



Rendszerterv

**Plan a trip! Web alapú útvonal tervező
szolgáltatás**

Készítették:

Kovács Gergely

Gángoly Ákos

Tartalomjegyzék

[A rendszer általános ismertetése](#)

[Feladatkiírás](#)

[Funkciók](#)

[Admin](#)

[Felhasználó](#)

[Futtatási környezet](#)

[Megvalósítás](#)

[Architektúra összefoglaló](#)

[Model / adatbázis réteg](#)

[View / business logic réteg](#)

[Template / GUI réteg](#)

[Adat- és adatbázis-terv](#)

[ER diagram](#)

[Különböző entitások](#)

[Vonal entitás](#)

[Állomás entitás](#)

[Állomásrend entitás](#)

[Járat entitás](#)

[Menetrend bejegyzés entitás](#)

[Késés entitás](#)

[GUI terv](#)

[Admin interface terv](#)

[Telepítési leírás](#)

A rendszer általános ismertetése

Ebben a fejezetben bemutatjuk a program funkcionalitását, hogy milyen kontextusban illetve környezetben lehet használni.

Feladatkiírás

Kifejlesztendő egy webes rendszer, amellyel az adminisztrátorok bejelentkezés után vonat- vagy buszjáratokat, vonalakat, állomásokat, menetrendeket és viteldíjakat vihetnek fel. A vonal állomásokat köt össze bizonyos sorrendben. Egy busz- vagy vonatjárat egy vonalon közlekedik. A menetrend azt írja le, mikor melyik állomásról indul egy járat. A menetrendben meg lehet különböztetni hétfégi/ünnepnapos menetrendet is. A viteldíjak különbözőek lehetnek ugyanazon a vonalon két állomás között buszra és vonatra. Az adminisztrátorok az aktuális napi várható késéseket is felvehetik.

A felhasználók az utazás napja, a kiinduló és végállomás megadásával kereshetnek javasolt útiterveket, akár több átszállással is, a késéseket is figyelembe véve. A keresést optimalizálhatjuk útiköltségre, hossza vagy időre.

Funkciók

Admin

Az admin felhasználónév és jelszóval történő azonosítás után tud belépni és elérni a megfelelő funkciókat.

Az adminnak lehetősége van:

- Felvinni a rendszerbe új állomást a neve megadásával.
- Felvinni a rendszerbe új vonalat a következő adatok megadásával:
 - a vonal neve,
 - típusa(vonat/busz)
 - a hozzá tartozó állomások és sorrendjük.
- Felvinni a rendszerbe új járatot a következő adatok megadásával:
 - a vonal, melyhez a járat tartozik,
 - érvényesség kezdetének napja,
 - érvényesség végének napja,
 - típusa(normál/hétfégi/szünnap),

- költsége(a távolságot ezzel a számmal szorozzuk meg - km mértékegység esetén km díjként fogható fel).
- Felvinni a rendszerbe adott menetrend bejegyzéshez új késést a következő adatok megadásával:
 - a késés napja,
 - a késés mértéke.
- Módosítani az adott járáshoz és állomáshoz tartozó menetrend bejegyzést az időpont módosításával.
- Módosítani adott járat következő adatait:
 - a vonal, melyhez a járat tartozik,
 - érvényesség kezdetének napja,
 - érvényesség végének napja,
 - típusa(normál/hétvégi/szünnap),
 - költsége(két állomás között értendő).
- Módosítani adott állomás nevét.
- Módosítani adott vonal nevét és típusát(busz/vonat), illetve a hozzá tartozó állomásokat és sorrendjüket.
- Módosítani egy adott menetrend bejegyzéshez tartozó létező késés napját és mértékét.

Felhasználó

A felhasználó bárki lehet, aki az oldalunkat meglátogatja. A felhasználónak lehetősége van:

- Útvonalak tervezésére az alábbi adatokat megadni és egy útvonalat tervezni:
 - kezdőállomás,
 - végállomás,
 - az indulás pontos időpontja.
- A tervezés optimalizálását az alábbi szempontok szerint kérheti:
 - leutazott távolság,
 - utazási idő,
 - utazás összköltsége.
- A "Plan!" gombra kattintás után blokkok jelennek meg, melyek tartalmazzák:
 - hogy milyen vonalakon,
 - melyik megállótól,
 - melyik megállóig,
 - milyen időpontokban,
 - mekkora költségen tud utazni.
- Az egyes blokkokra kattintva megjelenik a részletes menetrend, amely tartalmazza azokat a megállókat is, amelyeken csak át kell utazni.
- A blokkok alatt összefoglaló szöveg, hogy az utazás
 - mennyi ideig fog tartani,

- mekkora távot fog megtenni,
 - mennyibe fog kerülni.
- A blokkokat tartalmazó oldal URL-je könnyen olvasható, nem tartalmaz adatbázis kulcsokat, a dátum szövegesen és nem másodpercekben jelenik meg, stb. Az alábbi módon épül fel (a <> belül az egyes argumentum leírása található):
 - <Tervezési mód>/<év>.<hónap(szóvegesen)>.<nap>/<óra>:<perc>/<kezdőállomás neve>-<végállomás neve>
 - pl.: time/2019.November.26/15:44/Budapest-Miskolc

Futtatási környezet

A szoftver bármilyen szerveren futtatható, amely támogatja a Python nyelvet és a hozzá tartozó django package-t.

