VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta informačních technologií

POKROČILÉ DATABÁZOVÉ SYSTÉMY

2024/2025

Projekt

Pokročilý systém monitorování a řízení dopravních spojů

David Drtil (xdrtil03)

Dominik Pop (xpopdo00)

Brno, 1. prosince 2024

Obsah

1. Specifikace

Aplikace bude sloužit k monitorování a zobrazování aktuálních dopravních spojů pro veřejnou dopravu, jimiž jsou autobusy, tramvaje a vlaky. Kromě základních informací o trasách a spojích nabídne uživateli i pokročilé funkce, jako je aktuální poloha vozidel v reálném čase, obsazenost vozidel, teplota uvnitř vozidla, predikce zpoždění, zobrazení nehod na trasách a další užitečné metriky.

Systém využívá relační databáze (SQL) pro správu klíčových dat o zastávkách a trasách, která podléhají častým změnám, z důvodu uzavírek, výluk či nehod. Nerelační databáze (NoSQL) pak slouží k ukládání velkých objemů dat, která se postupně hromadí a typicky zůstávají neměnná. V případě naší aplikace se jedná o záznamy senzorů sledujících aktuální polohu vozidel nebo historii vyhledávání spojů uživateli. Oba databázové systémy budou propojeny a umožní efektivní kombinaci transakčně orientovaných datových operací a zároveň rychlé dotazování nad rozsáhlými daty.

1.1. Popis datové sady

Data o vozidle v reálném čase budeme čerpat z datové sady obsahující polohy v zeměpisné šířce a délce pro vozidla MHD statutárního města Brno. Vozidla jsou rozlišena identifikátorem „`id`“. Kromě polohy sada obsahuje také dodatečné informace o vozidlech, jako jeho zpoždění, úhel směru vozidla, typ vozidla, kurzové číslo vozu, nízkopodlažnost vozidla, či jestli je vozidlo právě aktivní. Dále sada obsahuje sloupce „`laststopid`“ a „`finalstopid`“, což jsou identifikátory pro poslední navštívenou a koncovou zastávku. Tyto identifikátory se odkazují do datové sady Jízdní řád IDS JMK ve formátu GTFS, kde se v souboru stops.txt nachází zastávky, na kterých zastavují. Stejně tomu tak je u sloupce „`lineid`“, který se odkazuje do souboru routes.txt. Posledním sloupcem je „`lastupdate`“, což je časový údaj, konkrétně datum, pro který jsou daná data z řádku aktuální.

Data z datové sady jsou pravděpodobně získávána ze GPS senzorů nacházejících se na jednotlivých vozidlech. Tyto senzory posílají data v malých intervalech na server, kde jsou ukládána. Oprava zasílaných dat v případě nějaké chyby se zde pravděpodobně vzhledem k vysoké frekvenci zasílání dat neprovádí.

Použitá datová sada je pouze úryvek, celá sada se nachází zde. Vzhledem k objemu dat jsou zde ukládány pouze data stará maximálně dva dny.

1.2. Operace nad systémem

Tabulky v relační databázi mohou být modifikovány pomocí *CREATE, UPDATE, DELETE* operací, přičemž níže jsou uvedeny pouze klíčové operace.

1.2.1. Ukládání dat

- createUserProfile(name, email, password) – vytvořit uživatelský profil
- createRouteTicket(userId, scheduledRouteId, seatId) – přidat jízdenku včetně vytvoření rezervace sedadla
- *addLocationToFavourite(userId, locationId) – přidat lokaci do oblíbených
- *addLineToFavourite(userId, lineId) – přidat linku do oblíbených

- createVehicle(vehicleTypeId, label, capacity) – vytvořit vozidlo
- createVehicleService(vehicleId, serviceName, description) – přidat službu vozidla
- createLocation(name, latitude, longitude) – vytvořit lokaci (nástupiště zastávky)
- createStop(locationId, name, latitude, longitude) – vytvořit zastávku
- createRoute(routeStops[], startStopId, endStopId) – vytvořit trasu z kolekce zastávek
- createLine(activeRouteId, name) – vytvořit dopravní linku
- scheduleRoute(routeId, startTime, endTime) – vytvořit spoj, neboli záznam o naplánované trase
- *[1]addVehicleStatus(vehicleId, timestamp, vehicleData) – přidat status o vozidle v daný čas, kde parametr vehicleData obsahuje polohu vozidla, rychlost, teplotu a další údaje
- *addSearchedRoute(userId, startStopId, endStopId) – přidat vyhledávanou trasu (počáteční a koncovou zastávku) uživatelem pro budoucí návrhy tras

