzenettestében kötelező paraméter). Ehhez kapcsolódik a `refreshIsochroneRangeLimits()` függvény is, mely az útvonalprofil (autós, biciklis, gyalogos stb.) megváltoztatásakor mindenképp frissíti a felhasználói felületen beállítható csúszkák minimum, maximum értékét és felbontását (ezzel megelőzöm, hogy a felhasználó pl. 3 órás izokrónt generáljon autós profillal – az ORS limitációinak megfelelően az autós profil izokrónjainak maximális ideje 1 óra, így ez a lekérés hibaüzenettel végződne).

Az útvonaltervezés függvényéhez hasonlóan itt is lefut a lekérési limit ellenőrzése, valamint, ha sikeresen megérkezett az izokrónokat tartalmazó válasz, a bal panelen lévő táblázat is ürül. A válasz feldolgozásánál annyi *feature* elemet kapunk, ahány izokrónt kértünk – ezen a tömbön iterálok végig. Az adott poligon elem csomópontjai alapján létrehozok egy poligont, melyet az `isopolygons[]` tömbbe helyezek. Még mielőtt hozzáadnám a térképhez, szükséges a kapcsolódó tooltip megformázása és poligonhoz való hozzárendelése. Minden poligonhoz tartozik egy időtartam vagy egy távolság attól függően, izokrónokat, vagy izodisztansokat kértünk le. Ezt a következő kódrészlet mérlegeli, és hozza létre a tooltipeket a megfelelő formázással (`timeFormatter()` és `distanceFormatter()`):

```
if (!preference_isotype.checked) {
    // ido
    isopolygons[x] = L.polygon(nodes, {color: polygonColor(x)}).addTo(lg)
        .bindTooltip('Idő: '+timeFormatter(r.features[x].properties.value),
        {sticky: true}).bringToBack();
} else if (preference_isotype.checked) {
    // tavolsag
    isopolygons[x] = L.polygon(nodes, {color: polygonColor(x)}).addTo(lg)
        .bindTooltip('Távolság:
        '+distanceFormatter(r.features[x].properties.value),
        {sticky: true}).bringToBack();
};
```

A poligonok színét a `polygonColor()` függvény szimplán, a poligonok száma alapján határozza meg. A `.bindTooltip()` beállításaként használt `sticky: true` paraméternek köszönhetően az egérkurzor poligon fölé mozgatásával a tooltip nem a poligon súlypontjában, hanem az egérkurzorhoz csatolva jelenik meg és azzal együtt mozog. Ennek köszönhetően könnyebb a poligonhoz tartozó értékleolvasás. Végül, a `.bringToBack()` függvény az éppen érintett poligont a térképen a leghátsó rétegre helyezi (ha ezt nem tenném, alaphelyzetben egymásra kerülnének a poligonok, és hierarchiájuknak köszönhetően csak a legutoljára felhelyezett poligon tooltipjét látnánk, mivel az lefedi az összes többi poligont).

A poligonok megjelenítése után a bal oldali panel táblázatát töltöm fel a poligonok sorszámával, távolságértékével, és a hozzájuk tartozó színnel kitöltött cellával. Az első oszlopban lévő sorszám mellé elhelyezek egy `checkbox` típusú `input` HTML-elemet is, melynek `onchange=` paraméterében az `isochroneCheckboxChange()` függvényt dinamikusan, az adott poligonra hivatkozva állítom be. Ez a függvény akkor hívódik meg, amikor a felhasználó ezen checkboxok egyikét ki-, vagy bekapcsolja. Futása közben eltünteti, vagy megmutatja az érintett poligont. A függvény kódja a következő (`checked` az adott checkbox állapota booleanként, `i` a poligon sorszáma):

```
function isochroneCheckboxChange(checked, i) {
    if (checked == false) {
        map.removeLayer(isopolygons[i]);
    } else {
        map.addLayer(isopolygons[i]);
        for (var j = 0; j < isopolygons.length; j++) {
            isopolygons[j].bringToBack();
        }
    }
}
```

Ez a funkció hasznos, ha több megjelenített poligon közül valamelyiket szeretnénk kikapcsolni (pl. 15, 30, 45 és 60 perces poligonok esetében mégsem szeretnénk látni a 15 és 45 perceseket), így vizuálisan kevésbé terhelt térképen vizsgálódhatunk.

Végső lépésként kikerül a kiinduló pont (középpont) markerje a térképre, a Leaflet térkép nézete a célterületre mozog, valamint kikerülnek a hivatkozások és a lekérés időpontja a felhasználói felület egyes részeire.

6.4 Technikai jellegű függvények

Az alkalmazásban olyan függvények is helyet kaptak, melyeket a teljes programkód során többször hívok meg, megkönnyítve így a többszöri használatot. Ezen függvények főként technikai jellegűek, pl. a válaszban érkező státuszkódok feldolgozására, vagy az időtartamok és távolságok megfelelő formázására irányuló programkódok. Az ilyen jellegű függvények közül néhányat ebben a szakaszban mutatok be.

6.4.1 A lekérés esetén használt útvonalprofilt karbantartó függvény

`setProfile(prof)` függvény:

A `profile` globális változót tartja naprakészen, hogy a lekérések mindig a megfelelő profillal történjenek. Az alkalmazás indulásánál fut (ekkor az elmentett cookie-ból olvassuk ki a profilt és azt állítjuk be), illetve minden alkalommal, amikor a felhasználó megváltoztatja az útvonalprofilt (pl. autóról biciklire vált). A függvény fő része a `switch` vezérlési szerkezet, mely az adott profil beállítása során elmenti a profil megjegyzésére szolgáló cookie-t a készüléken, valamint az útvonal- és izokrónkészítési beállításokat is frissen tartja a felületen (pl. biciklis és gyalog profil esetén kikapcsolja autópályát elkerülendő lehetőség checkbox-ját). Minden esetben meghívja a `refreshIsochronesOptions()` függvényt, mely beállítja az izokrónkészítés paramétereinek alsó és felső határait (pl. autós izokrón tervezésénél a maximum idő 1 óra, míg gyalogosnál 20 óra lehet).

6.4.2 Lekérési státuszkód-ellenőrző függvények

`statusCodeHandler(code)` és `statusCodeHandler_elev(code)` függvények:

Az Openrouteservice-től érkező válasz része a státuszkód, melyet a túloldali szerver generál le és küld vissza minden lekérés esetén. A státuszkód egyértelműen azonosítja, hogy az adott lekérést sikeresen feldolgozta-e, vagy valamilyen probléma lépett fel a feldolgozás során (azt is, milyen az adott probléma, pl. a szerver túlterhelt, vagy a lekérési paraméterek nem szintaxisnak megfelelőek és ezért nem teljesíthető a lekérés). Ez a két függvény ezt a státuszkódot értelmezi (az ORS dokumentációja alapján), és függvényértékként egy igaz/hamis boolean értéket ad vissza, szükség esetén üzenetablakot mutat, melyben szöveges formában értesíti a felhasználót a problémáról. Ha a visszatérő boolean érték hamis, az alkalmazás egyéb részei ezt lekezelik, és az adott folyamat megáll (pl. az útvonaltervezés függvényében egy hibás

válasz érkezése esetén nem haladunk tovább az útvonal megjelenítésére, mert az nem létezik). Az olyan alkalmazásokban, melyek egy távoli szerverrel kommunikálnak, elengedhetetlen egy ilyen típusú függvény megléte és használata.

6.4.3 Idő- és távolságformázó függvény

`timeFormatter(t)` és `distanceFormatter(d)` függvények:

Az útvonaltervező alkalmazás természetéből kifolyólag több helyen jelennek meg idő és távolság értékek a felhasználói felületen. Az ORS által visszaadott értékek mértékegységei több esetben előre beállíthatóak a lekérésben, az alkalmazás viszont ezeket mindig másodpercben és méterben kéri le (így a kapott adatok pontosságából nem veszít). Ez a két függvény az idő (mp) és távolság (m) értékeit kapja meg bemeneti értékként, és megformázásukért felel (a megformázott szöveg típusú értéket adja vissza). Az időformázó a bonyolultabb függvény: ebben először kiszámolom, az adott időtartamban hány óra, perc és másodperc van, majd egy többlépcsős feltételrendszer szerint dől el, hogy a formázásnál az óra, perc és másodperc értékei közül melyiket jelenítem meg úgy, hogy ha az adott részből 0 van, elhagyom. Pl.:

- 121 másodperc = “2 perc 1 mp”
- 3630 másodperc = “1 óra 30 mp” (0 perc, így az nem jelenik meg)
- 3660 másodperc = “1 óra 1 perc” (0 másodperc, így azt elhagyjuk)
- 3690 másodperc = “1 óra 1 perc 30 mp”

