# Návrh

V tejto kapitole sa budeme zaoberať návrhom jednotlivých častí aplikácie. Najskôr spomenieme funkcie, ktoré bude naša aplikácia ponúkať. Ďalej popíšeme, ako sme spomedzi mnohých alternatív vybrali vhodný algoritmus pre potrebu našej aplikácie. Pri navrhovaní aplikácie sme sa venovali analýze získaných statických dát a dát o meškaní. Spomíname tiež, ako budeme pristupovať k týmto dátam pri implementácii, teda akým spôsobom ich uložíme do databázy a následne do dátovej štruktúry. V neposlednom rade spomenieme, ako bude fungovať naša aplikácia z pohľadu jej architektúry.

## 2.1 Funkcie aplikácie

Ako sme si mohli všimnúť v 1.9, všetky aplikácie ponúkajú vyhľadávanie z aktuálnej polohy rovnako ako aj možnosť výberu zastávky priamo z mapy. Tieto funkcie bude používateľovi ponúkať aj naša aplikácia. Väčšina spomenutých aplikácií ponúkala zobrazenie histórie vyhľadávania, ktorá odľahčí používateľa od zadávania parametrov v prípade, že vyhľadáva väčšinou tie isté spoje. Túto funkcionalitu nájde používateľ aj v našej aplikácii. História vyhľadávania sa bude ukladať do pamäte zariadenia.

Okrem aplikácie UBIAN a CP si používatelia vedia zobraziť všetky linky v MHD a postupnosti zastávok, ktoré obsluhujú. Túto funkciu bude ponúkať aj naša aplikácia. Čo sa týka nastavenia prídavných preferencií pri vyhľadávaní, aplikácie ponúkajú rôzne preferencie. Najčastejšími z nich sú: maximálny počet prestupov, minimálny čas na prestup, limit pre peší presun a zobrazenie len nízkopodlažých vozidiel. Naša aplikácia bude mať možnosť nastavenia všetkých týchto preferencií.

Jediná aplikácia, ktorá ponúka informácie o reálnom pohybe vozidiel vo forme meškania, je aplikácia UBIAN. Predpokladáme však, že aj táto aplikácia vyhľadáva v statických cestovných poriadkoch a pri vyhľadanom spoji len pripíše informáciu vo forme meškania. Uvažujeme tak na základe toho, že po vyhľadaní spojov zo zastávky po zadaní aktuálneho času aplikácia ponúkne také spoje, ktoré podľa statických cestovných poriadkov majú na túto zastávku v blízkej budúcnosti príchod. Ak existuje taký

spoj, ktorý mal odchod z danej zastávky v minulosti, ale má meškanie a na zastávke ešte nebol, aplikácia ho nezobrazí. Naša aplikácia ponúkne aj tie spojenia, ktoré kvôli meškaniu na zastávku ešte nedorazili. Pre vyhľadaný spoj zobrazí čas odchodu zo statických dát cestovného poriadku a pripíše k nemu informáciu o meškaní.

Medzi ďalšie funkcionality patrí zakúpenie lístka priamo cez aplikáciu. Táto funkcionalita však nesúvisí priamo so zadaním našej práce a navyše je potrebná zmluva s dopravcami. Preto naša aplikácia túto možnosť ponúkať nebude.

Len aplikácia CG Tranzit funguje aj v offine režime. Hoci je užitočné ponúknuť vyhľadávanie bez možnosti prístupu na internet, znamenalo by to že náš algoritmus by bežal na klientskej strane. Keďže hlavnou úlohou našej aplikácie je vyhľadávanie spojov z reálnych dát, aplikácia bude fungovať online s tým, že zaručí používateľom, vždy aktuálne spojenia.

V tabuľke 2.1 je zobrazený prehľad funkcionalít našej navrhovanej aplikácie a iných

existujúcich aplikácii na vyhľadávanie spojov v MHD Bratislava.

------------------------------tabuľka-------------------------------------------------------------------------

## 2.2 Algoritmus

Pri hľadaní algoritmu na nájdenie optimálnej cesty sme najskôr siahli po najznámejšom vyhľadávacom grafovom algoritme. Dijkstrov algoritmus 1.2.1 sa zdal vhodný, avšak jeho vylepšená verzia A\* algoritmus 1.2.2 je pri správne zvolenej heuristike efektívnejšia.