1.2.2. Úprava dat

- updateVehicle(vehicleId, label, capacity) – aktualizovat základní data o vozidle
- updateRoute(routeId, startStopId, endStopId, routeStops[]) – aktualizovat trasu včetně kolekce zastávek trasy
- updateStop(stopId, name, latitude, longitude, isValid) – upravit zastávku (včetně nastavení příznaku neplatné zastávky)
- rescheduleRoute(routeId, startTime, endTime) – přeplánovat spoj

1.2.3. Vymazání dat

- deleteScheduledRoute(scheduledRouteId) – odstranit spoj
- deleteVehicleService(vehicleId) – odstranit službu vozidla
- deleteRouteTicket(userId, scheduledRouteId) – odstranit jízdenku
- removeLineFromFavourite(userId, lineId) – odebrat linku z oblíbených
- removeLocationFromFavourite(userId, locationId) - odebrat lokaci z oblíbených

1.3. Dotazy nad systémem

- getLine(name) – získat linku podle názvu (např. šalina č. 12)
- getUserFavouriteLines() – získat *N* oblíbených linek
- getScheduledRoute(routeId) – získat detail spoje (získá počáteční a koncový čas, ID vozidla, mezizastávky)
- getAvailableSeats(scheduledRouteId) – získat dostupná sedadla spoje
- *getLocation(name) – získat lokaci podle názvu
- *getClosestLocation(latitude, longitude) – získat nejbližší lokaci od polohy uživatele
- *getScheduledRoutes(startLocationId, endLocationId, startTime) – získat *N* spojů podle počáteční a cílové destinace a času odjezdu
- *getFavouriteLocations(userId) – získat *N* oblíbených lokací
- *getLocations(locationSearchPhrase) – získat *N* lokací podle vyhledávané fráze
- *getSearchHistory(userId) – získat posledních *N* vyhledávaných tras uživatele
- *getVehicleRealTimeInfo(vehicleId) – získat polohu a další informace o vozidle daného spoje

[1] (*) Označené operace či dotazy budou prováděny nad nerelační databází.

1.4. Testovací scénáře

ID: #1
Název: Vyhledání spojů
Kroky:

1. Získání počáteční lokace – *getLocation(name)*
2. Získání koncové lokace – *getLocation(name)*
3. Zadání času odjezdu
4. Vyhledání spojů (získání nejlepších výsledků vyhledávání podle ID počáteční lokace, ID koncové lokace a času odjezdu) – *getScheduledRoutes(startLocationId, endLocationId, startTime)*

Očekávaný výsledek:

1. Získání *N* spojů s danou počáteční a koncovou lokací a časem odjezdu od zadaného času
2. V uživatelské historii vyhledávání spojů je uložen záznam o právě vyhledávané trase (počáteční lokace → cílová lokace) – *getSearchHistory(userId)*

ID: #2
Název: Načtení spoje
Kroky:

1. Vyhledání spojů stejně jako ve scénáři s ID #1, z nichž získá mimo jiné i ID spoje
2. Získání detailu spoje získá (počáteční a koncový čas, ID vozidla, mezizastávky) – *getScheduledRoute(routeId)*
3. Získání nejaktuálnější informace o vozidle (poloha, zpoždění, poslední projetá zastávka, příznak zda je spoj v koloně nebo je porouchané, zaplněnost vozidla, rychlost, teplota uvnitř vozidla) – *getVehicleRealTimeInfo(vehicleId)*

Očekávaný výsledek:

1. Kompletní načtení detailu spoje, včetně dat o vozidle.

ID: #3
Název: Přidání oblíbené lokace, včetně jejího následného odebrání
Kroky:

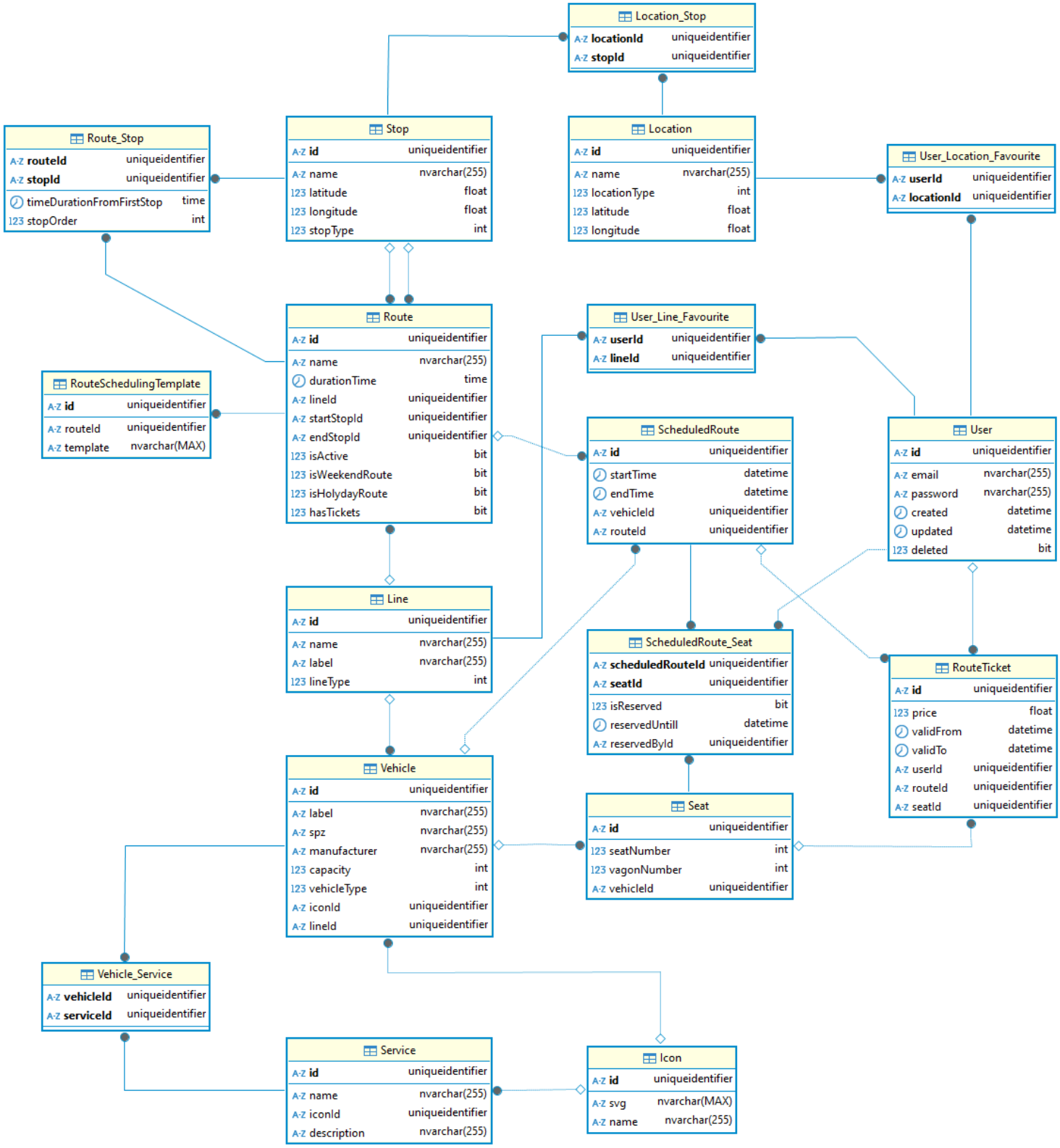
1. Vytvoří nového uživatele
2. Přihlášení uživatele do systému
3. Vyhledá lokaci a získá její ID – *getLocation(name)*
4. Přidá lokaci do oblíbených – *addLocationToFavourite(userId, locationId)*
5. Načte *N* oblíbených oblíbených lokací – *getFavouriteLocations(userId)*
6. Vyhledá 1. oblíbenou lokaci – *getLocation(locationId)*
7. Odebere oblíbenou lokaci – *removeLocationFromFavourite(userId)*
8. Načte *N* oblíbených oblíbených lokací – *getFavouriteLocations(userId)*

Očekávaný výsledek:

1. U nového uživatele se přidaná oblíbená lokace se objeví v oblíbených jako jediná.
2. Po odebrání oblíbené lokace je kolekce oblíbených lokací prázdná.

2. Analýza

2.1. Schéma relační DB



2.2. Popis NoSQL dat

NoSQL databáze v této aplikaci bude sloužit k uchování dvou hlavních typů dat:

1. **Data o vozidlech**: Informace o vozidlech v provozu, budou využita pro dotazy týkající se spojů a jejich stavu.
2. **Uživatelská data**: Záznamy o vyhledávání lokalit uživatelem, využívány pro doporučování lokací a historii hledání.
3. **Naplánované trasy:** Záznamy o naplánované trase doplněné o zastávky a konkrétní čas odjezdu ze zastávky. Využity pro vyhledávání tras.