A távolságformázó függvény egyszerűbb: ha a bemeneti érték (méter) kisebb, mint 1000, akkor csupán megkapja az “m” rövidített mértékegységet, ha nagyobb, akkor viszont 1000-rel osztom és két tizedesjegy-pontosságra kerekített kilométert jelenítek meg. Pl.:

- 650 méter = “650 m”
- 20564 méter = “20.56 km”
- 20565 méter = “20.57 km”

6.4.4 Elérhető napi lekérések számának ellenőrzése

`rateLimitCheck(n, milyen, resettime)` függvény:

Az ORS által visszaadott válasz fejlécében megtalálható a még hátra lévő napi kérések száma, valamint az is, hogy mikor várható a napi számláló újraindulása. Ez az egyszerű függvény ezeket megkapja bemeneti értékként: ha a kérések száma 0 vagy egy kisebb szám, meghívja a `showMsgBox()` függvényt a megfelelő figyelmeztetéssel (szöveggel), mellyel tudatja a felhasználó felé, hogy nem lehet többet lekérni, valamint azt is, hogy mikortól lehet majd. Az időpont, mikor az ORS napi számlálója újraindul Unix-idő formátumú, mely az Unix epoch, azaz 1970. január 1. 00:00:00 UTC óta eltelt másodpercek száma. Ezt a számot az

`unixTimestampSolver()` függvény számolja át év/hónap/nap/óra/perc/másodperc formátumra és adja vissza azt. A visszaadott, formázott értéket mutatja a felugró üzenetablak.

6.4.5 Internetkapcsolat-ellenőrzés

`internetConnectionCheck()` függvény:

Egy egyszerű ellenőrzés, mely a készülék böngészője által nyilvántartott `window.navigator.onLine` nevű boolean változót ellenőrzi. A változó hamis értéket vesz fel, ha a böngészőnek nincs hálózati kapcsolata. Hátránya, hogy ha a készüléknek van is hálózati kapcsolata, viszont a hálózatból kifelé már nem érhető el az internet, a változó így is `true` marad. Ha ez a változó hamis, értesítem róla a felhasználót egy üzenetablakban, valamint a függvény visszatérő értéke is hamis lesz, így az alkalmazás adott folyamata le tud állni.

6.4.6 Üzenetablak megjelenítése (hiba, figyelmeztetés vagy sikeres futás esetén)

`showMsgBox(szoveg, tipus)` függvény:

Az alkalmazás futása során szükséges tudatni a felhasználóval, ha egy folyamat során hiba történt – ez a függvény erre szolgál. A két bemeneti változójával bárhonnan meghívható, így bármilyen üzenetet ki lehet íratni, egyszerűen. A `tipus` változó `error`, `warning` vagy `success`-re állításával idézhető elő a háromfajta üzenetablak egyike, melyek a probléma súlyosságát, vagy a sikeres folyamatot hivatottak jelezni háttérszínükkel (piros, narancssárga és zöld). A függvény kezeli az üzenetablak megjelenésének és eltüntetésének animációját is, stílusszabályok segítségével.

6.4.7 Háttérben futó folyamat jelzése a felhasználó számára

`showLoadingIcon()` és `hideLoadingIcon()` függvények:

Minden lekérés feldolgozása eltart egy ideig, néha akár másodpercekig, mielőtt az eredmények megjelennek és a felhasználó észleli ezeket. Az útvonaltervezés folyamata egy rendkívül összetett folyamat, mely során akár 5 lekérés is indulhat (három geokódolás, egy útvonaltervezés és egy magasságiadat-lekérés). Ez a két rendkívül egyszerű függvény megjelenít egy forgó ikont az oldalpanelen (amíg a folyamat fut a háttérben), majd, ha a folyamat sikeresen (vagy sikertelenül) befejeződik, eltünteti az ikont. Ezeket a programkód során több helyen hívom meg, értelemszerűen a `showLoadingIcon()` függvényt egy folyamat megkezdése előtt (pl. útvonaltervezés, izokrónkészítés, vagy fordított geokódolás), a `hideLoadingIcon()` függvényt pedig minden sikeres folyamat végén, valamint minden folyamat megszakadásakor vagy sikertelen futásakor is meghívom (pl. az érkező státuszkód nem megfelelő, vagy nincs internetkapcsolat).

6.4.8 Cookie-olvasó függvény

`getCookie(cname)` függvény:

Szinte az interneten böngészhető weboldalak mindegyike cookie-kat (süti) ment el a felhasználó számítógépén, melyekkel bizonyos adatokat tud rögzíteni (`név = érték` formátumban), esetleg később kiolvasni és felhasználni azokat. Az alkalmazásban ilyen például az útvonalprofil (autós, biciklis vagy gyalogos) mentése, melyet a weboldal megnyitásakor megpróbál kiolvasni, és ezáltal beállítani az utolsó ismert értékre. Ez a kényelmi háttérfunkció akkor hasznos, ha valaki többször, viszont pl. mindig csak biciklis útvonalak tervezésére használja a weboldalat – rendkívül bosszantó lenne, ha minden megnyitásnál visszaugrana autós profilra. A függvény visszaadja egy adott nevű cookie tartalmát (értékét), szövegként. A kódrészlet forrása: W3Schools (https://www.w3schools.com/js/js_cookies.asp).

6.5 Felhasználói felület, formázás, megjelenés

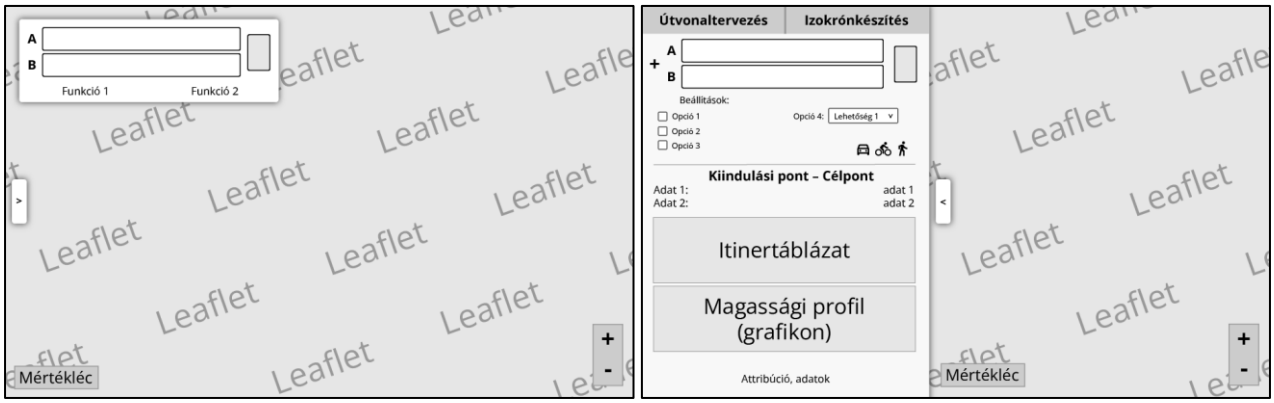
Az alkalmazás felhasználói felületének kialakításánál és formázásánál, stilizálásánál egy modern, letisztult kinézetű, reszponzív weboldal létrehozása volt a célom. Ezek mellett fontos volt, hogy a weboldal jól használható, kellő interaktivitású legyen. Az interneten elérhető térképszolgáltatások és útvonaltervezők legtöbbje olyan kifejlett, évek óta meglévő, interaktivitást segítő funkciókat kínál, melyek mára a felhasználó számára alapvetőnek, elvárhatónak tűnnek. Ilyen például az, hogy a felhasználó az útvonaltervezés valamely útvonalpontját a térképre kattintva is elhelyezheti (így nem csak szöveges bevitellel tudja megadni), vagy az, hogy a keresési paraméterek megváltoztatása, átírása esetén ne kelljen minden alkalommal az indítás gombra kattintani – automatikusan frissüljön a megjelenített adat, ha változik egy paraméter, vagy a felhasználó az Enter billentyűt nyomja le egy mező szerkesztése után. A Leaflet térképobjektum rendkívül fejlett és ennek köszönhetően a térkép interaktivitását nagy részben kezeli (térkép mozgatása, nagyítása, adatmegjelenítés markerekkel, polyline-okkal, tooltipekkel stb.), viszont a webes alkalmazás teljes felhasználói felületét manuálisan szükséges interaktívvá tenni, a HTML, CSS és JavaScript programkódok megfelelő kombinációjával. Nem elhanyagolható a mobilos környezetre való optimalizálás sem: egy útvonaltervező alkalmazásra nagy eséllyel lehet szükség mobilos környezetben is, pl. terepen, amikor nincs lehetőség az asztali számítógépen vagy laptopon való útvonaltervezésre. A weboldal reszponzív kialakítása (*responsive web design*) lehetővé teszi, hogy az oldal a felületelemek méreteinek változtatásával rugalmasan alkalmazkodjon különféle megjelenítőkhöz, kijelzőkhöz (asztali számítógépes és mobiltelefonos méretekhez egyaránt).