Keďže zastávky, ktoré predstavujú vrcholy v grafe majú dané súradnice, môžeme pri vyhľadávaní optimalizovať prehľadávaný priestor. Pri štúdiu článkov sme narazili na rôzne optimalizácie prehľadávaného priestoru. Minimalizácia v okolí virtuálnej cesty a minimalizácia bounding boxom sú spomenuté v 1.5.

V prípade cestovných poriadkov je náročné správne namodelovať graf, ktorý dokáže efektívne spracovať časovo závislé dáta. V 1.4 boli spomenuté dva overené prístupy Time dependent a Time expanded model, ktoré tento problém riešia.

Ďalšou výzvou je prispôsobiť grafový vyhľadávací algoritmus, aby dokázal vypočítať najoptimálnejšiu cestu, prihliadal na prestupy medzi rôznymi módmi a popri tom počítal s ďalšími pridanými kritériami. V 1.3 boli spomenuté návrhy časovo závislých algoritmov. Algoritmus 1.3.1 sa dokáže vysporiadať aj viacerými módmi. Vráti však len jednu cestu. V 1.7 je popísaný algoritmus, ktorý efektívne vyhľadáva rôzne alternatívne cesty.

Kvôli dynamickej povahe verejnej dopravy, grafový prístup v kombinácii s vyhľadávacím grafovým algoritmom vyžaduje veľa pre-processingu a to sa odráža na výpočtových časoch.

Výhodou vo verejnej doprave je, že vozidlá sa pohybujú po vyznačených linkách, ktorých trasy poznáme. Schéma verejnej dopravy sa preto dá zachytiť do pomerne jednoduchých dátových štruktúr. Tento fakt si všimli aj autori algoritmu RAPTOR, ktorý sme opísali v 1.8.

V našej aplikácii sme sa rozhodli použiť tento negrafový algoritmus. Jeho výhodou je, že nie je potrebné vytvárať model a nie je potrebné riešiť multimodalitu hromadnej dopravy. Ľahšie zvláda dynamickosť dát ako meškanie linky, zrušenie linky alebo zmenu trasy. Využijeme základnú verziu RAPTOR algoritmu popísanú v 1.8.2 a súčasne využijeme aj jeho optimalizáciu opísanú v 1.8.3, kedy označujeme zastávky, aby sme nemuseli prechádzať tie linky, ktorým sa nevylepšil čas Tauk-1(p). Optimalizáciu local-prunning nevyužijeme, keďže chceme použiť rozšírenú verziu RAPTOR algoritmu a to je rRAPTOR algoritmus popísaný v sekcii 1.8.4. Algoritmus rRAPTOR nám nevráti len jednu najkratšiu cestu, ktorá začína najskôr od zadaného času, ale vráti nám množinu najkratších ciest,

začínajúcich v zadanom časovom úseku.

Cesta, ktorú nám vráti základná verzia RAPTOR algoritmu je najkratšia a nezáleží koľko prestupov bude obsahovať. Našou úlohou je používateľovi poskytnúť optimálnu cestu a keďže optimálna cesta môže byť pre každého používateľa iná, ponúkneme mu viacero alternatív. Túto funkciu ponúka vylepšený RAPTOR algoritmus popísaný v 1.8.7. Algoritmus nám vráti viacero optimálnych ciest, pričom prihliada na to, aby jednotlivé cesty neboli rovnaké na veľkej časti úseku.

Našim cieľom je zlúčiť rRAPTOR algoritmus s vylepšeným RAPTOR algoritmom na hľadanie viacerých ciest a zakomponovať mechanizmus, schopný zohľadniť zadané používateľské preferencie, ktoré bude algoritmus prijímať ako vstupné parametre. Preferencie, ktoré bude vedieť algoritmus zohľadniť: minimálny čas na prestup, maximálny počet prestupov, maximálna dĺžka pešieho prestupu a vyhľadanie len nízkopodlažných spojov.

Na obrázku (algorithm.png) sú zachytené jednotlivé podalgoritmy, ktoré budú tvoriť náš výsledný algoritmus.

Výstupom z nášho algoritmu bude množina ciest, začínajúcich na zastávke ps, po čase Tau a končiacich v zastávke pt. Jednotlivé cesty sú optimálne, vyhovujú prípadným používateľským preferenciám a ich časy odchodov sú z časového intervalu <Tau; Tau +delta Tau>.