2.2.1. Společné vlastností dat

- **Velikost dokumentů**: Všechny dokumenty budou relativně malé. U každého typu dat budou ukládány jednoduché záznamy, které nezahrnují velké množství polí nebo komplexní struktury.
- **Spolehlivost**: Pro všechny typy dat je spolehlivost kritická, protože se používají pro klíčové funkce aplikace.
- **Dostupnost**: Vysoká dostupnost dat je klíčová, ať už se jedná o záznamy o vozidlech nebo o historii uživatelských vyhledávání.
- **Škálovatelnost**: Systém musí být schopný dobré škálovatelnosti, aby dokázal reagovat na rostoucí počet uživatelů i nárůst objemu dat generovaných senzory vozidel.

2.2.2. Data o vozidlech v reálném čase

Jedná se o data, které generují senzory samostatných vozidel. Každý dokument reprezentuje záznam o konkrétním vozidle v daném časovém okamžiku. Záznamy jsou generovány v krátkých časových intervalech, v rozmezí 5-10 sekund.

Použití:

- Zobrazení aktuální polohy vozidla na mapě.
- Zobrazení informací o vozidle obsluhujícím linku.
- Monitorování zpoždění a výkonnosti vozidla.

Struktura:

```
{
    "_id": "GUID", // VehicleId
    "updated": "2024-10-15T08:45:00Z", // Timestamp of the event
    "coordinates": {
      "latitude": 50.0755,
      "longitude": 14.4378
    },
    "speed": 60.0, // Speed of the vehicle in km/h - double
    "temperature": 22.5, // Temperature inside the vehicle in Celsius - double
    "occupancy": 20, // Number of passengers inside the vehicle - int
```

```
    "delay": 5, // Delay in minutes - int
    "isInactive": false, // Boolean if vehicle is active or not
    "isStuck": false, // Boolean if vehicle is active or not
    "lineId": "GUID",
    "scheduledRouteId": "GUID",
    "lastStopId": "GUID",
    "finalStopId": "GUID",
  }
```

Vlastnosti:

- **Zdroje dat:** Senzory umístěné ve vozidlech.
- **Rychlost vzniku dat**: Data jsou generována v reálném čase, v intervalech 5–10 sekund.
- **Doba platnosti**: Kratší, 24 hodin
- **Rozsahy hodnot:**
    - Rychlost: 0–200 km/h.
    - Teplota: -50 až 50 °C.
    - Souřadnice: Geografické souřadnice v rozsahu zeměpisných poloh.
        - zeměpisná šířka: od −90° (jižní pól) do +90° (severní pól)
        - zeměpisná délka: od −180° na západ od nultého poledníku do +180° na východ

2.2.3. Uživatelské vyhledávání spojů

Jedná se o záznamy o vyhledávání spojů, které uživatelé zadávají v aplikaci. Tyto dokumenty obsahují informace o lokalitách, v rámci kterých uživatel vyhledával spoje.

Použití:

- Zobrazení historie vyhledávání.
- Doporučování lokalit na základě historie vyhledávání.

Struktura:

```
{
    "userId": "GUID",
    "fromLocation": "string",
    "toLocation": "string",
    "timestamp": "2024-10-15T08:45:00Z",
    "searchResults": ["GUID", "GUID"] // List of scheduledRouteIds
}
```

Vlastnosti:

- **Zdroje dat:** Interakce uživatelů s aplikací.
- **Rychlost vzniku dat**: Závisí na počtu aktivních uživatelů a jejich aktivitě.
- **Doba platnosti**: Data mají střední dobu platnosti, 2 týdny.
- **Rozsahy hodnot:** Textové hodnoty představující názvy lokalit.

2.2.4. Lokace

Tato data představují statické záznamy o lokalitách.

Použití:

- Navrhování nejbližší lokality na základě polohy uživatele.

Struktura:

```
{
  "locationId": "GUID",
  "name": "string",
  "type": "string",
  "coordinates": {
    "latitude": 50.084,
    "longitude": 14.4208
  }
}
```

Vlastnosti:

- **Zdroje dat:** Relační databáze, která ukládá všechny dostupné lokace.
- **Rychlost vzniku dat:** Velmi malá, protože se jedná o statická data, která se nemění často.
- **Doba platnosti:** Neomezená, protože tyto záznamy jsou dlouhodobě platné.
- **Rozsahy hodnot:** Geografické souřadnice v rozsahu zeměpisných poloh.
    - zeměpisná šířka: od −90° (jižní pól) do +90° (severní pól)
    - zeměpisná délka: od −180° na západ od nultého poledníku do +180° na východ

2.2.5. Naplánované trasy

Jedná se o dokumenty reprezentující jednotlivé naplánované trasy, které budou synchronizované s relační databází, ale obohacené o konkrétní zastávky a jejich časy odjezdu.

