BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Ukládání a příprava dat

Projekt 1.časť

25. října 2022

Team xziska03

Žiška Marek xziska03
Osvald Martin xosval03
Daša Nosková xnosko05

Obsah

I Analýza zdrojových dat a návrh jejich uložení v NoSQL databázi **2**

1 Analýza zdrojových dat **3**
1.1 Typy správ . . . 3
1.1.1 CZPTTCISMessage . . . 3
1.1.2 CZCanceledPTTMessage . . . 5

2 Návrh způsobu uložení dat **6**
2.1 Získavanie dát . . . 6
2.2 Spracovanie dát . . . 7
2.3 Štruktúra uloženia dát . . . 7
2.3.1 Alternatívy . . . 9

3 Zvolená NoSQL databáza **10**

II Návrh, implementácia a použitie aplikácie **11**

4 Návrh aplikácie **12**
4.1 Data . . . 12
4.1.1 Sťahovanie dát . . . 12
4.1.2 Ukladanie dát . . . 13
4.1.3 Vyhľadávanie dát . . . 13
4.2 Server . . . 14
4.3 Client . . . 14
4.3.1 Vizuálna stránka . . . 14
4.3.2 Komunikácia so serverom . . . 15
4.4 Dockerizácia . . . 15

5 Způsob použití **17**
5.1 Spôsob spustenia projektu . . . 17

6 Experimenty **18**
6.1 Časová náročnosť stiahnutia a spracovania dát . . . 18

Část I

Analýza zdrojových dat a návrh jejich uložení v NoSQL databázi

Kapitola 1

Analýza zdrojových dat

Potrebné dátové sady pre verejnosť sa nachádzajú na stránke FTP CIS. Dátová sada GVD2022.zip obsahuje súbory vo formáte `xml` popisujúce jednotlivé vlaky a ich trasy. Je definovaná elementom `CZPTTCISMessage`. V zložkách označujúcich mesiace sa nachádzajú informácie o odrieknutých, popr. odklonených vlakoch v daných dňoch. Súbory označené prefixom `cancel` a začínajúce elementom `CZCanceledPTTMessage` informujú o zrušených vlakoch. Vo zvyšných súboroch obsahujúcich v názve `KADR` sa nadchádzajú informácie o plánovaných odklonoch vlakov.

Správy CZPTT majú unikátne **PA ID** a sú prepájané zhodným **TR ID**, pretože jeden vlak môže mať viac správ.

Správy o zrušení a vytvorení novej trasy sú pridávané do systému za sebou a správy sa pridávajú počas celého dňa.

Vlaky je možné zoskupiť podľa druhu vlaku alebo liniek alebo na jazdiace, zrušené a odklonené, popr. nahradené a taktiež podľa mesiacov. Veľkosť všetkých potrebných zdrojových dát sa pohybuje dokopy okolo 2GB.

1.1 Typy správ

1.1.1 CZPTTCISMessage

- `Identifiers` - (povinný, element) obsahuje dva elementy `PlannedTransportIdentifiers` identifikujúce vlak a trasu. V prípade odkloneného vlaku obsahuje aj `RelatedPlannedTransportIdentifiers`.
  - `PlannedTransportIdentifiers` (povinný, element)
    - `ObjectType` (povinný, string)
      - **TR** identifikuje vlak
      - **PA** identifikuje trasu
    - ďalšie detské uzly dok. str. 7
  - `RelatedPlannedTransportIdentifiers` - **IBA V PRÍPADE AK SPRÁVA OZNAČUJE ODKLON VLAKU** (povinný, element) označuje pôvodný plánovaný vlak s PAID totožným s pôvodným vlakom.
- `CZPTTCreation` (povinný, datetime) určuje čas vytvorenia CZPTT, rozhoduje, ktorý CZPTT je aktuálny k danému vlaku.
- `CZPTTHeader` - iba pri medzištátnych vlakoch, detské elementy viď dok. str. 8
- `CZPTTInformation` (povinný, element) údaje o trase, obsahuje aspoň dva elementy `CZPTTLocation`
  - `CZPTTLocation` - (povinný, element) zložený z elementov `Location`, ktorý má detské uzly:
    - `CountryCodeISO` (povinný, string) skratka krajiny
    - `LocationPrimaryCode`(povinný, číslo) kód miesta
    - `PrimaryLocationName` (nepovinný, string) názov miesta
    - `LocationSubsidiaryIdentification` (element, nepovinný)