Ennek köszönhetően a weboldal bármilyen környezetben olvasható, könnyen használható marad, nem szükséges a teljes oldal nagyítása, görgetése.

Az oldalt felépítő HTML-elemek formázása egyidejűleg zajlott az alkalmazás építésével és programozásával, egyes funkciók megírása közben. A HTML dokumentumban az elemek stílusát közvetlenül a `style=””` paraméterrel, az osztályok megadásával (`class=””`), vagy pedig a konkrét elem-azonosítókra hivatkozva definiálom.

Az alkalmazás felületét úgy alakítottam ki, hogy a térképobjektum alaphelyzetben kitölti a teljes rendelkezésre álló helyet, és felette egy kis “lebegő” panel kap helyet a bal felső sarokban. Ezen a panelen csupán a legalapvetőbb funkciók érhetőek el: ilyen a kiinduló- és célpont megadása, valamint az indítás. Útvonaltervezés indítása esetén megnyílik egy oldalpanel a bal oldalon, melyen látható az összes beállítható útvonalkeresési paraméter, valamint a generált útvonal minden részlete is itt jelenik meg (kiinduló- és célpont geokódolt neve (vagy koordinátapárja), útvonalhossz és időtartam, az itiner táblázat, a magassági profil grafikonja, valamint egyéb attribúciós és technikai adatok). Ez az oldalpanel egy dedikált, középen elhelyezkedő gombbal bármikor nyitható és csukható.

A felület vázlatrajza:



1. ábra: Felhasználói felület vázlata: alaphelyzet csukott (balra), valamint nyitott oldalpanellel (jobbra) (saját illusztráció; autó, bicikli és gyalogos ikonok forrása: Google Fonts / Icons)

A HTML dokumentum törzsét (*Body*) öt logikai elemre bontottam:

- `<div id=”lebego_start_ablak” ... >`: a csukott oldalpanel esetén bal felső sarokban látható panel,
- `<div id=”sidebar” ... >`: az oldalpanel,
- `<div id=”sidebar_openclose” ... >`: az oldalpanel nyitására és csukására szolgáló gomb eleme,
- `<div id=”mapdiv” ... >`: a Leaflet térképobjektum,

- `<div id=”uzenetablak” ... >`: a weboldal alján időnként megjelenő üzenetablak, mely hiba vagy figyelmeztetés esetén jön elő.

Ezen elemek hierarchikus jellegük miatt több, mélyebben lévő elemet is tartalmaznak. Ilyen például a legösszetettebb fő elem, az oldalpanel (`sidebar`). Az oldalpanelen szerkezetileg elkülönül a felső funkcióválasztó fül (mely az oldalpanelt útvonaltervezés, illetve izokrónkészítés módba kapcsolja), maga a két tartalmi egység (`navi` és `isochrone`) felülete az azokhoz tartozó paraméterekkel és panelekkel), valamint az alsó, szerzői jogi és egyéb információk kijelzésére szolgáló `sidebar_bottom` elem. Ahhoz, hogy a magasabb szinten lévő elemek tartalmát könnyen és esztétikailag megfelelően rendezzem el, a befoglaló elemek többsége `flexbox` típusú megjelenítést használ. Ennek köszönhetően a bennük lévő elemek mindig kitöltik a befoglaló elem által kínált helyet (W3C, 2018). A `flexbox`-ok használata nagyban elősegíti a reszponzív weboldal kialakítását, amikor egy adott befoglaló elem mérete dinamikusan változhat. Az oldalpanelen látható még a háttérben futó folyamat jelzésére szolgáló ikon is, melynek láthatóságát egy JavaScript függvény irányítja (lásd 6.4.7 szakasz).

A CSS-szabvány által kínált osztálydefiníció lehetővé teszi, hogy ún. pszeudoosztályok segítségével egyes elemek bizonyos események hatására automatikusan változtassák az elemhez rendelt stílusosztályt. Ilyen esemény lehet pl. egy bemeneti mező elem aktiválása (`:focus`), vagy az egérkurzor elem fölé való mozgatása (`:hover`). Ilyen szabályt alkalmazok többek között az útvonalpont-mezőknél is, melyek szerkesztése közben egy kék, kissé megvastagított körvonalat ad az adott mezőnek:

```
.ABfield:focus {
    border: solid 2px;
    border-color: #3333FF;
}
```

A weboldalon megjelenő szöveges elemek betűtípusa Open Sans, mely a Google Fonts betűtípusokat gyűjtő, ingyenes adatbázisban érhető el (https://fonts.google.com/). A felületen lévő kezelőszervek ikonjai és a térképobjektumban megjelenő ikonok egy része saját készítésű, egyes ikonok pedig a Google Fonts ikonokat gyűjtő, ingyen és szabadon hozzáférhető adatbázisból származnak (https://fonts.google.com/icons).

6.6 Felület optimalizációja mobilos környezethez

A reszponzív kialakítástól elvárható, hogy mobiltelefonos vagy tabletes környezetben is olvasható, használható legyen a felület. Ezért az optimalizációért a CSS-szabvány “media query” szabályai felelnek. `@media` szabállyal ellátott stílusosztály csak adott feltétel mellett lép

életbe. Ilyen feltétel lehet pl. a böngésző szélessége, a kijelző felbontása, annak magasságának és szélességének az aránya, vagy a készülék orientációja (álló/fekvő). (W3Schools, 2022)

Az alábbi CSS-részlet definiálja, milyen paraméterek legyenek érvényesek a `lebego_start_ablak` azonosítóval rendelkező HTML-elemre, ha a kijelző vagy böngésző szélessége nem haladja meg az 550 pixelt (ilyenkor mobilos környezetet feltételezünk; a webes alkalmazásban lévő lebegő panel az oldal tetejére kerül, és kitölti a teljes szélességet):

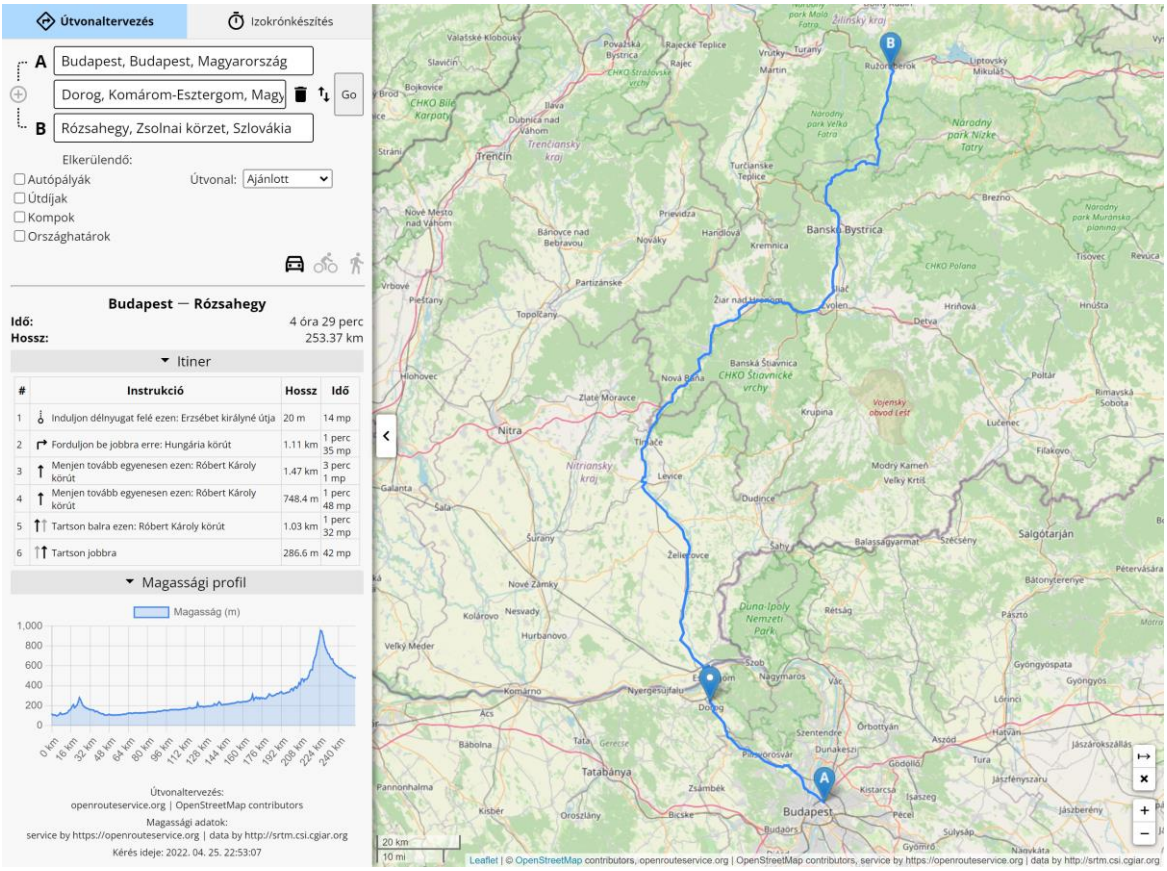
```
@media (max-width: 550px) {
    #lebego_start_ablak {
        position: absolute;
        top: 0px;
        left: 0px;
        width: 100%;
        border-radius: 0px;
    }
}
```