A kliens oldalon az alábbi böngészők támogatottak:

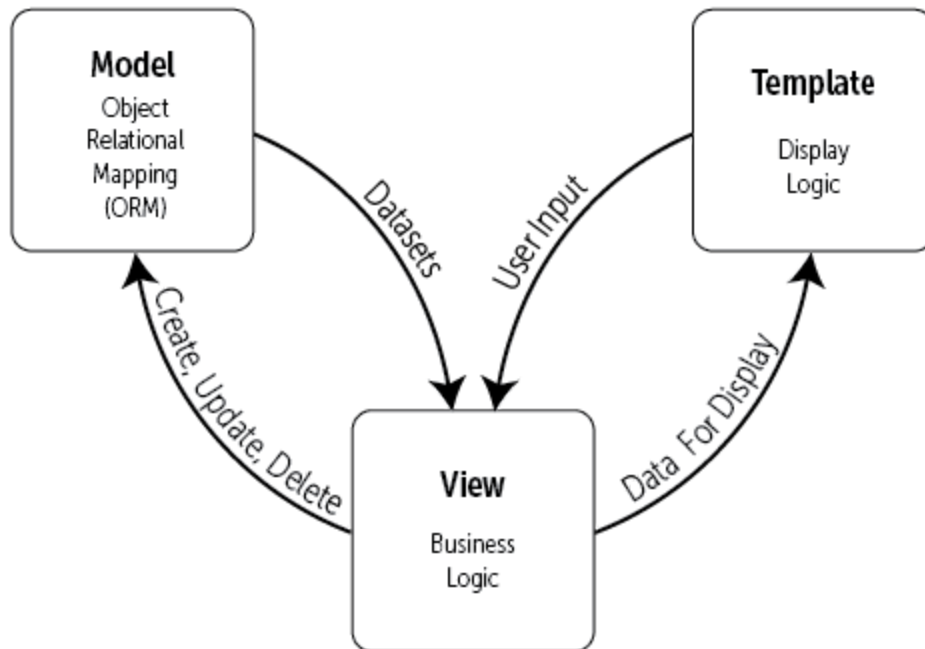
- Chrome 35+
- Firefox 31+
- Safari 9+
- Opera
- Edge
- IE 11+

Megvalósítás

Architektúra összefoglaló

Rendszerünk tervezett architektúrája klasszikus háromrétegű: adatelérési, üzleti logikai és prezentációs rétegből áll. A django keretrendszer alapvetően a Model-View-Template (MVT) architektúrát támogatja:

- Model (M):
 - adatokat tárolja
- View (V):
 - Adatokat hív le a Model-ből, azokat formázza, majd HTTPS protokollon keresztül elküldi a kliensnek.
- Template (T):
 - A View-től kapott adatokat prezentálja, a felhasználó inputot elküldi a szervernek.



Model / adatbázis réteg

Az adatbázis réteg feladata az adatok perzisztálása. Tábláit Python nyelven definiáljuk. Minden a `django.db.models.Model` osztályból származó osztályhoz tartozik egy tábla az adatbázisban is. Emellett még keletkeznek további look-up táblák is, ha az entitások között olyan relációkat alkalmazunk, amik ezt használják(pl.: ManyToMany, de, hogy pontosan milyen táblák generálódnak csak akkor kell foglalkoznunk, ha saját SQL lekérdezést írunk, de erre szinte soha sincs szükség.) A `manage.py makemigrations` parancs futtatásával a django legenerálja az adatbázis létrehozásához szükséges sql parancsokat. A `manage.py migrate` paranccsal frissíthetjük az adatbázis sémát. További információkat találhat a folyamattal kapcsolatban a [django hivatalos dokumentációjában](#).

View / business logic réteg

Az üzleti logikát szintén Python nyelven definiáljuk. Az adatok eléréséhez használhatjuk az előző alfejezetben definiált `django.db.models.Model` leszármazottjaiként definiált osztályokat. Mivel a django magában foglalja az adatok lekérdezésének logikáját nincs szükség saját adatelérési osztályok definiálására. Például, ha le akarjuk kérdezni az összes állomást, akkor elég csak "`Service.objects.all()`" parancsot hívunk és visszakapunk egy listát ami Station objektumokat tartalmaz. Ha használjuk az egyes objektumok property-eit akkor lazy módon a django elvégzi az adatbázis lekérdezéseket és cache-eli az

eredményeket. További információkat találhat az [adatok lekérdezéséről](#), valamint a [lekérdezések eredményeinek cacheléséről](#) a django hivatalos dokumentációjában.