### 2.2.1 Vyhľadávanie z aktuálnej polohy

Ak používateľ zadá ako začiatočnú zastávku aktuálnu polohu, nájdeme zastávky v okolí radiálnym

vyhľadávaním, ako bolo spomenuté v 1.6. Najbližšia vyhľadaná zastávka k aktuálnemu bodu nemusí znamenať optimálne riešenie a preto budeme považovať za začiatočnú zastávku každú z nich. Pre každú zastávku zvlášť spustíme algoritmus, ktorý nám pre každú zo zastávok vráti množinu optimálnych ciest a na záver z nich vyberieme n najoptimálnejších ciest. Do finálneho riešenia zakomponujeme peší presun od aktuálnej polohy po začiatočnú zastávku vybraných optimálnych ciest.

## 2.3 Dáta

Pri vývoji aplikácie a pre jej testovanie sú nevyhnutné dáta. Pre účely aplikácie budeme potrebovať dáta zo statických cestovných poriadkov a dáta omeškaní jednotlivých jázd.

### 2.3.1 Dáta statických cestovných poriadkov

Od Dopravného podniku Bratislava sme získali statické cestovné poriadky, ktoré mali platnosť od 5.2.2018 – 31.12.2018. Dáta sú vo formáte GTFS.

#### 2.3.1.1 GTFS

General Transit Feed Specification (GTFS) je dohodnutý formát dát, ktorý používajú tisíce poskytovateľov verejnej dopravy a mnohé softvérové aplikácie. Špecifikácia definuje súbory, v ktorých sú reprezentované entity v tabuľke. V stĺpcoch sú popísané vlastnosti entity a v každom riadku je nový záznam.

Súbor **agency.txt obsahuje vlastnosti:** agency\_id, agency\_name, agency\_url, agency\_timezone,agency\_lang, agency\_phone. V našich dátach sa nechádza len jedna spoločnosť, ktorá prevádzkuje spoje a preto súbor s touto entitou nebudeme spracovávať.

Zástavky sú zachytené v súbore **stops.txt. Vlastnosti sú: s**top\_id, stop\_name, stop\_lat, stop\_lon, zone\_id. V zozname sa zastávka s rovnakým názvom (stop\_name) nachádza viac krát. Jedná sa skôr o názov platformy (napr. Račianske mýto), pretože jednotlivé zastávky sú umiestnené v rámci platformy na rôznych súradniciach.

Súbor **routes.txt** obashuje zoznam liniek. Pre každú linku ponúka vlasnotsti**:** route\_id, agency\_id, route\_short\_name, route\_long\_name, route\_type, route\_text\_color. Vlastnosti agency\_id a route\_text\_color budeme ignorovať pretože sú stále rovnaké. Vlasnosť route\_long\_name nie je určená. Vlastnosť route\_type predstavuje typ módu. Podľa štandardu GTFS, 0 definuje električku, 3 autobus a 11 trolejbus.

Jazdy jednotlivých liniek sú zaznamenané v súbore **trips.txt,** ktorý obsahuje vlastnosti: route\_id, service\_id, trip\_id, trip\_headsign, trip\_short\_name, direction\_id. Podľa GTFS špecifikácie by service\_id mala byť množina service\_day oddelená pomlčkou. V našich dátach existuje pre jeden záznam jazdy len jeden service\_day. V prípade, že jazda patrí viacerým service\_day, je vytvorený nový záznam s rovnakými vlastnosťami jazdy pre každý service\_day. Vlastnosť trip\_headsign predstavuje konečnú zastávku jazdy. Túto vlastnosť nevyužijeme pretože ju vieme získať iným spôsobom. V žiadnom zázname jazdy nie je vyplnené pole trip\_shortname, takže túto vlastnosť budeme ignorovať. Direction\_id má hodnoty 0 a 1. V dátach chýba informácia o nízkopodlažných spojoch.

Súbor **stop\_times.txt** má vlastnostitrip\_id, arrival\_time, departure\_time, stop\_id, stop\_sequence, stop\_headsign, pickup\_type, drop\_off\_type. Arrival\_time a departure\_time sú vždy rovnaké, preto jeden z nich budeme ignorovať. Vlastnosť stop\_sequence je poradie zastávky vrámci linky. Poradnie nie je iterované od 1, ale začína rôznymi číslami. Položka stop\_headsign nie je v žiadnom zázname vyplnená. Vlastnosť pickup\_type a drop\_off\_type nadobúdajú hodnoty 0 a 3. V GTFS špecifikácii znamená číslo 0, že jazda na tejto zastávke stojí vždy a 3 určuje, že na zastávke stojí len na znamenie. Hodnoty sú pre obe vlastnosti vždy rovnaké, takže jednu z nich môžeme opäť ignorovať.