7. Bemutatás

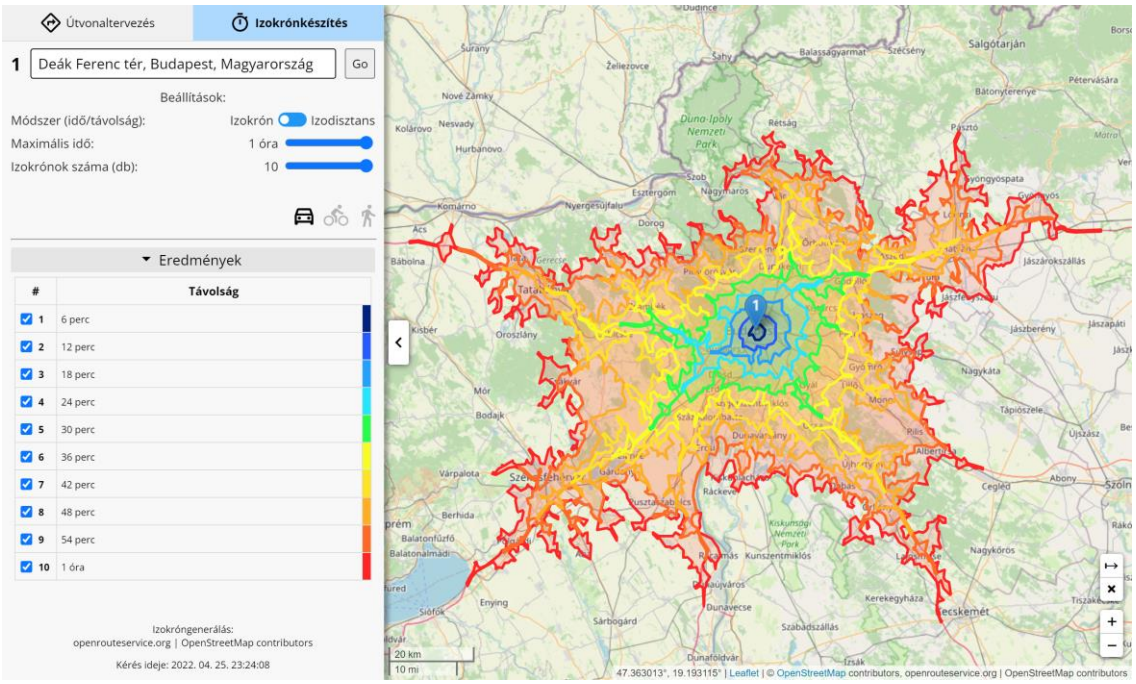
A dolgozatban fejlesztett útvonaltervező és izokrónkészítő alkalmazás képes:

- kettő vagy három pont közötti autós, kerékpáros vagy gyalogos útvonaltervezésre,
- az útvonaltervezés paramétereinek (útvonal típusa: ajánlott, legrövidebb és leggyorsabb, elkerülendő objektumok, határok) figyelembevételére,
- az útvonalra illesztett magassági profil előállítására és interaktív bemutatására,
- itiner táblázat előállítására, mely a navigáció lépéseit tartalmazza,
- megadható középpontú izokrónok és izodisztansok készítésére és megjelenítésére,
- térképen való többpontos szakasz- és távolságmérésre és iránymérésre,
- bármely pont szöveges bemenetű geokódolására, koordinátapár-alapú bemenetének értelmezésére, valamint térképen kurzorral való vagy helymeghatározás alapú lerakására,
- fordított geokódolásra (“Mi található itt?”).

Ebben a fejezetben az alkalmazás működését és használatát mutatom be, felhasználói szemszögből.



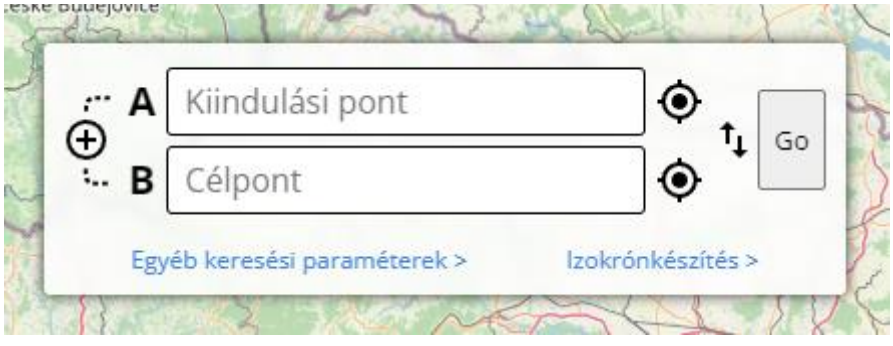
2. ábra: Útvonaltervezés: Budapest – Rózsahegy, Dorogon keresztül, autóval (háttértérkép: © OpenStreetMap contributors)



3. ábra: Izokrónkészítés: 10 db izokrón, 1 órás maximális idővel, autóval, Deák Ferenc tér középponttal (háttértérkép: © OpenStreetMap contributors)

Az alkalmazás asztali számítógépes környezetben való megnyitásakor az interaktív térkép fogad, melynek bal felső részében egy egyszerű, kétpontos útvonaltervezés indítható (lásd 4. ábra). A pontok (legyen az útvonalpont vagy izokrónközéppont) ötféleképpen adhatóak meg:

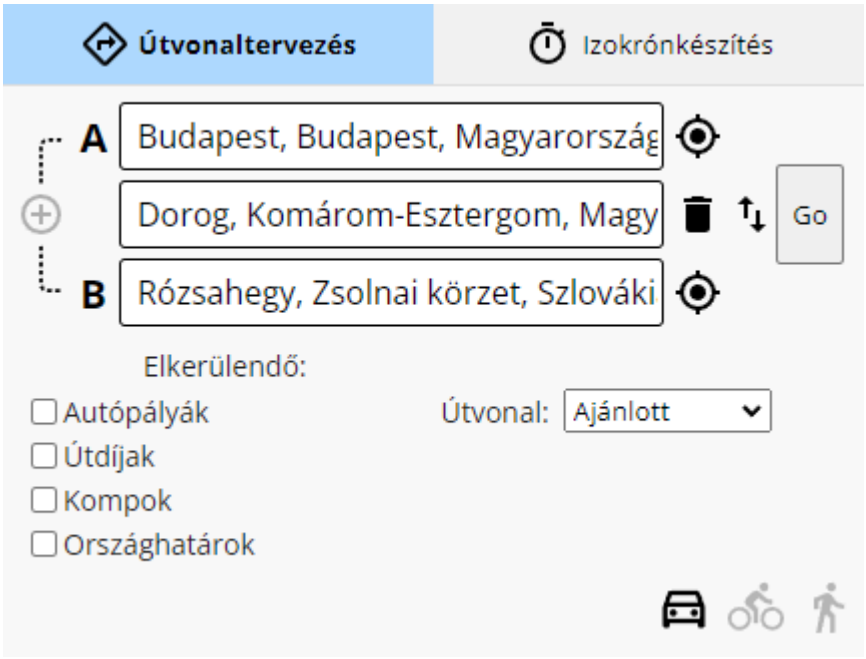
1. szöveges bevitellel (pl. "`Bécsi út 6, Budapest`"),
2. koordinátapár megadásával (szélesség, hosszúság sorrendben, pl: "`47.52505, 19.03676`"),
3. a pont bemeneti mezőjére, majd a térképre kattintva,
4. a térképre jobbegérgombbal-kattintás után megnyíló menüben (**A** Útvonal innen, **B** Útvonal ide, ⊙ Izokrón innen),
5. helymeghatározással ⌖ .



4. ábra: A bal felső panel, mellyel kétpontos útvonaltervezés indítható (háttértérkép: © OpenStreetMap contributors)

Az útvonaltervezés indításakor megnyílik az oldalpanel a bal oldalon, melyen finomítható az útvonaltervezés mikéntje.