  - `TimingAtLocation`
    - `Timing` určuje časy pre príjezdy a prejazdy lokalitou (nepovinný, element)
      - `Time` (povinný, time hh:mm:ss + čas. zóna)
      - `Offset`(povinný, číslo) - počet prechodov cez polnoc
      - `TimmingQualifierCode`(povinný, string) určuje, čo popisuje element `Time`
        1. **ALA** - príchod
        2. **ALD** - odchod
  - `TrainType` (nepovinný, číslo) kategória vlaku.
    - **1** = verejný osobný vlak
  - `TrainActivity` (nepovinný, element) - existuje, ak v lokalite bude prebiehať aktivita.
    - `TrainActivityType` (povinný, číslo)
      - **1** - nástup a výstup cestujúcich
  - `OperationalTrainNumber` (nepovinný, číslo) číslo vlaku/trasy `NetworkSpecificParameter` (nepovinný, element) obsahuje národné parametre zložené z elementu name, popisujúceho typ a hodnoty k danému typu napr:
    `Name` = CZAlternativeTransport
    `Value` = 1 popisuje náhradnú dopravu.
    ostatné viď dok. príloha 8.1.1

- ostatné elementy viď dok. str. 11

- `PlannedCalendar` (povinný, element) pre element `CZPTTInformation` určuje prvý bod trasy v ČR. Pre element `CZCanceledPTTMessage` určuje dni, kedy je vlak odrieknutý.
  - `BitmapDays` (povinný, bin. číslo) Podle jednotlivých dnov jazdy uvedených v bitovej mape sa počíta denný tvar identifikátoru PA ID. Pokud je třeba zjistit konkrétní den jízdy v konkrétním bodě, pak se musí vzít den z kalendáře a k němu připočítat hodnotu z elementu Offset v tomto bodě.
    - **1** – Vlak je v daný den řešen zprávou, která tento kalendář obsahuje. Takže je jedoucí pre správu `CZPTTInformation` nebo odřeknutý pre správu `CZCanceledPTTMessage`.
    - **0** – Vlak je v daný den neřešen zprávou, která tento kalendář obsahuje. Takže je buďto nejedoucí pre správu `CZPTTInformation` alebo neodřeknutý pre správu `CZCanceledPTTMessage`.
  - `ValidityPeriod` (povinný, element) určuje obdobie platnosti správy
    - `StartDayTime` (povinný, datetime) počiatočný deň platnosti
    - `EndDayTime` (povinný, datetime) posledný deň platnosti
  - `NetworkSpecificParameter` (nepovinný, element) viď dok. str. 13 a príloha 8.1.2

1.1.2 CZCanceledPTTMessage

Vlaky môžu byť odriekané dvomi spôsobmi, viď dok. str. 14.

- `PlannedTransportIdentifiers` (povinný, element) identifikuje identifikátormi vlak a dílčí cestný řád (trasa), ktorý bol odrieknutý.
- `CZPTTCancelation` (povinný, datetime) dátum a čas odrieknutia. Rovnaká dvojica TRID a PAID môžu byť odrieknuté viackrát. `PlannedCalendar` (povinný, element) obsahuje dni, kedy je vlak odrieknutý, pre elementy viď 1.1.1.

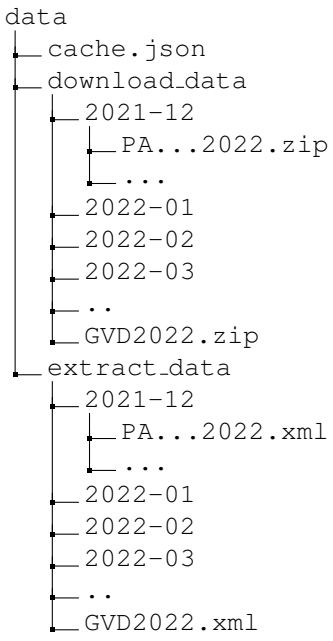
Kapitola 2

Návrh způsobu uložení dat

2.1 Získavanie dát

Pre prehľadné oddelenie od ostatných častí implementovanej aplikácie budú všetky spracované a stiahnuté dáta spolu s ich skriptami ukladané do samostatnej zložky `data` v koreňovom adresári projektu.