**Súbor celandar.txt obsahuje stĺpce** service\_id, monday, tuesday, wednesday, thursday, friday, saturday, sunday, start\_date, end\_date. Vlasntosť service\_id predstavuje unikátny názov typu služby (pracovný deň, víkend,...). Vlasnosti ,monday,...sunday môžu nadobúdať hodnoty 0 a 1. Ak je napr. hodnota monday = 0, znamená to, že všetky pondelky medzi start\_date a end\_date patria do služby service\_id. Naopak, ak je hodnota 1, tak nejazdí.

V súbore **calendar-dates.txt sú vlastnosti:** service\_id, date, exception\_type. V prípade, že vlastnosť exception\_type má hodnotu 1, znamená to, že service\_id navyše jazdí v deň date. Ak je exception\_type = 2, znamená to, že v deň date service\_id nejazdí.

### 2.3.2 Dáta o meškaní

Podarilo sa nám získať aj informácie o meškaní za rok 2018, pre mesiac február, marec a apríl. Pre každý deň v mesiaci existuje súbor vo formáte .csv. Každý záznam obsahuje údaje:

Id – identifikačné číslo záznamu

Datum a cas – dátum a čas, kedy bol záznam o meškaní zaevidovaný

Vozidlo – identifikačné číslo vozidla

Linka – identifikačné číslo linky

Poradie – poradie jazdy v linke

Cislo zastavky – identifikačné číslo zastávky

Nazov zastavky – názov zastávky

Meskanie – hodnota meškania v minútach.

Z pozorovania dát vyplýva, že hodnoty meškania majú hodnotu 0, záporné celé čísla alebo hodnotu n/a. V prípade, že narazíme na hodnotu 0 alebo n/a, tento záznam neberieme do úvahy. Môže ale platiť, že hodnota 0 znamená, že meškanie je z intervalu <0 min, 1 min) minúta, hodnota 1 z intervalu <1 min, 2 min). Mení to nejako náš pohľad na vec? Na prelome dní sa tam vyskytuje hodnota -1229. Kto vie čo znamená?

Takže naša aplikácia bude môcť ponúknuť správne cesty s prihliadnutím na meškanie spojov od 5.2.2018 – 30.4.2018. Statické vyhľadávanie bez meškania bude správne fungovať od 5.2.2018 do konca roka 2018.

### 2.3.3 Pešie presuny

V dátach sa nenachádzajú informácie o peších presunoch. Google API ponúka možnosť vyhľadania peších vzdialeností medzi 2 bodmi. Počet bezplatných dopytov na Distance Matrix API je obmedzený. Nebudeme vyhľadávať pešie vzdialenosti medzi každou dvojicou zastávok, nakoľko je to nepotrebné.

Budeme postupovať nasledovne: pre každú zastávku p nájdeme zastávky v okolí 800 metrov radiálnym vyhľadávaním. Pomocou Distance Matrix API zistím vzdialenosti nájdených zastávok od zastávky p. Každú dvojicu zastávok uložíme do súboru foot\_paths.txt spolu so zistenými vzdialenosťami určených v minútach.

## 2.4 Databáza

Na serveri budú uložené dáta aplikácie v PostgreSQL databáze. Databáza sa naplní pri prvotnom spustení aplikácie na serveri. Schéma databázy je popísaná entitno-relačným diagramom na obrázku 2.2.

---------------------------------Obr. 2.2: Entitno-relačný diagram -------------------------------------------------------

Entita stop\_areas obsahuje zoskupenie zastávok, ktoré majú rovnaké názvy (name).

V entite stops evidujeme zónu mesta (zone), do ktorej zastávka patrí. Hodnota on\_request určuje, či sa na zastávke nastupuje a vystupuje na znamenie. Každá zastávka má priradené súradnice, ktoré sa udržujú v entite coords. V entite coords sú súradnice určené atribútmi zemepisná šírka (latitude) a zemepisná výška (longitude).

Entita foot\_paths obsahuje atribút time, ktorý určuje čas v sekundách potrebný na peší presun zo zastávky from na zastávku to.