Ha megvannak az adatok, azokat ki szeretnénk jelezni felületünkön. Ezt úgy tesszük meg, hogy a `views.py` és az `admin.py` fájljainkban nézeteket hozunk létre, majd ezekben a nézetekben a `django.shortcuts.render` függvény segítségével egy olyan HTTP válasszal térünk vissza, mely a törzsében tartalmazza az adatokat. Ebben a válaszban megadjuk a válaszfelület template-jét. Ebben a template-ben lesz megtalálható a konkrét felület leírása HTML nyelven.

Egyszerű módon tudjuk kezelni a 404-es hibákat is a `django.shortcuts.get_object_or_404` függvénnyel. Ezen kívül az `urls.py` fájlban meg kell adnunk, melyik linkhez melyik nézetek fognak tartozni.

Az admin nézetekhez segítséget kapunk abból a szempontból, hogy vannak alap admin nézet osztályok (<https://docs.djangoproject.com/en/2.2/ref/contrib/admin/#modeladmin-objects>), melyeket továbbfejlesztve tudjuk elkészíteni saját admin nézeteinket. Ezekhez az admin nézet osztályokhoz alap template-ek is implementálva vannak, amik ugyancsak jó kiindulási pontot ad. Illetve a linkek és nézetek párosítása is meg van oldva automatikusan.

Itt van megvalósítva útvonalkereső algoritmusunk is, melyet a [Dijkstra-algoritmus](#) alapján készítettünk el. Az algoritmus az adott állomáson megnézi az őt érintő összes vonalból a lehetséges következő állomásokat, majd ezeknek megfelelő bejegyzést készít a keresési feltételnek megfelelően. A mohó Dijkstra algoritmus megtalálja így az adott feltétel alapján a legrövidebb utat a kezdő és végállomás között, és visszaadja az ahhoz szükséges vonalakat és állomásokat. Az algoritmus futásideje: $O(|V|^2)$, ahol V az állomások halmaza.

A Dijkstra algoritmusnak egy általános formája van megvalósítva a Dijkstra osztályban. A konstruktorában meg lehet adni egy `getweight` függvényt, ami egy `WeightArgs` struktúrát vár. A struktúrákban szereplő adatokból számolja, hogy az adott él mekkora súlyú lesz. Ezt lehet módosítani, ha költségre, időre vagy távolságra optimalizálunk. Ez a `Component Configurator` mintának egy karcsúsított változata. A Dijkstra egy `RouteInfo` objektummal tér vissza. Ez egy gráf struktúrát tartalmaz, az egyes csomópontokat a `RouteInfoStation` osztály írja le. A `planning_alg.plan()` függvény alakít át táblázatos formába. Ez a táblázat kerül aztán a `view.py`-ba ahol átadásra kerül a template-nek.

tripplanner.bi.dijkstra.Dijkstra

__init__(self, get_weight: Callable[[WeightArgs], float])

__call__(self, start_station: int, start_time: datetime.datetime)

__get_stations_with_min_dist(stations: List[int], dist: Dict[int, float])

__get_neighbors(self, u: int, t: datetime.datetime)

Dijkstra.NeighbourResult

get_weight

tripplanner.bi.dijkstra.RouteStationInfo

prev

line_prev

time_leave_prev

time_arrive

fee

distance

tripplanner.bi.dijkstra.RouteInfo

__init__(self, start_station, time_arrive)

update(self, u: int, res: 'Dijkstra.NeighbourResult')

__getitem__(self, item)

__station_infos

tripplanner.bi.dijkstra.WeightArgs

distance

fee

v_arrive_time

t

Powered by y-files

Template / GUI réteg

A GUI-t a template rétegben implementáljuk. A view rétegben adott viewhoz lehet csatolni templatet, ami a view megjelenését határozza meg. Ez HTML5, CSS illetve JS nyelven íródik a django template nyelvvel kiegészítve(lásd bővebben itt:

<https://docs.djangoproject.com/en/2.2/ref/templates/language/>).

A felhasználói felületünk stílusát a materialize (<https://materializecss.com/>) stíluslapjai segítségével alakítottuk ki.

Adat- és adatbázisterv

Adatbázisunk típusát és nevét a settings.py Python fájl módosításával állíthatjuk(bővebb információk itt: <https://docs.djangoproject.com/en/2.2/intro/tutorial02/#database-setup>).