Az oldalpanel szolgál az alkalmazás irányítására és az eredmények bemutatására. Megadható itt, milyen közlekedési eszközzel (autó, kerékpár vagy gyalogos), milyen szemléletű útvonalat tervezzünk (Openrouteservice által ajánlott, legrövidebb vagy leggyorsabb útvonal), valamint az is, milyen terepi objektumokat, vagy az útvonal költségét vagy módját befolyásoló tényezőt kerüljünk (autópályák, útdíjak, kompok vagy országhatárok elkerülése). Az oldalpanelen nyílik lehetőség egy közbeeső pont megadására is, mely hasonlóan a többi ponthoz, az említett módszerekkel definiálható. A két- vagy hárompontos útvonalterv iránya könnyen megfordítható. Itt jelenik meg a háttérben futó folyamatot jelző ikon is (amíg ez látható, új útvonalkeresés vagy izokrónkészítés indítása nem ajánlott).



5. ábra: Útvonaltervezés paraméterei (ikonok forrása: Google Fonts - Icons)

Útvonaltervezés vagy izokrónkészítés után a keresési paraméterek és beállítások alatti szakasz szolgál az eredmények bemutatására. Ilyen pl. az útvonal kezdő és végpontja, teljes időtartama és hossza. Ezen kívül itt látható még, egy-egy becsukható panelben az itiner táblázat és a magassági profil grafikonja is. Előbbi megmutatja a navigációhoz szükséges lépéseket és szakaszokat azok hosszával és időtartamával, valamint a szöveges instrukcióval (lásd 6. ábra). A táblázat sorai feletti egérkurzor-mozgatás közben az érintett szakasz a térképen is kiemelődik, és a szakaszra vonatkozó információk a térképen is megjelennek (ez a szakaszkiemelés a térképen, az útvonalon mozgatott egérkurzorral is megtörténik) (lásd 7. ábra).

Itiner

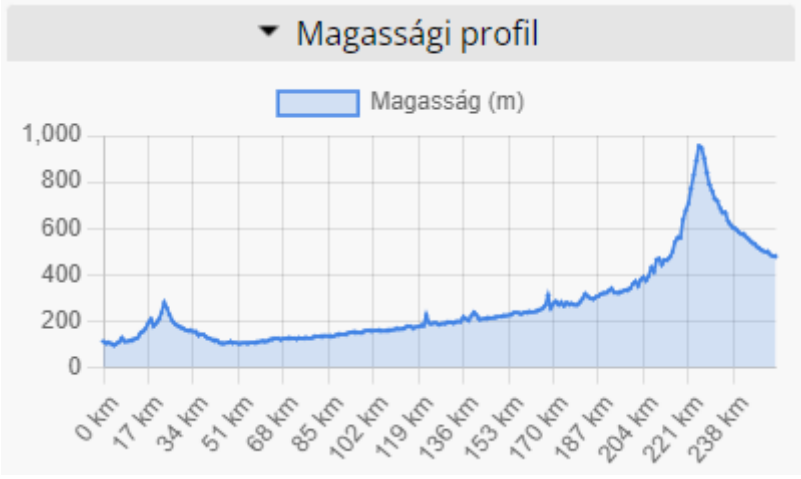
	a(z) második kijáraton, ezen: 10		
24	Forduljon be enyhén jobbra erre: Bécsi út, 10	3.61 km	4 perc 3 mp
25	Menjen tovább egyenesen ezen: Bécsi út, 10	2.33 km	2 perc 30 mp
26	Forduljon be jobbra erre: Gorkij utca	2.93 km	5 perc 9 mp
27	Forduljon be balra erre: Akácfa utca	142.7 m	34 mp
28	Forduljon be enyhén balra erre: Diófa utca	8.4 m	2 mp
29	**Megérkezett: Diófa utca (a jobb oldalon)**		
30	Induljon északnyugat felé ezen: Diófa utca	241.8 m	58 mp

6. ábra: Itiner táblázat

7. ábra: Az útvonal egy szakaszának kiemelése (háttértérkép: © OpenStreetMap contributors)

A magassági profil egy grafikonon mutatja meg, hogy milyen terepi viszonyok, jelen esetben magasságkülönbségek lépnek fel az útvonal során (lásd 9. ábra). Kifejezetten hasznos kerékpáros és gyalogos útvonalak készítésénél, mivel szépen láttatja, van-e az útvonal során egy-egy domb, magaslat, melyet fizikai erőnk segítségével kell majd leküzdenünk (autós utazás esetében egy bizonyos magasságkülönbségig ez kevésbé releváns). A grafikonon az egérkurzort mozgatva leolvashatjuk a tengerszint feletti magasságot az útvonal egy adott pontján. Az így felugró információ mellett a térképen is megjelenik a magassági adat, konkrét helyre vonatkoztatva (lásd 8. ábra). Ennek köszönhetően gyorsan le tudjuk ellenőrizni, hogy az útvonal során hol vannak azon pontok, melyeknél magasságváltozás várható. A magassági grafikon támogatja a tengerszint, 0 m alatti magasság megjelenítését is. Rövid útvonalak esetén (amennyiben az útvonal teljes hossza nem haladja meg a 3 km-t) a grafikon vízszintes tengelye (útvonalhossz) méter egységet vesz fel.

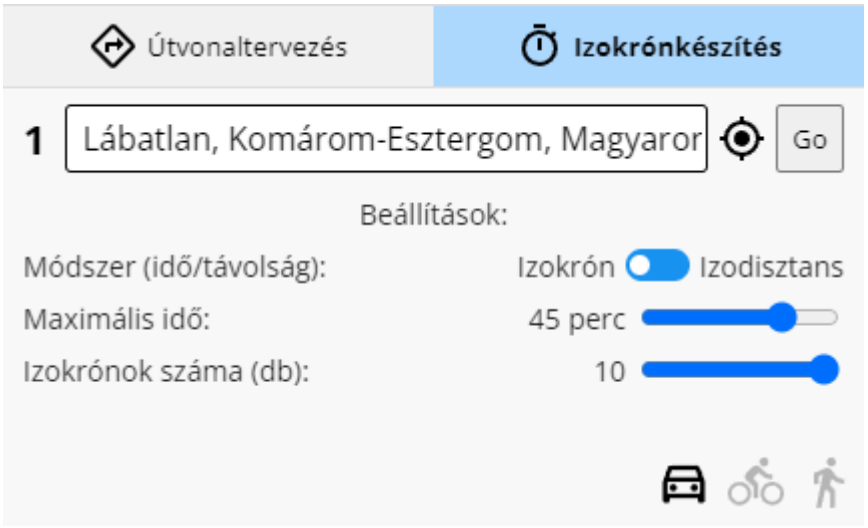
8. ábra: A térképen megjelenő magassági adat (háttértérkép: © OpenStreetMap contributors)



9. ábra: Az útvonalra vonatkozó magassági metszet grafikonja

Az alkalmazás az oldalpanel felső részén kapcsolható át izokrónkészítő-módba. Az izokrón egy megadott középponttól számított, azonos idő alatt elérhető pontokat összekötő vonal – ez a térképen egy ennyi idő alatt elérhető területet lefedő poligonként jelenik meg. Lehetőség van izodisztansokat is generálni, melyek hasonló elven működnek, csupán idő helyett távolság alapon. Mivel útvonaltervező alkalmazásról van szó, mindkét módszer az úthálózatot veszi figyelembe és ezen számol (e feltétel nélkül pl. az izodisztansok légvonalban számított egyszerű, koncentrikus körök lennének a térképen).

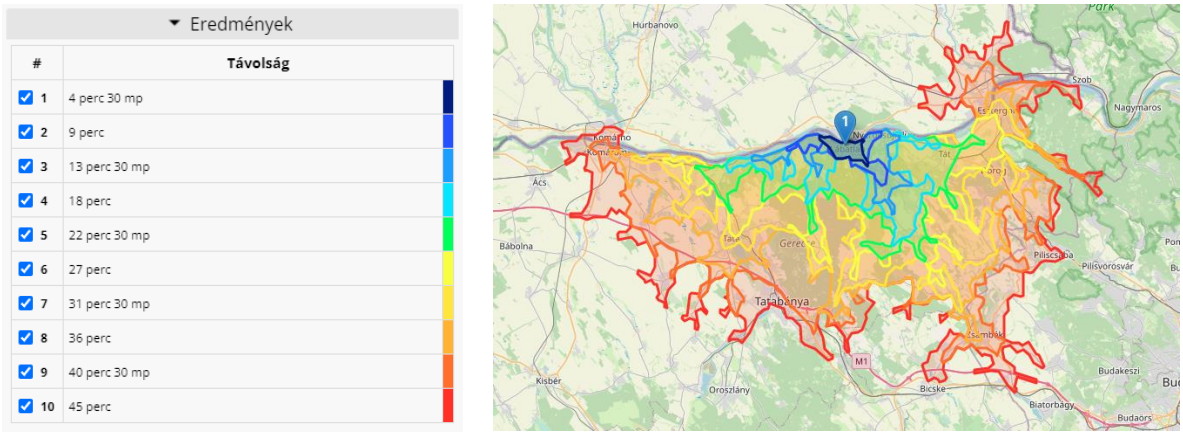
A középpont megadása szintén a pontmegadás korábban említett egyik módszerével történhet. Izokrón készítése esetén megadható a maximális idő, mely a legmesszebb elnyúló izokrón poligon határvonala lesz. Az izokrónok darabszáma megszabja, hogy ezt a maximális időt hány darab, közbeeső poligonra/intervallumra osszuk (pl. ha 30 perces izokrónt szeretnék és 3 a darabszám, a következők jönnek létre a térképen: 10, 20 és 30 perces). Izodisztans generálása esetén a maximális idő helyett maximális távolságot tudunk megszabni. A maximális időt/távolságot az Openrouteservice által megadott alsó és felső limitek közé lehet beállítani, mely az útvonaltervezés profiljától függően változhat (autós, biciklis vagy gyalogos).