Použití:

- Cachování naplánovaných tras pro rychlé vyhledávání.

Struktura:

```
{
  "scheduledRouteId": "GUID",
  "routeId": "GUID",
  "vehicleId": "GUID",
  "startTime": "2024-01-01T00:00:00Z",
  "end": "2024-01-01T00:00:00Z",
  "stops": [ // List of stop documents
    {
        "stopId": "GUID",
```

```
        "startTime": "2024-01-01T00:00:00Z",
        "order": 1
    }
  ]
}
```

Vlastnosti:

- **Zdroje dat:** Relační databáze, která ukládá naplánované trasy.
- **Rychlost vzniku dat:** Velmi malá, jednou za rok.
- **Doba platnosti:** Jeden rok.
- **Rozsahy hodnot:** Textové a datumy.
    - **Dokumenty** obsahující informace o zastávkách naplánované trasy
        - Textové, datumy, celočíselné

2.3. Popis způsobu aktualizace

2.3.1. Relační DB

Aktualizace dat v relační databázi probíhá prostřednictvím standardních CRUD SQL operací, jako je INSERT, UPDATE a DELETE. Aktualizace relační databáze budou vyvolávat dva typy operací, administrátorské a uživatelské.

Uživatelské operace, provádějící aktualizace, se budou týkat hlavně uživatelských účtů, "favourite" relací a nákupu jízdenek. Nad uživatelskými účty se převážně bude provádět vkládání a editace. Mazání bude podporováno formou "soft delete". Mnohem běžnější budou ovšem operace pro relace a jízdenky. U obou se bude provádět pouze vkládání a mazání. Editace není pro tyto případy nutná.

Pod **administrátorské operace** spadá jakákoliv úprava dat spojů, tras, lokací, zastávek a vozidel. Hlavním typem takové operace bude vytváření nových naplánovaných tras, které bude prováděno automaticky jednou za rok, a přidáno bude obsáhlé množství dat. O přidání se bude starat automatizovaný skript, který pro každou trasu naplánuje trasy pro následující rok na základě informací v databázi a šablony pro plánování trasy. Trasy se budou také mazat na základě jejich stáří. Změny nad ostatními daty se nebudou provádět příliš často, spíše výjimečně a předpokládá se hlavně využití operace vkládání.

2.3.2. NoSQL

Data o vozidlech:

Dokumenty o vozidlech budou vytvářeny periodicky v krátkém časovém rozsahu (5-10 sekund). Předpokládá se hlavně operace vkládání, kdy se bude vkládat nový záznam o vozidle s časovým razítkem platnosti. Dokumenty o platnosti starší jak 24 hodin se budou automaticky mazat. Editace dokumentů se nepředpokládá, jelikož záznam popisuje stav vozidla v daném čase.

Druhou možnou variantou aktualizací by bylo pomocí operace upsert, kdy by se o vozidle udržoval pouze jeden aktuální záznam. Tato varianta by byla lepší co se do počtu uchovávaných záznamů v

databázi týče, ovšem nedaly by se pak provádět různé pozorování nad daty v čase. I zde by se dokumenty mazaly a to v případě, že by nepřišel žádný update po dobu 24 hodin.

Uživatelské vyhledávání spojů:

Dokumenty reprezentující vyhledání spoje uživatelem budou vytvářeny pokaždé, když uživatel zadá nové vyhledávání. Každý vyhledávací dotaz se tedy přidá do historie hledání. Předpokládá se tedy, že operace vložení se nad touto kolekcí bude provádět velice často. Historie vyhledávání bude uchovávána po dobu 2 týdnů, a poté budou staré záznamy smazány. Nepředpokládá se provádění aktualizace dokumentů, jelikož se jedná o záznam neměnné události.

Lokace:

Aktualizace lokací budou prováděny pouze při změně statických informací o lokalitách, což bude probíhat velice zřídka. Jelikož zdrojem těchto dat je relační databáze, tak bude aktualizace bude automaticky vyvolána na základě úpravy dat v relační databázi.

Naplánované trasy:

Aktualizace budou prováděny jednou za rok. Trasy budou načteny z relační databáze a doplněny o zastávky s jejich časem odjezdu. Mazání bude probíhat automaticky, po expiraci záznamu (1 rok). Editace se bude vyvolávat pouze na základě změny v relační databázi a to velice zřídka.