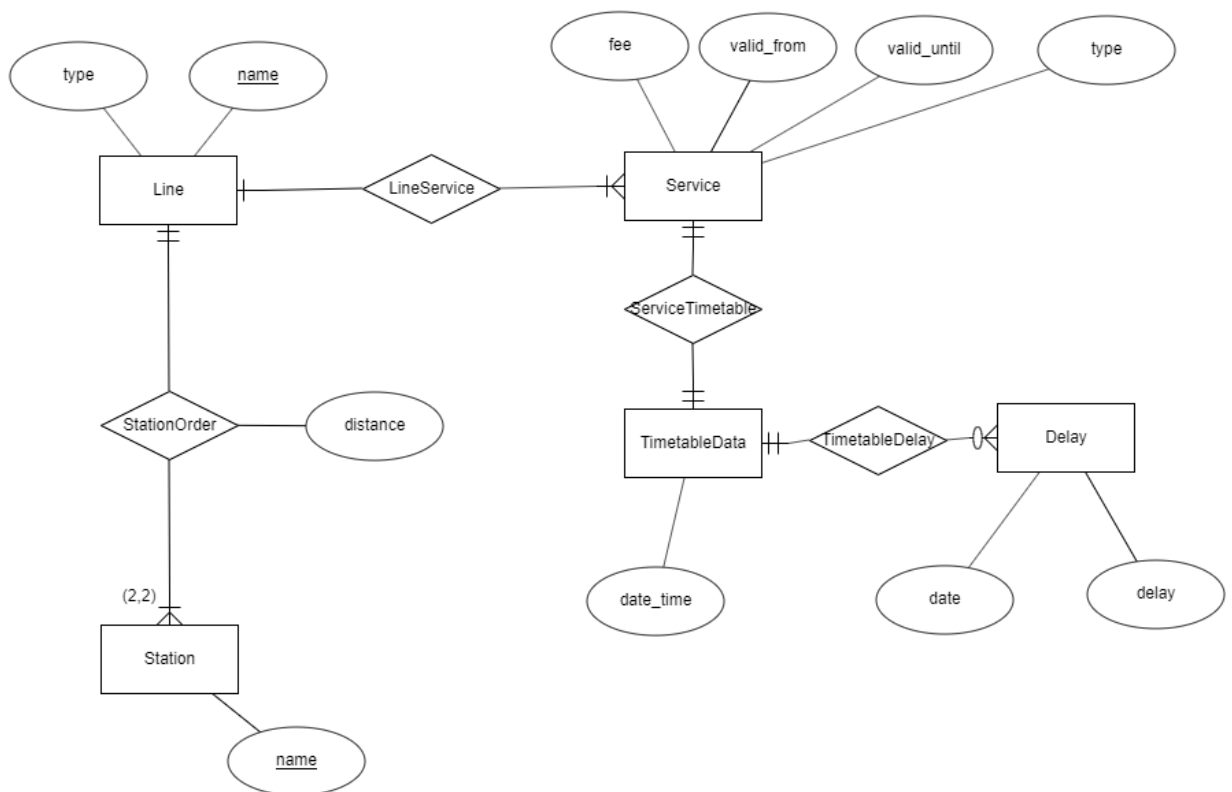
Nekünk SQLite3-ra esett a választásunk, mivel ez az alapértelmezett és kis méretű és

egyszerű, tesztjellegű alkalmazásunk miatt elég számunkra az SQLite és egyszerűbb a használata mint a többi opciónak.

Az adatbázist ezután a “Model / adatbázis réteg” fejezetben leírtaknak megfelelően a django a `manage.py makemigrations` és a `manage.py migrate` parancsok segítségével tudjuk létrehozni, majd később módosítani modellünk alapján. Mielőtt létrehoznánk, a `manage.py sqlmigrate` paranccsal ellenőrizhetjük, hogy az adott migration milyen sql parancsoknak megfelelő változásokat fog végrehajtani az adatbázison.

Adatbázistervünk alább látható:

ER diagram



Különböző entitások

Vonal entitás

Célja

Egy vonal adatainak reprezentálása.

Leképzése a kódban

A `django.db.models.Model` osztályból származó `Line` osztály.

Tulajdonságai

| Python mezőnév | Python adattípus |
|-------------------|---|
| <code>name</code> | <code>django.db.models.CharField</code> |
| <code>type</code> | <code>django.db.models.CharField</code> |

- A **name** mező tárolja a vonal nevét, maximum 30 karakter hosszú, unique.
- A **type** mező tárolja a vonal típusát(busz/vonat), maximum 1 karakter hosszú, értékét a BUS konstansnak megfelelő 'B' és a TRAIN konstansnak megfelelő 'T' karakterek közül választhatja, defaultja a 'T'.