V entite routes sa udržuje zoznam liniek, jazdiacich aktuálne v bratislavskej hromadnej doprave. Linka je určená názvom (name) a módom (mode). V Bratislave jazdia 3 rôzne módy: električka, trolejbus a autobus. Každá linka má počas dňa viaceré jazdy (trips).

Entita (trips) uchováva informáciu o tom, či je vozidlo, ktoré bolo pridelené konkrétnej jazde nízkopodlažné, ktorým smerom ide (direction) a zároveň počas akých typov dní (service\_day) jazda premáva. Každá jazda linky je tvorená postupnosťou zastávok, ktoré linka obsluhuje.

V entite stop\_times je zachytený čas (time), kedy jazda stojí na zastávke a v akom poradí sú zastávky v rámci jazdy (sequence\_order).

V entite service\_days sú názvy rôznych typov dní, v ktorých jazdia jazdy liniek.

Entita calendar\_dates obsahuje zoznam všetkých dátumov (date), v rozsahu platnosti aktuálneho cestovného poriadku. Ku každému dátumu je priradený typ dňa (service\_day). Jeden dátum môže prislúchať k viacerým typom dňa a naopak jeden typ dňa môže prislúchať viacerým dátumom.

## 2.5 Dátová štruktúra

Rovnako dôležitá je dobre navrhnutá dátová štruktúra, s ktorou bude algoritmus vedieť rýchlo a efektívne pracovať. Budeme sa inšpirovať dátovou štruktúrou z 1.8.6, ktorá bola navrhnutá pre základnú verziu RAPTOR algoritmu. Dátová štruktúra bude obsahovať aktuálne cestovné poriadky s prispôsobenými príchodmi a odchodmi vozidiel podľa prípadných meškaní.

Dátová štruktúra uvedená v článku ráta s tým, že jednotlivé jazdy v rámci linky majú rovnakú postupnosť zastávok. V našich dátach to tak nie je. Linka obsahuje jazdy, ktoré idú jedným aj druhým smerom. Zástavky cez ktoré prechádza linka majú síce rovnaký názov, ale majú iné identifikačné čísla, ktorými sú definované. Okrem rôznych smerov obsahuje linka aj také jazdy, ktorých postupnosť zastávok je iná ako pri väčšine. Najmä v ranných a večerných hodinách prechádzajú niektoré jazdy len cez určitú podpostupnosť zastávok.

RAPTOR algoritmus potrebuje, aby všetky jazdy, ktoré patria konkrétnej linke mali rovnakú postupnosť zastávok. Rozhodli sme sa preto zoskupiť jazdy s rovnakou postupnosťou zastávok do úsekov. Teraz platí, že 1 linka má viacero úsekov a jednému úseku linky prislúcha viac jázd linky.

Často sa potrebujeme dopytovať na všetky linky, ktoré stoja na konkrétnej zastávke a rovnako potrebujem vedieť postupnosť zastávok, ktoré patria konkrétnej linke.

StopTimes bude objekt alebo pole, ktoré obsahuje nielen informáciu o čase, kedy podľa statického poriadku má stáť jazda linky na zastávke, ale aj informáciu o predpokladanom časovom meškaní danej jazdy na zastávku

VYMYSLIEŤ a Popísať čo všetko sme tam zmenili.

## 2.6 Architektúra aplikácie

Klient sa bude dopytovať na server pre vyhľadanie spojenia. Server spustí výpočet nad dátovou štruktúrou, ktorá má aktuálne cestovné poriadky s informáciou o prípadnom meškaní spojov a vráti odpoveď klientovi.

Algoritmus bude pracovať nad dátovou štruktúrou, ktorá bude obsahovať stále aktuálne dáta. Dátová štruktúra bude rovnako ako algoritmus uložená na serverovej strane.

### 2.6.1 Serverová strana

Na serverovej strane bude bežať aplikačný server Tomcat. Na uchovanie dát použijeme relačnú databázu. Na komunikáciu s klientom budeme používať REST API.

### 2.6.2 Klientská strana

Na klientskej strane sme sa rozhodli pre progresívnu webová aplikácia PWA. Je to webová aplikácia, ktorá sa dokáže správať ako mobilná aplikácia, ktorá sa neustále aktualizuje, pričom nie je potrebná jej inštalácia. Po návšteve webovej stránky na mobilnom zariadení používateľ dostane upozornenie od stránky, či si ju chce uložiť do zariadenia ako mobilnú aplikáciu. Progresívna webová aplikácia zaberá minimum miesta v pamäti a má svoj vlastný úložný priestor, kde sa budú ukladať preferencie a história vyhľadávania.