- Z hľadiska automatizácie sťahovania súborov, dátový portál jízdních řádů obsahuje dva typy súborov. Jeden typ je špecifický pre jeden súbor a to `GVD2022.zip`. Tento súbor je individuálny a pre jeho stiahnutie vystačí link samotný. V prípade súborov uložených v jednotlivých mesiacov by podobný prístup pre ich vysoký počet nebol vhodný. Ako riešenie sa javí zautomatizovanie spracovania webovej stránky, ktorá predstavuje jeden z mesiacov na portály. Na tento účel je vhodná knižnica *beautifulsoup4*, ktorá nám umožní získať zoznam všetkých dostupných linkov k *zip* súborom na danej stránke pre daný mesiac. Ukladanie stiahnutých súborov uložíme do samostatnej zložky `download_data`, ktorá bude obsahovať podzložky reprezentujúce určitý mesiac v roku. Obsah podzložiek bude už predom spomínaný zoznam stiahnutých *zip* súborov.
- Jednotlivé súbory uložené v podzložkách v `download_data` budú jednotlivo extrahované pomocou *gzip*, popr. *zip* a postupne spracované do XML súborov. XML súbory budú uložené v rovnakej štruktúre zložiek tak ako uložené *zip* súbory, tentoraz ale v zložke `extract_data`.
- Keďže súbory na portály môžu byť postupne časom priebežne pridávané, je vhodné zaistiť aby nedochádzalo k repetitívnemu sťahovaniu a spracovaniu súborov. Na toto sme navrhli využitie `cache.json` súboru, ktorý bude ukladať cestu k už stiahnutým a taktiež aj spracovaným súborom. Tento *json* súbor bude využívaný pri následnom sťahovaní a spracovaní dostupných súborov na webovom portáli, takým spôsobom že súbor už uložený v `cache.json` nebude znova sťahovaný.



Obrázek 2.1: Štruktúra uloženia získaných dát

2.2 Spracovanie dát

- Vlaky sú jednoznačne identifikované pomocou TRID a PAID.
- Je nutné porovnávať, čas vytvorenia správ, pretože niektoré správy sa odkazujú na rovnaký vlak.
- Pri hľadaní vlakov, je potrebné kontrolovať bitovú mapu. Plánovaný vlak musí obsahovať na indexe bitovej mapy 1. Vlaky odklonené alebo zrušené musia obsahovať na bitovom indexe 0, aby plánovaný vlak v daný deň išiel.
  - Bitový index sa vypočíta ako rozdiel začiatku kalendára a hľadaného dátumu.
- Pri spracovávaní dát je nutné rozlišovať typ správy a to podľa:
  - Typu xml root elementu pre rozdelenie na zrušené a ostatné
  - Podľa obsahu elementu Identifiers - ak je prítomný element RelativePlannedTransportIdentifiers tak ide o vlak odklonený, inak o vlak plánovaný.
- Niektoré dáta sú pre návrh aplikácie nepodstatné napr. (`NetworkSpecificParameters`, a stačí aby boli ukladané jednotlivé elementy aj s detskými elemtami ako nerozparsovanéa ako binárne.
- Časové údaje a údaje obsahujúce dátum musia byť prekonvertované na DateTime object, pre jednoduché porovnávanie dátumov.

2.3 Štruktúra uloženia dát

Dáta budú uložené do 3 trvalých kolekcií a to `plannedTrains`, `rerouteTrains`, obidva so štruktúrou 2.1. `RerouteTrains` má pridanú časť obsahujúcu info o pôvodnom vlaku, ktorého sa správa týka uložené v atribúte 2.2. Tretia kolekcia `canceledTrains` so štruktúrou 2.3. Pomocné dočasné štruktúry budú `canceledTrainsInDate`, `rerouteTrainsInDate`, `validTrains`, `goingToLocation`.

Všetky objekty v kolekciách sú identifikované jednoznačne pomocou `_id` a pomocou `TR.ID`, podľa ktorého je vytvorený aj index.

```
{
    _id: ObjectId(),
    "PA" : {
                "ID": String
                "Company" : String,
                "Variant" : Integer,
                "Year" : Integer
        },
    "TR" : {
                "ID" : String
                "Company" : String,
                "Variant" : Integer,
                "Year" : Integer
        },
    "created" : ISODate("2022-09-26T09:48:05Z"),
    "path": [{
            "country": String,
            "locationCode": Integer,
            "name": String,
            "subsidiaryIdentification": BinData, #unparsed
            "departure": {
                "time": DateTime
            },
            "Responsible": {
                "responsibleRU": Integer,
                "responsibleIM": Integer
            },
            "trainType": Integer,
            "traffic": {
                "trafficType": String,
                "commercialTrafficType": String
            },
            "trainActivity": [
                String,
                ...
            ],
            "operationalTrainNumber": Integer,
            "networkSpecificParameters": [binData] # unparsed,
        }, .... ],
    "calendar" : {
                "bitmap" : String,
                "startDate" : DateTime,
                "endDate" : DateTime
        },
    header: Array #unparsed
    NetworkSpecificParameters: [binData] # unparsed,
}
```