Állomás entitás

Célja

Egy állomás adatainak reprezentálása.

Leképzése a kódban

A `django.db.models.Model` osztályból származó `Station` osztály.

Tulajdonságai

| Python mezőnév | Python adattípus |
|-------------------|---|
| <code>name</code> | <code>django.db.models.CharField</code> |

- A **name** mező tárolja az állomás nevét, maximum 30 karakter hosszú, unique.

Állomásrend entitás

Célja

Egy adott vonalhoz tartozó két egymást követő állomás összekötése, mellyel a vonalhoz tartozó állomások sorrendjét szándékozunk megadni.

Leképzése a kódban

A `django.db.models.Model` osztályból származó `StationOrder` osztály.

Tulajdonságai

| Python mezőnév | Python adattípus |
|---------------------------|--|
| <code>station_from</code> | <code>django.db.models.ForeignKey</code> |
| <code>station_to</code> | <code>django.db.models.ForeignKey</code> |
| <code>line</code> | <code>django.db.models.ForeignKey</code> |
| <code>distance</code> | <code>django.db.models.IntegerField</code> |

- A **`station_from`** mező tárolja a kapcsolatban előbb következő állomás idegen kulcsát. Az említett állomás törlésekor ez a bejegyzés is törlődik.
- A **`station_to`** mező tárolja a kapcsolatban a `station_to` után következő állomás idegen kulcsát. Az említett állomás törlésekor ez a bejegyzés is törlődik.
- A **`line`** mező tárolja a bejegyzéshez tartozó vonal idegen kulcsát. Az említett vonal törlésekor ez a bejegyzés is törlődik.
- A **`distance`** mező tárolja a bejegyzéshez tartozó két állomás közötti távolság értékét. Alapértelmezetten 0 ez az érték, lehet null.

Járat entitás

Célja

Egy járat adatainak reprezentálása.

Leképzése a kódban

A `django.db.models.Model` osztályból származó `Service` osztály.

Tulajdonságai

| Python mezőnév | Python adattípus |
|----------------|-------------------------------|
| fee | django.db.models.IntegerField |
| line | django.db.models.ForeignKey |
| valid_from | django.db.models.DateField |
| valid_until | django.db.models.DateField |
| type | django.db.models.CharField |

- A **fee** mező a járat két állomása közötti utazási költséget tárolja, default értéke 1.
- A **line** mező a járhoz tartozó vonal idegen kulcsát tárolja. Az említett vonal törlésekor ez a bejegyzés is törlődik.
- A **valid_from** mező a járat érvényességének kezdeti napját tárolja, alapértelmezett értéke a bejegyzés létrehozásának napja.
- A **valid_until** mező a járat érvényességének végső napját tárolja, alapértelmezett értéke a bejegyzés létrehozásának napja+1 év.
- A **type** mező tárolja a járat típusát(normál/hétvégi/szünnap), maximum 1 karakter hosszú, értékét a NORMAL konstansnak megfelelő 'N', a WEEKEND konstansnak megfelelő 'W' és a HOLIDAY konstansnak megfelelő 'H' karakterek közül választhatja, defaultja az 'N'.

Menetrend bejegyzés entitás

Célja

Egy adott járhoz tartozó menetrend egy (adott állomáshoz tartozó) bejegyzésének adatainak tárolása.

Leképzése a kódban

A django.db.models.Model osztályból származó TimetableData osztály.

Tulajdonságai

| Python mezőnév | Python adattípus |
|----------------|------------------|
|----------------|------------------|

| | |
|-----------|-----------------------------|
| service | django.db.models.ForeignKey |
| station | django.db.models.ForeignKey |
| date_time | django.db.models.TimeField |

- A **service** mező a menetrend bejegyzéshez tartozó járat idegen kulcsát tárolja. Az említett járat törlésekor ez a bejegyzés is törlődik.
- A **station** mező a menetrend bejegyzéshez tartozó állomás idegen kulcsát tárolja. Az említett állomás törlésekor ez a bejegyzés is törlődik.
- A **date_time** mező a bejegyzéshez tartozó időpontot tárolja(a dátumot nem!). Ez jelenti, hogy az adott járat a menetrend szerint mikor lesz az adott állomásán. Default értéke a bejegyzés létrehozásának pillanata.

Késés entitás

Célja

Egy adott menetrend bejegyzéshez tartozó késés adatainak tárolása.

Leképzése a kódban

A django.db.models.Model osztályból származó Delay osztály.