3. Návrh architektury

Pro implementaci relační databáze zvolíme **Microsoft SQL Server**, který vyniká integritou dat prostřednictvím transakčního zpracování a podporou referenční integrity pomocí cizích klíčů, což je klíčové pro zajištění konzistentnosti v našem systému monitorování a řízení dopravních spojů. Tento databázový systém navíc také nabízí skvělou integraci se zvolenými .NET technologiemi.

Pro implementaci nerelační databáze zvolíme dokumentově orientovanou NoSQL databázi **MongoDB**, která je ideální pro ukládání dynamických, rychle rostoucích dat, jako jsou údaje o aktuální poloze vozidel, jejich obsazenosti a teplotě. MongoDB poskytuje vysokou škálovatelnost, což umožňuje čtení i zápis velkého objemu dat bez výrazného dopadu na výkon, a podporuje rychlé dotazování na polohové údaje díky integrované geoprostorové podpoře. Tato schopnost efektivního zpracování geografických dat je klíčová pro sledování a řízení dopravních prostředků v reálném čase.

3.1. Architektura systému

Systém bude navržen v souladu s principem Command-Query Separation (CQRS), budou tedy odděleny operace, které mění stav systému (příkazy), od operací, které pouze čtou data (dotazy). Příkazy budou prováděny relační databází, zatímco dotazy budou zastřešeny nerelační databází.

Pro integraci databází bude využito ASP.NET Core Web API, které bude vystavovat rozhraní pro přístup k datům uloženým v relační i nerelační databázi. Systém bude využívat architekturu mikroslužeb, kde každá služba bude zodpovědná za určitou část aplikace, například správa tras, spojů nebo monitorování vozidel.

3.2. Testování

Testování funkčnosti systému bude probíhat pomocí jednotkových testů (anglicky Unit Tests). Pro každou část systému, například správu vozidel, tras či zastávek, bude vytvořena samostatná sada testů, které budou ověřovat správnost jednotlivých metod řadičů aplikačního rozhraní pro načtení, vytvoření, úpravu a případné smazání dat. Inicializační metody jednotlivých testů budou zajišťovat přípravu potřebných dat, například vytvoření testovacích zastávek, tras a simulaci provozu vozidel. Testovací data budou generována pomocí nuget balíčku *Bogus*, pomocí něhož zajistíme realisticky vypadající data.

Integrační testy budou ověřovat správnou komunikaci mezi relační a nerelační databází a zajištění konzistence dat napříč systémy. Testuje se série API metod, např. pro vytvoření entity, její následné získání, úpravu a vymazání.

Pro účely testování je využit `framework xUnit`. SQL databáze je simulována pomocí `SQLite`, zatímco pro NoSQL databázi je v MongoDB vytvořena dedikovaná testovací databáze s názvem `transitconnex_test`.

4. Implementace

4.1. Struktura projektu

- **Tests**
  - **TransitConnex.Test** - obsahuje unit/integrační testy na aplikační rozhraní
  - **TransitConnex.TestSeeds** - obsahuje seed classy pro naseedovaní testing dat do NoSQL/relační DB
- **TransitConnex.API**
  - Controllers - obsahuje controllery s definovanými endpointy
  - Handlers - obsahuje command/query handlery pro zpracování operace, zpracovávají synchronizace mezi relační a NoSQL DB
  - Middleware
  - Configuration
  - Automapping - obsahuje definice pro použití automapperu
- **TransitConnex.Command**
  - Commands - obsahuje command classes pro Create/Update/Delete + specifické operace nad relační DB
  - Data - obsahuje DB kontext pro relační DB
  - Migrations - obsahují migrace relační DB
  - Repositories - obsahují operace nad DB
  - Services - obsahují operace potřebné pro command/query endpointy nad relační DB
- **TransitConnex.Domain**
  - Collections - obsahují definice entit uložených v NoSQL DB
  - Models - obsahují definice entit uložených v relační DB
  - Mappings - TODO -> delete??