10. ábra: Izokrónkészítés paraméterei (ikonok forrása: Google Fonts - Icons)

Eredményként megjelennek a kért, azonos idő alatt vagy távolságban elérhető területek a térképen, valamint ezek jegyzéke az oldalpanelen, táblázatos formában. A poligonok kitöltésük (és határvonaluk) színe alapján azonosíthatóak, valamint az egérkurzort föléjük mozgatva is megjelennek a hozzájuk kapcsolódó idő- vagy távolságértékek. A táblázat első oszlopában található kapcsolókkal az egyes poligonok láthatósága ki- és bekapcsolható. Ez akkor lehet hasznos, ha vizuálisan kevésbé terhelt térképen szeretnénk vizsgálódni, esetleg egy konkrét

izokrón kiterjedése érdekel minket. Az itiner táblázathoz hasonlóan a kurzort egy adott sor felett mozgatva az érintett poligon színt változtat a térképen.



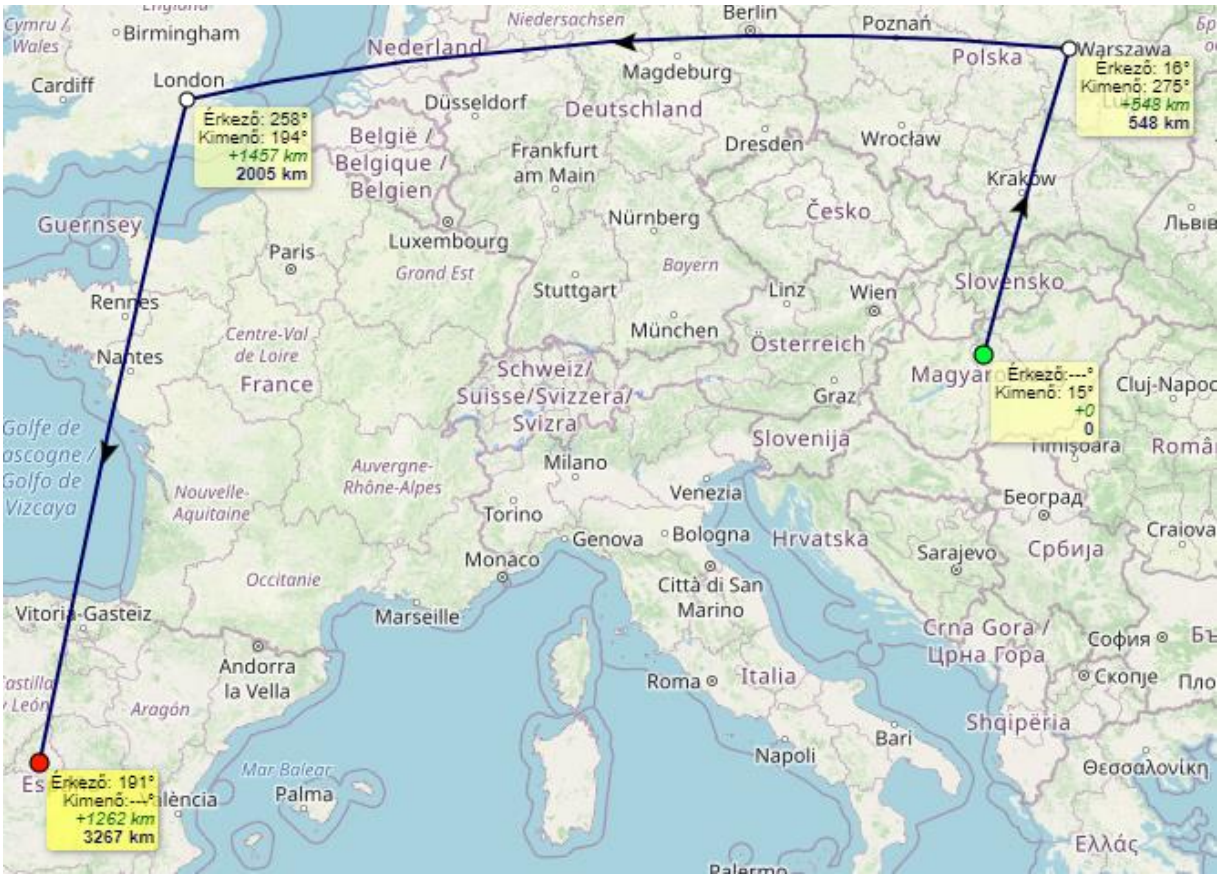
11. ábra: Izokrónkészítés példa: 10 db izokrón, 45 perc maximális idővel, autóval, Lábatlan középponttal (balra: izokrónok jegyzéke, jobbra: izokrónok a térképen) (háttértérkép: © OpenStreetMap contributors)

A térképre jobb egérgombbal kattintva előjön egy menü, melyen egyéb funkciók érhetőek el. Az itt elérhető funkciók: a "Mi található itt?" lehetőség egy felugró üzenetben kiírja, mi található az adott helyen (lásd 12. ábra); a térkép egy helyre való mozdítása, nagyítása és kicsinyítése; valamint az A, B útvonalpont- és izokrónközéppont-lerakó lehetőségek.

12. ábra: "Mi található itt?" (háttértérkép: © OpenStreetMap contributors)

Távolság- és iránymérést a jobb alsó sarokban található ikonokkal lehet indítani (lásd 13. ábra). Ennek bekapcsolása után a térképre kattintva rakhatóak le a pontok, melyek között távolságot és irányt mér (lásd 14. ábra). Két lerakott pont között lehetséges egy új pont beszúrása (CTRL+kattintás), az elhelyezett pontokat lehet mozgatni és törölni (SHIFT+kattintás). A távolságmérő minden pontra megadja az arra vonatkozó érkező és kimenő irányt (azimutot), az előző és a pont közötti szakasz hosszát, valamint a teljes mérés odáig tartó szakaszának hosszát is.

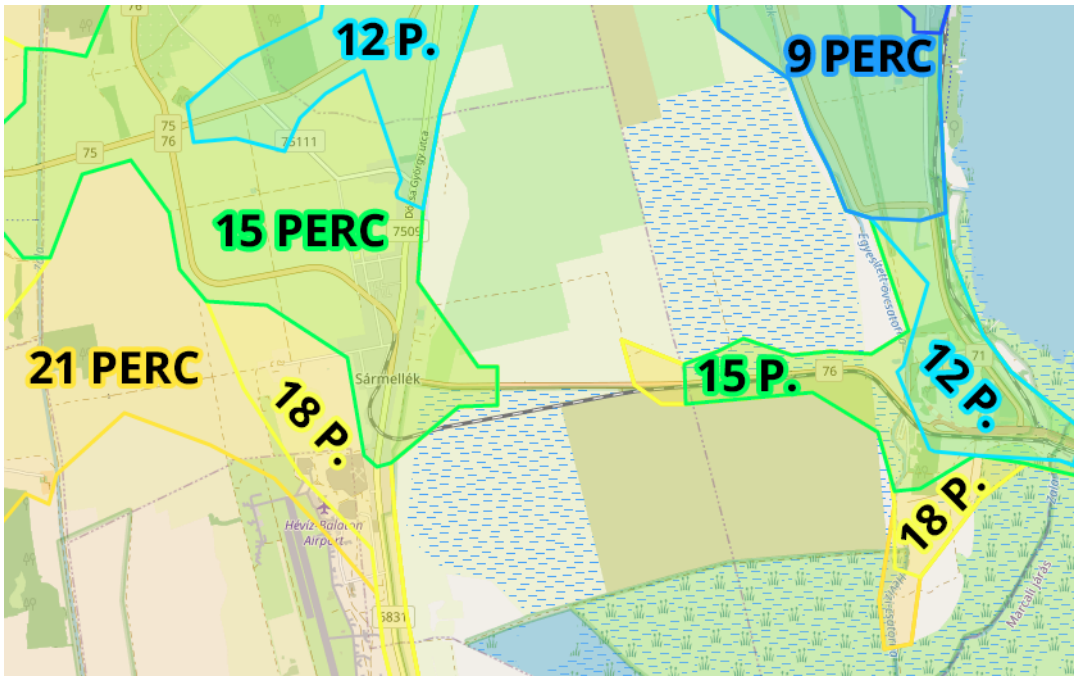
13. ábra: Távolság- és iránymérés funkció