Tulajdonságai

| Python mezőnév | Python adattípus |
|----------------|--------------------------------|
| timetable | django.db.models.ForeignKey |
| delay | django.db.models.DurationField |
| date | django.db.models.DateField |

- A **timetabledata** mező a késéshez tartozó menetrend bejegyzés idegen kulcsát tárolja. Az említett menetrend bejegyzés törlésekor ez a bejegyzés is törlődik.
- A **delay** mező a késés mértékét tárolja, alapértelmezett értéke 0 hét 0 nap 0 óra 0 perc 0 másodperc 0 miliszekundum 0 mikroszekundum (<https://docs.python.org/3/library/datetime.html#datetime.timedelta>).

- A **date** mező a késés napját tárolja, alapértelmezett értéke a bejegyzés létrehozásának napja.

GUI terv

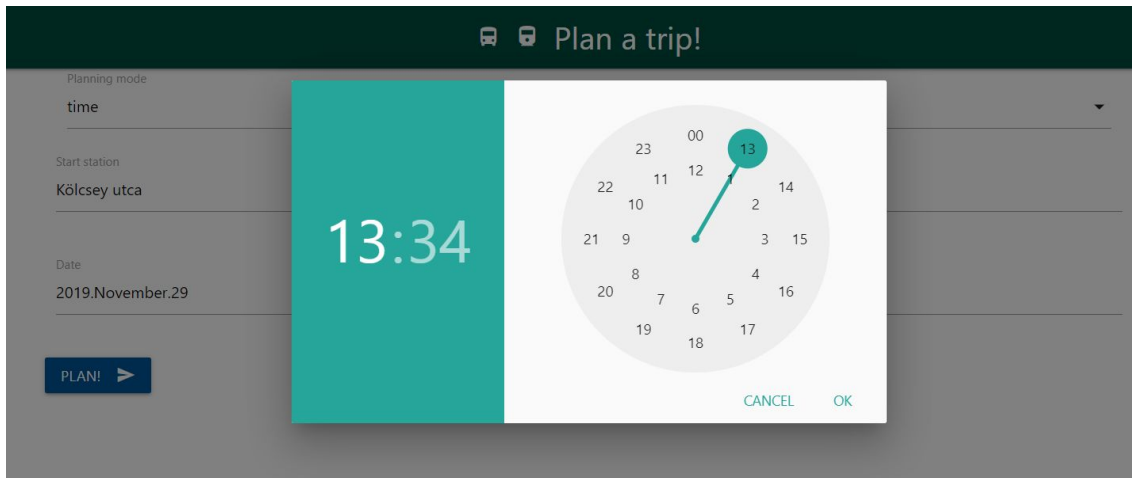
A felhasználói felület tervezésénél minimalista és letisztult design-t szerettünk volna, ami intuitív is. Alapvetően két felülete van a felhasználói felületnek: a tervezési felület és az eredmény felület.

A tervezési felület a következő designnal rendelkezik:

The screenshot shows a web form titled "Plan a trip!" with a dark green header. Below the header, there's a "Planning mode" dropdown menu set to "time". The form has two main sections: "Start station" and "Destination station". The "Start station" field contains "Kölcsey utca" and the "Destination station" field contains "Nagyvér". Below these, there are "Date" and "Time" fields. The "Date" field contains "2019.November.29" and the "Time" field contains "13:34". At the bottom left, there is a blue button labeled "PLAN!" with a right-pointing arrow.

A **planning mode** dropdown-ból a három tervezési mód(time/cost/distance) közül választhatunk. A **start station** és **destination station** mezőkbe beírhatjuk a tervezett útvonal kezdő-, és végállomását. Miután elkezdjük begépelni, felajánlja a létező állomásokat amiknek substringja a beírt szöveg. A tervezett útvonal kezdetének dátumát és időpontját a **date** és **time** mezőkben egy dátum-, és időpont pickerben tudjuk megadni:

This screenshot shows the same "Plan a trip!" form, but with a date and time picker overlay. The "Date" field is highlighted, and a calendar is displayed. The calendar shows the month of November 2019. The date "Fri, Nov 29" is highlighted in a teal box. The "Time" field is also highlighted, and a time picker is displayed. The time picker shows the time "13:34" in a teal box. The "PLAN!" button is still visible at the bottom left.



Az eredményfelület pedig a következő designnal rendelkezik:

| Plan a trip! | | | |
|------------------------|--------------|--|------------------------------|
| 13:55 2019.November.29 | Kölcsey utca | | 257 |
| 14:03 2019.November.29 | Turul utca | | Fee: 16.0€ |
| | | | Travel time: 0 days 00:08:00 |
| 14:06 2019.November.29 | Turul utca | | 63 |
| 14:09 2019.November.29 | Nagyret | | Fee: 2.0€ |
| | | | Travel time: 0 days 00:03:00 |



Total travel time: 0 days 00:14:00

Total cost: 18.0€

Az útvonal különböző vonalon megtett részei külön blokkokban jelennek meg. Minden blokkban a részútvonal kezdő-, és végállomásának neve, illetve az útvonalon hozzájuk tartozó dátum és időpont látszik bal oldalon. Jobb oldalon a busz-, vagy vonatikkal jelzi a felület a vonal típusát. Alatta az adott vonal neve látható, illetve a részútvonal összköltsége, össztávolsága és összideje.