### 2.6.3 Spracovanie dát

Pri spustení aplikácie alebo po aktualizácii cestovných poriadkov sa spustí služba, ktorá nám z úložiska, kde sú aktuálne cestovné poriadky namapuje dáta do našej databázy. Po tom ako budú dáta uložené v databáze sa spustí ďalšia služba, ktorá obnoví dátovú štruktúru podľa nových cestovných poriadkov.

Ďalšia služba bude vytvorená na spracovanie údajov o meškaní. Hoci máme v súbore pre konkrétny deň údaje o meškaní jázd na celý deň, chceme sa čo najviac priblížiť reálnemu nasadeniu. Budeme teda rátať s tým, že nové údaje o meškaní pribúdajú po minúte. Služba bude spúšťaná každú minútu. Bude čítať súbor pre aktuálny deň, získa záznamy, ktoré pribudli v poslednej minúte a aktualizuje meškanie pre konkrétnu jazdu. Údaje o meškaní sú evidované pre zastávku s, na ktorej meškanie vzniklo. Aktualizácia meškania jazdy bude prebiehať tak, že pre všetky zastávky jazdy od zastávky s až po konečnú zastávku jazdy zapíše do dátovej štruktúry hodnotu získaného meškania.

Spôsob akým bude aplikácia nasadená je znázornená na obrázku 2.1.

---------------------------------------------------Obr. 2.1: Diagram nasadenia---------------------------------------------

# Implementácia

Na implementáciu použijeme moderný JavaScriptový framework VUE.js.

Jazyk, v ktorej bude aplikácia implementovaná je Java 11.

Na mapovanie java objektov na tabuľky databázy použijeme ORM framework Hibernate. Hibenate zároveň ponúka implementáciu Java Persistnace API.

Na vytvorenie RESTových webových služieb použijeme na serverovej strane Spring framework. Na špecifikáciu REST API použijeme Swagger framework. API budeme popisovať v JSON súbore.

**Import -** meranie času vykonania importu jednotlivých súborov

**stops.txt** - (8,3s) so stopAreas (12s)

**routes.**txt – (0,07s)

**calendar + calendar\_dates.txt –** 18 687 915 800 (18.75s) – hľadanie serviceDay v databáze

387 457 000 (0.4s) – na začiatku všetkých z db, hľadanie v poli

**trips.txt – I. (18.45 min) II. (0,5s)**

split rows – 541 850 000 (0,5s)

split row Luki – 5000

I. serviceDay z db – 17 304 100

I. route z db – 8 464 800

II. serviceDay z pola - 1 300

II. route z pola – 7 300

**stop\_times.txt –(13.5 min)**

**HIBERNATE**

spring.jpa.hibernate.ddl-auto pôvodne nastavené na **create-drop**: pri každom spustení sa premaže databáza a vytvorí sa na novo. Zmenené na **update**: pri spustení by mali zostať dáta. Pred importom ich bude treba premazať.

**Importovanie dát zo súboru do databázy**

Pri čítaní súboru stops.txt sme vytvorili 4 tabuľky v našej databáze. Polohu zastávky sme vyčlenili do zvlášť entity coords. Zastávky s rôznymi súradnicami majú rovnaký názov. Zastávky s rovnakým názvom sme zoskupili do entity stop\_areas. Keďže v dátach nemáme informáciu o peších presunoch medzi jednotlivými zastávkami, ... v entite foot\_paths.

Každý dátum z rozmedzia dátumov platnosti aktuálneho cestovného poriadku potrebujeme zaradiť do nejakej skupiny dní (pracovné dni, víkendy, školské prázdniny, štátne sviatky,...). Z dát uložených v  GTFS štandarde sa komplikovane zisťuje, ktoré skupiny dní prislúchajú konkrétnemu dátumu. V entite calendar\_dates budeme mať uložené všetky dátumy z rozmedzia dátumov platnosti cestovného poriadku. V entite service\_days budeme mať uložené jednotlivé skupiny dní.

Pri súboroch stops, trips, routes a stop\_times len kopírujeme štruktúru databázy podľa štruktúry týchto súborov. Jediou zmena je, že informáciu o tom, či je zastávka na znamenie si neudržujeme v tabuľke stop\_times pre každú jazdu zvlášť, ale primo v enetite stops.