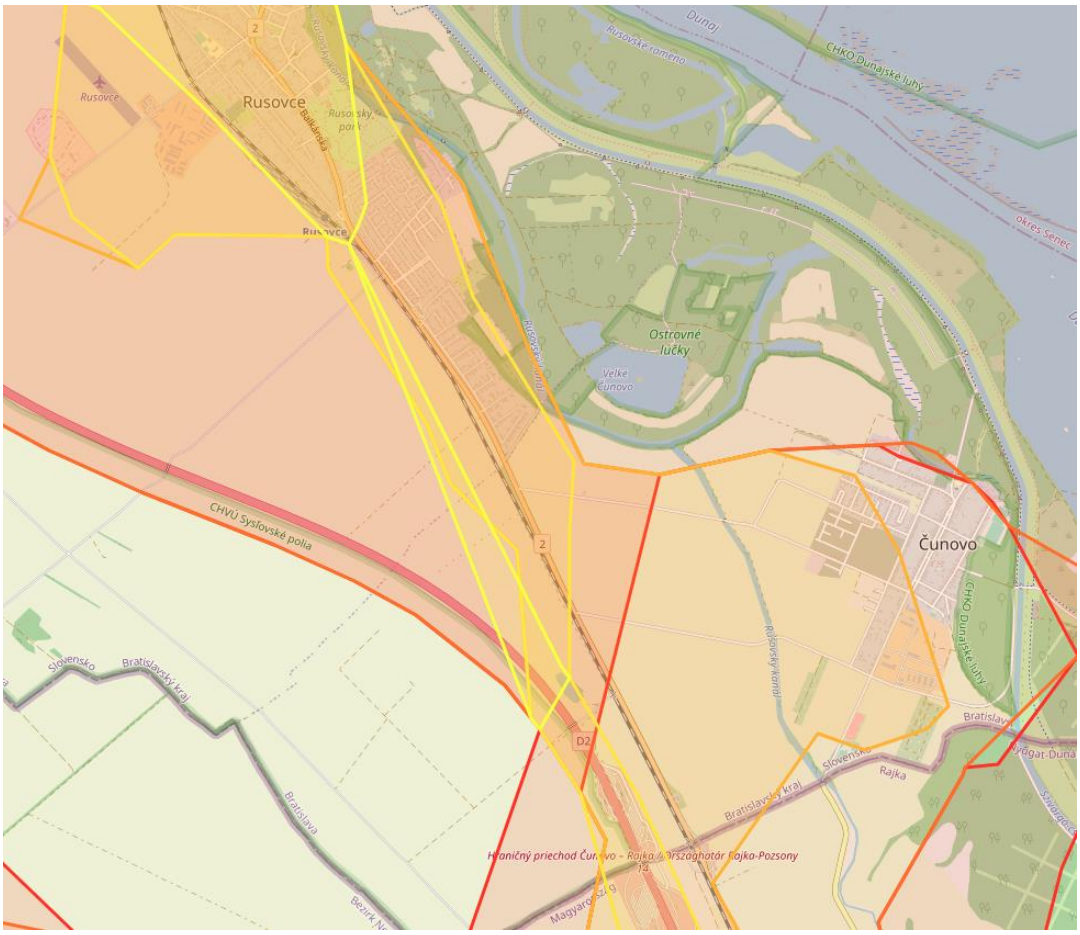
14. ábra: Távolság- és iránymérés: mérés Budapest, Varsó, London és Madrid között (háttértérkép: © OpenStreetMap contributors)

7.1 Lehetséges anomáliák az Openrouteservice által szolgáltatott adatokban

Az izokrónok generálása feltehetően egy bonyolult és nagyobb számítási kapacitást igénylő feladat (vélhetően az OSM rendkívül összetett, rengeteg elemből álló útadatbázisa miatt is), melyet az ORS algoritmusa végez. Egy ilyen méretű adatbázis esetében, ha egy adott középpontú izokrónt kérünk le, az algoritmusnak minden irányban le kell futtatnia az útvonalkereső algoritmusait a háttérben (a lehető legtöbb útkombináción), hogy megfejtse, hol éri el az útvonal az adott időhatárt. Az ORS által generált izokrónok és izodisztans poligonok általában pontosak és geometriailag helyesek, viszont néhány esetben a poligonok helyenként rendszertelenül, hiányosan (ha az adott úton nem futott az algoritmus), vagy egymás határát keresztezve jelenhetnek meg (lásd 15. és 16. ábra).



15. ábra: Keszthely (é.sz. 46.77137°, k.h. 17.24750°) központú, 30 perces, 10 darabos, autós izokrónok: a Sármelléktől kelet felé futó út egy szakaszát nem fedi egyik izokrón sem (feltehetően a sárga 18 percesnek fednie kellene)



16. ábra: Pozsony (é.sz. 48.15142°, k.h. 17.12219°) központú, 30 perces, 10 darabos, autós izokrónok: Oroszvár és Dunacsún között meglehetősen kaotikus izokrónokat kaptunk, melyek határai egymást keresztezik

8. Összefoglalás

Diplomamunkámban egy komplex, webes útvonaltervező alkalmazást fejlesztettem, mely kombinálja az Openrouteservice által nyújtott szolgáltatásokat, demonstrálja azok működését és megjeleníti az általuk generált, főként GeoJSON formátumú térbeli adatokat egy interaktív felületen és a Leaflet térképobjektumon. Az alkalmazás képes: útvonalpontok közötti autós, kerékpáros vagy gyalogos útvonaltervezésre, mely a navigáció egyes lépéseit egy táblázatban mutatja meg; ajánlott, legrövidebb és leggyorsabb útvonaltípus tervezésére; az útvonalra illesztett magassági profil előállítására és bemutatására; izokrónok és izodisztans poligonok készítésére és térképen való megjelenítésére. Kiegészítő funkcióként lehetőséget ad a térképi szakasz-, távolság- és iránymérésre, valamint képes a szöveges mezők geokódolására és fordított geokódolásra is. A fejlesztett webes felület interaktív, és alkalmazkodik a készülék megjelenítőjéhez, így akár mobilos környezetben is jól használható (ez főként a terepen való használatot segíti).

Az eszköz felhasználói felületét főként HTML és CSS nyelveken, a fő programkódot, mely az alkalmazás alapját képezi, pedig JavaScript nyelven írtam. A fejlesztés során elsajátítottam a három, webes tartalmak alapját képező nyelv nyújtotta lehetőségek egy részét, valamint ezek megfelelő összekapcsolását és -hangolását annak érdekében, hogy a felület interaktív és dinamikusan változtatható legyen. A fejlesztés során előkerülő problémákat (pl. az útvonal polyline újraszámítása/egyszerűsítése, koordinátafelismerés, vagy a lekérések állapotának lekezelése) sikeresen megoldottam.

Az alkalmazás működőképes és a tervezett és dokumentált funkciók mindegyikére képes, de ahogy a számítógépes/webes alkalmazások legtöbbje, “teljes” vagy “tökéletes” állapot ritkán érhető el, így ez az alkalmazás is esetlegesen továbbfejleszthető, pl.: funkció az útvonaltervezés során elkerülendő területek berajzolására, több közbeeső útvonalpont támogatása, útvonal menti POI-kigyűjtés, alternatív útvonalak lekérése, GPX/KML import és export lehetősége.

Az Openrouteservice útvonaltervező webalkalmazás a következő webhelyen érhető el:

https://balladaniel.github.io/ors/

Megjegyzés: a hivatkozott webes változat nem feltétlen tükrözi a jelen dolgozat leadásakor (2022. május 15.) fennálló állapotot, továbbfejlesztés esetén eltérhet attól.