A blokkok alatt pedig az egész útvonal összköltsége, össztávolsága és összidőtartama látható nagyobb betűmérettel.

Az egyes blokkokat lenyitva azok részletei jelennek meg:



Plan a trip!

13:55 2019.November.29
Kölcsey utca

14:03 2019.November.29
Turul utca


257

Fee: 16.0€

Travel time: 0 days 00:08:00

13:55 2019.November.29
Kölcsey utca

13:57 2019.November.29
Községház utca


13:59 2019.November.29
Solymári elágazás

14:01 2019.November.29
Csatlós utca

14:03 2019.November.29
Turul utca

14:06 2019.November.29
Turul utca

14:09 2019.November.29
Nagyvér


63

Fee: 2.0€

Travel time: 0 days 00:03:00

Zölddel a kezdő-, és végállomás van kiemelve, köztük a köztes állomások nevei és időpontjai láthatók a részútvonalnak megfelelő sorrendben.

Ha nem létezik a két állomás között útvonal, azt a következőképpen jelzi a felület:



Plan a trip!

Route doesn't exist!

Ha ugyanazt adjuk meg kezdő-, és végállomásnak azt pedig a következőképpen:

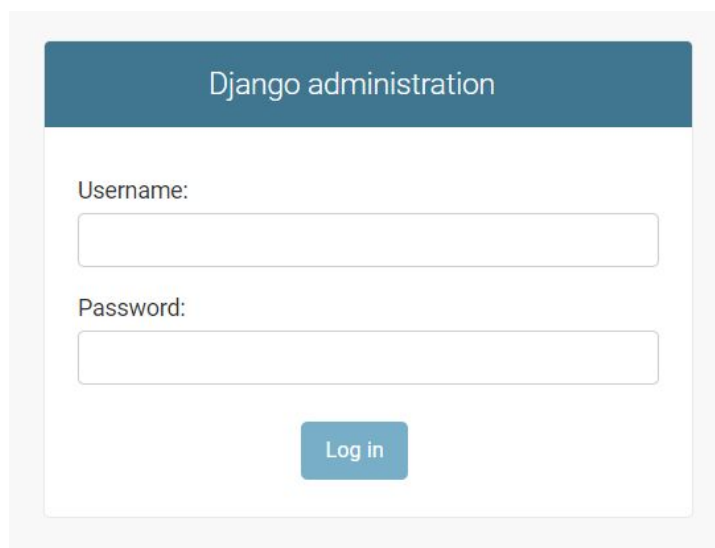
The start and destination can't be the same!

Admin interface terv

Rendszerünk egyik fő feladata volt, hogy az admin joggal rendelkező felhasználók tudjanak a nekik megfelelő funkciókat (lásd Funkciók című fejezet Admin alfejezet). Így ehhez kellett egy admin interfacet is terveznünk.

A djangóban eleve implementálva van egy admin interface, mellyel az adatbázis bejegyzéseit tudjuk módosítani, illetve ahhoz új bejegyzéseket hozzáadni vagy meglévőket törölni. Viszont ez viszonylag általános interface, melyet mi saját igényünknek megfelelően módosítottunk. Ennek következtében az admin interface-ünk a következőképpen néz ki:

Az admin először egy bejelentkezési felületet lát, melyen a felhasználónevét és jelszavát kell megadnia a belépéshez:

The image shows a screenshot of the Django administration login interface. It features a dark blue header bar with the text "Django administration" in white. Below the header, there are two input fields: "Username:" and "Password:". The "Username:" field is a simple text input, while the "Password:" field is a password input with a small eye icon to toggle visibility. Below these fields is a blue button with the text "Log in" in white. The entire form is set against a light gray background.

Ha sikeresen belép, a következő felület fogadja:

Site administration

TRIPPLANNER

Lines

+ Add

Change

Services

+ Add

Change

Stations

+ Add

Change

Timetable datas

Change

Recent actions

My actions

257 (13:50:00): 4

Service

63 (14:00:00): 2

Service

257 (13:50:00): 400

Passenger

Jobb oldalt felül át tud váltani a felhasználói oldalra, jelszót tud változtatni vagy ki tud jelentkezni. Bal oldalt a vonalakat, járatokat, állomásokat és menetrend bejegyzéseket tudja változtatni, illetve az utolsó kivételével ezekből újat felvenni. Középen a legfrissebb tevékenységeink láthatók.