    - Enums
    - DTOs - obsahují DTO posílané na FE
- **TransitConnex.Query**
    - Abstraction
    - Data - obsahuje DB kontext pro NoSQL DB
    - Queries - obsahuje query classes pro quries nad NoSQL/relační DB
    - Repositories - obsahují operace nad NoSQL DB
    - Services - obsahují operace pro query endpointy nad NoSQL DB

4.2. Spuštění aplikace

4.2.1. Prerekvizity

- `SQL Server / MSSQL` včetně nastavení autorizace viz. */src/doc/SqlServerSetup.pdf*
- `MongoDb` s vytvořeným spojením s názvem `pdb-project`
- `Visual Studio 2022` pro překlad a spuštění

4.2.2. Lokální spuštění

1. Z příkazové řádky ve Visual Studio aktualizujte `Entity FrameWork Tools` příkazem: `dotnet tool update global dotnet-ef`
2. Přejděte do složky se implementací serveru a spusťte příkaz `dotnet ef database update`, který by vám měl vytvořit v databázi tabulky dle schématu z migrací.
3. Následně je možné příkazy:
    - `dotnet run unseed` – odstranit všechna data z databáze
    - `dotnet run seed` – nahrát ukázková data do databáze
    - `dotnet run` – spustí aplikaci

4.3. REST API

Endpointy jsou podrobněji popsané v implementaci pomocí DOXYGEN dokumentace.

4.3.1. Commands

- Icon
    - [POST]
        - `'api/Icon' - CreateIcon(IconCreateCommand)`
    - [PUT]
        - `'api/Icon' - EditIcon(IconUpdateCommand)`
    - [DELETE]
        - `'api/Icon' - DeleteIcon(Guid)`
- Line
    - [POST]
        - `'api/Line' - CreateLine(LineCreateCommand)`
        - `'api/Line/batch' - CreateLines(LineBatchCreateCommand)`
    - [PUT]

        - 'api/Line' - EditLine(LineUpdateCommand)
        - 'api/Line/batch' - EditLines(LineBatchUpdateCommand)
    - [DELETE]
        - 'api/Line/{id}' - DeleteLine(Guid)
- Location
    - [POST]
        - 'api/Location' - CreateLocation(LocationCreateCommand)
        - 'api/Location/batch' - CreateLocations(LocationBatchCreateCommand)
    - [PUT]
        - 'api/Location' - EditLocation(LocationUpdateCommand)
    - [DELETE]
        - 'api/Location/{id}' - DeleteLocation(Guid)
- Route
    - [POST]
        - 'api/Route' - CreateRoute(RouteCreateCommand)
        - 'api/Route/batch' - CreateRoutes(RouteBatchCreateCommand)
        - 'api/Route/add-stop' - AddStopToRoute(RouteStopAddCommand)
    - [PUT]
        - 'api/Route' - EditRoute(RouteUpdateCommand)
    - [DELETE]
        - 'api/Route/{id}' - DeleteRoute(Guid)
        - 'api/Route/remove-stop' - RemoveFromToRoute(RouteStopRemoveCommand)
- RouteSchedulingTemplate
    - [POST]
        - 'api/RouteSchedulingTemplate' - CreateRouteSchedulingTemplate(RouteSchedulingTemplateCreateCommand)
    - [PUT]
        - 'api/RouteSchedulingTemplate' - EditRouteSchedulingTemplate(RouteSchedulingTemplateUpdateCommand)
    - [DELETE]
        - 'api/RouteSchedulingTemplate/{id}' - DeleteRouteSchedulingTemplate(Guid)
        - 'api/RouteSchedulingTemplate/by-route/{id}' - DeleteRouteSchedulingTemplatesForRoute(Guid)
- RouteTicket
    - [POST]
        - 'api/RouteTicket' - CreateRouteTicket(RouteTicketCreateCommand)

    - [DELETE]
        - 'api/RouteTicket/{id}' - DeleteRouteTicket(Guid)
- ScheduledRoute
    - [POST]
        - 'api/ScheduledRoute' - CreateScheduledRoute(ScheduledRouteCreateCommand)
    - [PUT]
        - 'api/ScheduledRoute' - EditScheduledRoute(ScheduledRouteUpdateCommand)
    - [DELETE]
        - 'api/ScheduledRoute/{id}' - DeleteScheduledRoute(Guid)
- Seat
    - [POST]
        - 'api/Seat' - CreateSeat(SeatCreateCommand)
        - 'api/Seat/reserve' - ReserveSeat(SeatReservationCommand)
        - 'api/Seat/free' - FreeSeat(SeatReservationCommand)
    - [PUT]
        - 'api/Seat' - EditSeat(SeatUpdateCommand)
    - [DELETE]
        - 'api/Seat/{id}' - DeleteSeat(Guid)
- Service
    - [POST]
        - 'api/Service' - CreateService(ServiceCreateCommand)
    - [PUT]
        - 'api/Service' - EditService(ServiceUpdateCommand)
    - [DELETE]
        - 'api/Service/{id}' - DeleteService(Guid)
- Stop
    - [POST]
        - 'api/Stop' - CreateStop(StopCreateCommand)
        - 'api/Stop/batch' - CreateStops(StopBatchCreateCommand)
        - 'api/Stop/add-to-location' - AddStopToLocaiton(StopLocationCommand)
    - [PUT]
        - 'api/Stop' - EditStop(StopUpdateCommand)
    - [DELETE]
        - 'api/Stop/{id}' - DeleteStop(Guid)
        - 'api/Stop/remove-from-location' - RemoveStopFromLocaiton(StopLocationCommand)
- User
    - [POST]
        - 'api/User' - CreateUser(UserCreateCommand)