9. Hivatkozások

Agafonkin, V. (2022). *Leaflet - a JavaScript library for interactive maps*. Letöltés dátuma: 2022. február 4., forrás: https://leafletjs.com/

Alasdair, C. (2004). *Design Elements of Effective Transit Information Materials: Final Report.* Tampa, Florida: National Center for Transit Research, University of South Florida. Letöltés dátuma: 2022. május 8., forrás: https://rosap.ntl.bts.gov/view/dot/38609

Antrim, A., & Barbeau, S. J. (2013). *The many uses of GTFS data–opening the door to transit and multimodal applications.* Tampa: University of South Florida. Letöltés dátuma: 2022. május 8., forrás: http://www.locationaware.usf.edu/wp-content/uploads/2010/02/The-Many-Uses-of-GTFS-Data-%E2%80%93-ITS-America-submission-abbreviated.pdf

Budapesti Közlekedési Központ. (2022). *BKK OpenData Portal.* Letöltés dátuma: 2022. május 8., forrás: BKK OpenData Portal: https://opendata.bkk.hu/home

Cheung, P., & Sengupta, U. (2016). *Analysis of Journey Planner Apps and Best Practice Features.* Manchester School of Architecture, Centre for Complexity Planning & Urbanism. Manchester: Manchester Metropolitan University. Letöltés dátuma: 2022. május 8., forrás: https://e-space.mmu.ac.uk/618521/

Cordeau, J.-F., Gendreau, M., & Laporte, G. (1997. szeptember). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks, 30*(2), old.: old.: 105-119. doi:10.1002/(SICI)1097-0037(199709)30:2<105::AID-NET5>3.0.CO;2-G

Downie, N., Perkins, D., Timberg, E., Sylvestre, J., Panzarino, Z., Brunel, S., & Linsley, T. (2022). *Chart.js - Simple yet flexible JavaScript charting for designers & developers*. Letöltés dátuma: 2022. február 9., forrás: https://www.chartjs.org/

Duckham, M., Kulik, L., Worboys, M. F., & Galton, A. (2008. október). Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Pattern Recognition, 41*(10), old.: old.: 3224-3236. doi:10.1016/j.patcog.2008.03.023

Flanagan, D. (2011). *JavaScript: the definitive guide.* Peking; Farnham: O'Reilly.

Gede, M. (2022). *HTML összefoglaló*. Letöltés dátuma: 2022. március 31., forrás: http://mercator.elte.hu/~saman/hu/okt/html/

Gede, M. (2022). *Script nyelvek alkalmazása a web-kartográfiában*. Letöltés dátuma: 2022. február 9., forrás: http://mercator.elte.hu/~saman/hu/okt/script_jegyzet.html

Gravois, J. (2017. augusztus 24.). *Esri World Imagery in OpenStreetMap!* (ESRI) Letöltés dátuma: 2022. március 2., forrás: ESRI ArcGIS Blog: https://www.esri.com/arcgis-blog/products/constituent-engagement/constituent-engagement/esri-world-imagery-in-openstreetmap/

GTFS Realtime. (2022). Letöltés dátuma: 2022. május 8., forrás: Google Transit APIs: https://developers.google.com/transit/gtfs-realtime/

McHugh, B. (2013). Pioneering Open Data Standards: The GTFS Story. In B. Goldstein, L. Dyson, & A. Nemani, *Beyond Transparency: Open Data and the Future of Civic Innovation* (old.: 125-135). San Francisco, Kalifornia: Code for America. Letöltés dátuma: 2022. május 8., forrás: https://beyondtransparency.org/chapters/part-2/pioneering-open-data-standards-the-gtfs-story/

Meet the Team - ORS. (2022). Letöltés dátuma: 2022. március 9., forrás: Openrouteservice: https://web.archive.org/web/20220309124312/https://openrouteservice.org/team/

Mooney, P., & Corcoran, P. (2012). Characteristics of Heavily Edited Objects in OpenStreetMap. *Future Internet*, old.: 285-305. doi:10.3390/fi4010285

Neis, P., & Zipf, A. (2012). Analyzing the Contributor Activity of a Volunteered Geographic Information Project — The Case of OpenStreetMap. *ISPRS International Journal of Geo-Information, 1*(2), old.: 146-165. doi:10.3390/ijgi1020146

Neis, P., Zielstra, D., & Zipf, A. (2012). The Street Network Evolution of Crowdsourced Maps: OpenStreetMap in Germany 2007–2011. *Future Internet, 4*(1), old.: 1-21. doi:10.3390/fi4010001

OpenMobilityData. (2022). Letöltés dátuma: 2022. május 8., forrás: OpenMobilityData - Public transit feeds from around the world: https://openmobilitydata.org/

Openrouteservice.org. (2022). *Openrouteservice API interactive examples (API V2 Documentation)*. (The Heidelberg Institute for Geoinformation Technology) Letöltés dátuma: 2022. február 3., forrás: https://openrouteservice.org/dev/#/api-docs

OpenStreetMap. (2022). *About OpenStreetMap*. Letöltés dátuma: 2022. március 2., forrás: OpenStreetMap.org: https://www.openstreetmap.org/about

OpenStreetMap stats. (2022. március 1.). Letöltés dátuma: 2022. március 2., forrás: OpenStreetMap: https://web.archive.org/web/20220302203557/https://planet.openstreetmap.org/statistics/data_stats.html

OpenStreetMap wiki. (2022). Letöltés dátuma: 2022. március 3., forrás: OpenStreetMap: https://wiki.openstreetmap.org/wiki/

OpenStreetMap wiki: Openrouteservice. (2022). Letöltés dátuma: 2022. március 9., forrás: OpenStreetMap: https://web.archive.org/web/20220309123431/https://wiki.openstreetmap.org/wiki/Openrouteservice

ORS API Restrictions. (2022). Letöltés dátuma: 2022. április 4., forrás: Openrouteservice: https://openrouteservice.org/restrictions/

ORS Data sources. (2022). (The Heidelberg Institute for Geoinformation Technology) Letöltés dátuma: 2022. március 3., forrás: Openrouteservice documentation: https://web.archive.org/web/20220302230533/https://giscience.github.io/openrouteservice/Data.html

Regular expressions. (2022). Letöltés dátuma: 2022. április 4., forrás: MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

Rothkrantz, L. (2018). Hybrid Dynamic Route Planners. *ACM 19-th International Conference on Computer Systems and Technologies (CompSysTech'18).* Rusze, Bulgária. doi:10.1145/3274005.3274012

Turf.js Docs. (2022). Letöltés dátuma: 2022. április 12., forrás: Turf.js | Advanced geospatial analysis: https://turfjs.org/docs/#along

VROOM-Project Benchmarks. (2022). Letöltés dátuma: 2022. március 16., forrás: VROOM GitHub.com: https://github.com/VROOM-Project/vroom/wiki/Benchmarks

W3C. (2018. november 18.). *CSS Flexible Box Layout Module Level 1 (W3C Candidate Recommendation)*, 20181119. (World Wide Web Consortium (W3C)) Letöltés dátuma: 2022. április 21., forrás: World Wide Web Consortium (W3C): https://www.w3.org/TR/css-flexbox/

W3Schools. (2022). *CSS @media Rule*. Letöltés dátuma: 2022. április 21., forrás: W3Schools: https://www.w3schools.com/cssref/css3_pr_mediaquery.asp

W3Techs. (2022). *Usage statistics of JavaScript as client-side programming language on websites*. Letöltés dátuma: 2022. március 31., forrás: W3Techs.com: https://w3techs.com/technologies/details/cp-javascript/

Zielstra, D., & H. Hochmair, H. (2012). Comparing Shortest Paths Lengths of Free and Proprietary Data for Effective Pedestrian. *Transportation Research Record, 2299*, old.: 41-47. Letöltés dátuma: 2022. március 2., forrás: https://web.archive.org/web/20121215023804/http://flrec.ifas.ufl.edu/hochmair/pubs/TRR_zielstra_hochmair_2012_finaldraft.pdf

Zipper, D. (2017. április 9.). *Who Owns Transit Data?* Letöltés dátuma: 2022. május 8., forrás: Bloomberg: https://www.bloomberg.com/news/articles/2017-04-09/the-case-for-open-transit-data

10. Ábrajegyzék

11. Köszönetnyilvánítás

Köszönettel tartozom témavezetőmnek, Gede Mátyásnak, akinek a konzultációk során adott elméleti és gyakorlati iránymutatása és szakértelme nagy segítség volt a dolgozatom elkészítésében. Köszönöm, hogy elvállalta az általam felhozott téma vezetését, és türelmesen segített annak kidolgozásában és megfelelő irányba terelésében.

12. Nyilatkozat

DIPLOMAMUNKA LEADÁSI
és
EREDETISÉG NYILATKOZAT

Alulírott **Balla Dániel**, Neptun-kód: **G3WSRX**

az Eötvös Loránd Tudományegyetem Informatikai Karának, Térképtudományi és Geoinformatikai Intézetében

Az openrouteservice.org lehetőségeinek bemutatása
egy komplex webes útvonaltervező alkalmazással

című diplomamunkámat a mai napon leadtam.

Témavezetőm neve: Dr. Gede Mátyás

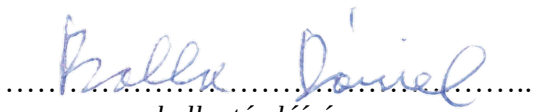
Büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom, hogy jelen diplomamunkám saját, önálló szellemi termékem; az abban hivatkozott szakirodalom felhasználása a szerzői jogok általános szabályainak megfelelően történt.
Tudomásul veszem, hogy szakdolgozat/diplomamunka esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

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

HOZZÁJÁRULOK NEM JÁRULOK HOZZÁ

Budapest, 2022. május 15.



……..
hallgató aláírása