A változtatáskor látható felület a következő:

Home > Tripplanner > Services

Select service to change

ADD SERVICE +

Action: Go 0 of 4 selected

| <input type="checkbox"/> | LINE | DEPARTURE TIME | TYPE | VALID FROM | VALID UNTIL | FEE |
|--------------------------|------|-----------------|---------|---------------|---------------|-----|
| <input type="checkbox"/> | 63 | 14:00:00 | normal | Nov. 29, 2019 | Nov. 30, 2019 | 2 |
| <input type="checkbox"/> | 257 | 23:16:17 | weekend | Nov. 23, 2019 | Nov. 23, 2019 | 100 |
| <input type="checkbox"/> | 257 | 23:16:17.733168 | normal | Nov. 22, 2019 | Nov. 23, 2019 | 300 |
| <input type="checkbox"/> | 257 | 13:50:00 | normal | Nov. 24, 2019 | Nov. 29, 2019 | 4 |

4 services

FILTER

By line

All

257

63

By type

All

normal

weekend

holiday

By valid from

Any date

Today

Past 7 days

This month

This year

By valid until

Any date

Today

Past 7 days

This month

This year



Bal oldalt-középen láthatók az adott modelltípusból az egyes bejegyzések soronként, oszlopaikban a különböző mezőket tartalmazva. Az oszlopok fejlécére kattintva rendezni tudjuk adott mező szerint a bejegyzéseket növekvő/csökkenő sorrendben. Jobb oldalt legfelül az "Add ... +" gomb átvizet minket a hozzáadó felületre. Alatta szűrni tudjuk a bejegyzéseket különböző mezők szerint, amiket beállítottunk. Például a képen látható járat bejegyzéseket a vonal, típus vagy akár az érvényesség kezdete/vége szerint tudjuk szűrni. Az oszlopokban az előbb említett mezőkön kívül a járat indulási ideje és ára van megjelenítve. Ugyanezen minta szerint alakítottuk ki a vonal, az állomás és a menetrend bejegyzés ezen felületeit.


Ha valamelyik sor elejére rákattintunk, szerkeszteni tudjuk az adott bejegyzést, a következő felület jelenik meg:


Home · Tripplanner · Services · 257 (13:50:00) · 4


Change service HISTORY

Fee: 4









Line: 257  

Valid from: 2019-11-24 Today 

Valid until: 2019-11-29 Today 





Type: normal 

TIMETABLE DATAS

| STATION | DATE TIME |
|--|--|
| 257 (Hévíz): 13:50:00   | 13:50:00 Now  |
| 257 (Hunyadi János utca): 13:53:00   | 13:53:00 Now  |
| 257 (Kőcsag utca): 13:55:00   | |

A változtatás felületen változtatni tudjuk az adott bejegyzéshez tartozó adatokat. Például a képen látható járatnál a vonal, költség, típus, érvényesség kezdete/vége adatokat illetve a járáshoz tartozó menetrend bejegyzések időpontját tudjuk szerkeszteni. Itt ha a menetrend bejegyzéseknél a “Change” ikonra kattintunk, az az adott menetrend bejegyzés változtató felületére visz át. Jobb felül a felületen a “History” gombra kattintva láthatjuk az adott bejegyzés eddigi összes változtatását időrendben.

Változtatásainkat a felület alján tudjuk elmenteni, ezzel együtt vagy folytatni a szerkesztést, vagy kilépni a mentés után vagy az új elem létrehozásának felületére váltani. Arra is lehetőség van, hogy a bejegyzést töröljük a bal alsó sarokban található “Delete” gombbal:




| | | |
|---|----------|---|
| Csatlós utca | 14:01:00 | Now  |
| 257 (Tutó utca): 14:03:00   | | |
| Tutó utca | 14:03:00 | Now  |


Delete
Save and add another
Save and continue editing
SAVE


Az új bejegyzés létrehozásának felületén az alapértékei jelennek meg az adott bejegyzéstípus mezőinek:


Add service

Fee: 1

Line:   

Valid from: 2019-11-29 Today 

Valid until: 2019-11-29 Today 

Type: normal 

TIMETABLE DATAS

| STATION | DATE TIME |
|---------|-----------|
|---------|-----------|

Save and add another
Save and continue editing
SAVE

Jelen példánknál azért nem jelentek meg a menetrend bejegyzések, mert még nincs a járathoz tartozó vonal értéke megadva. Miután megadjuk és elmentjük, ugyanúgy fog kinézni, mint a változtatás felülete.

Telepítési leírás

telepítsünk a Python 3.7-es változatát.

Telepítsük a django-t: “pip install django”

Az alkalmazás könyvtárában “python manage.py runserver” parancs kiadásával futtathatják a localhoston.

Saját szerverünknek megfelelő konfigurációt is írhatunk. Ehhez több információt talál itt: <https://docs.djangoproject.com/en/2.2/howto/deployment/wsgi/>