        - 'api/User/restore' - RestoreUser(Guid)
        - 'api/User/like-connection' - LikeConnection(UserLikeConnectionCommand)
        - 'api/User/like-location' - LikeLocation(UserLikeLocationCommand)
    - [PUT]
        - 'api/User' - EditUser(UserUpdateCommand)
    - [DELETE]
        - 'api/User/{id}' - DeleteUser(Guid)
        - 'api/User/dislike-connection' - DislikeConnection(UserLikeConnectionCommand)
        - 'api/User/dislike-location' - DislikeLocation(UserLikeLocationCommand)
- Vehicle
    - [POST]
        - 'api/Vehicle' - CreateVehicle(VehicleCreateCommand)
        - 'api/Vehicle/replace' - ReplaceVehicleOnScheduledRoutes(VehicleReplaceOnScheduledCommand)
        - 'api/Vehicle/AddVehicleRTI' - AddVehicleRTI(VehicleRTIDto)
    - [PUT]
        - 'api/Vehicle' - EditVehicle(VehicleUpdateCommand)
    - [DELETE]
        - 'api/Vehicle/{id}' - DeleteVehicle(Guid)

4.3.2. Queries

- Icon
    - [POST]
        - 'api/Icon' - GetFilteredSoT(IconFilteredQuery)
- Line
    - [POST]
        - 'api/Line/filter' - GetLinesFilteredSoT(LineFilteredQuery)
- Location
    - [GET]
        - 'api/Location/GetById/{id}' - GetById(Guid)
        - 'api/Location/GetByName' - GetByName(string)
        - 'api/Location/GetClosest' - GetClosestLocation(double, double)
    - [POST]

        - 'api/Location/filter' - GetLocationsFilteredSoT(LocationFilteredQuery)
- Route
    - [POST]
        - 'api/Route' - GetRoutesFilteredSoT(RouteFilteredQuery)
- RouteSchedulingTemplate
    - [GET]
        - 'api/RouteSchedulingTemplate/{id}' - GetRouteSchedulingTemplate(Guid)
    - [POST]
        - 'api/RouteSchedulingTemplate/filter' - GetRouteSchedulingTemplatesSoT(RouteSchedulingTemplateFilteredQuery)
- RouteTicket
    - [POST]
        - 'api/RouteTicket/filter' - GetRouteTicketsFiltered(RouteTicketFilteredQuery)
- ScheduledRoute
    - [GET]
        - 'api/ScheduledRoute/GetScheduledRoutes' - GetScheduledRoutes(Guid, Guid, DateTime?)
    - [POST]
        - 'api/ScheduledRoute/filter' - GetScheduledRoutesFilteredSoT(ScheduledRouteFilteredQuery filter)
- Seat
    - [GET]
        - 'api/Seat/route/{scheduledRouteId}/seats' - GetSeatsForScheduledRouteWithState(Guid)
    - [POST]
        - 'api/Seat/filter' - GetSeatsFilteredSoT(SeatFilteredQuery)
- Service
    - [POST]
        - 'api/Service' - GetFilteredSoT(ServiceFilteredQuery)
- Stop
    - [POST]
        - 'api/Stop' - GetFilteredStopsSoT(StopFilteredQuery)
- User
    - [GET]
        - 'api/User' - GetAllSoT()
    - [POST]

        - `'api/User/login' - LoginUser(LoginDto)`
        - `'api/User/logout' - LogoutUser()`
- Vehicle
    - [GET]
        - `'api/Vehicle' - GetVehicles()`
        - `'api/Vehicle/{id}' - GetVehicle(Guid)`
        - `'api/Vehicle/GetRTIByVehicleId/{id}' - GetRTIByVehicleId(Guid)`

SoT - je zkratka pro “Source of truth”, v našem případě tedy relační DB, nad kterou byl query dotaz proveden