Kód 2.1: Návrh štruktúry pre kolekciu plannedTrains, ktorá bude obsahovať informácie o vlakoch. Objekty LocationSubsidiaryIdentification, OtherTimming, TimingAtLocationOther, NetworkSpecificParameters budú obsahovať nerozparsované, zatiaľ nepotrebné informácie.

```
    "plannedPAID" : {
            "ID" : String,
            "Company" : String,
            "Variant" : Integer,
            "Year" : Integer
    },
```

Kód 2.2: Dodatočný objekt v štruktúre 2.1 pre kolekciu rerouteTrains

```
    "_id" : ObjectId,
    "PAID" : String,
    "PA" : {
            "Company" : String,
            "Variant" : Integer,
            "Year" : Integer
    },
    "TRID" : "KASO---1983A",
    "TR" : {
            "Company" : String,
            "Variant" : Integer,
            "Year" : Integer
    },
    "created" : DateTime,
    "calendar" : {
            "bitmap" : String,
            "startDate" : DateTime,
            "endDate" : DateTime
    }
}
```

Kód 2.3: Kolekcia canceledTrains

2.3.1 Alternatívy

1. Alternatívnym spôsobom uloženia by mohlo byť vytvorenie kolekcie Calendars, v ktorej by boli uložené všetky kalendáre, ktoré by odkazovali na objekt vlaku, ktorého sa kalendár týka. Kalendár by obsahoval informáciu o type správy.
2. Ďalšou možnosťou by bolo vytvorenie kolekcie Locations, ktorá by obsahovala názov lokácie a list odchodov a príchodov jednotlivých vlakov.

Kapitola 3

Zvolená NoSQL databáza

Je vhodná dokumentová databáza, pretože sa pracuje s dátami XML, ktorá umožňuje ukladanie zložených objektov a rôznych typov a poskytuje možnosť tvorby nových kolekcií a zmenu dát. Dokumentová databáza ukladá dáta čitateľným spôsobom.

Pre projekt bola vybratá databáza `MongoDB`, ktorá ukladá dokumenty do kolekcií a poskytuje flexibilný spôsob ukladania dát, pričom jej nevadí redundancia a ani neštrukturované dáta. Pre prácu s MongoDB nie je nutná normalizácia dát, a preto je práca s jednotlivými prvkami v kolekcii rýchle. Objekty v kolekcii sa dajú jednoducho filtrovať, spájať s inými a transformovať výsledok na požadovaný tvar, pre ďalšiu prácu. Na prezeranie dát pri vývovji sme využíval MongoDB Compas.

Časť II

Návrh, implementácia a použitie aplikácie

Kapitola 4

Návrh aplikácie

Aplikácia je rozdelená do 3 hlavných častí: `data`, `client`, `server`. Časť `data` a `server` je implementovaná v jazyku *Python* a client v jazyku *React*.

4.1 Data

Orchestrácia práce s dátami je daná skriptom `main.py` v zložke `data`. Prvým krokom celej aplikácie je stiahnutie požadovaných dát pomocou knižnice *Beautiful Soup* do zložky `data/download_data`, ich extrakcia z komprimovaných súborov do formátu xml cez nástroj *gzip* popr. *zip* do zložky `data/extract_data`.

4.1.1 Sťahovanie dát

Za účelom jednoduchého aktualizovania a vyhľadania statických informácií ako URL adresa portálu, cesta k datovým súborom jízdních řádů, sme vytvorili separátny súbor `constants.py`, ktorý tieto informácie združuje pre ostatné skripty.
Sťahovanie dát je zhrnuté v súbore `download.py` v zložke `downloader`. Súbor obsahuje radu funkcí obsluhujúcich parsovanie webovej stránky datového portálu za účelom nájdenie dátových súborov a ich následné stiahnutie a uloženie. Pre stiahnutie súboru `GVD2022.zip` slúži funkcia `downloadGVZIP`, ktorá explicitne špecifikovanou cestou súbor stiahne.
Ostatné súbory v zložkách pre každý mesiac nie je možné podobným spôsobom stiahnuť, pre ich veľký počet. Tento zoznam súborov je spracovaný vo funkcií `downloadAllZip`, kde pomocou knižnice *beautifulsoup4*, ktorá poskytuje jednoduché a pritom efektívne rozhranie na "web scrapping". Po analýze zdrojového kódu webovej stránky portálu, sme zistili že požadované súbory sú referencované v *HTML* odkazoch. Funkcia `find_all` z *beautifulsoup4* jednotlivé tagy vyhľadá a uloží do zoznamu objektov `PageElements`, v ktorých vieme jednoducho pristupovať k `href` argumentu *HTML* odkazom a tie následne stiahnuť pomocou funkcie `wget`.
Pre optimálnejšie stiahnutie súborov sme naimplemetovali sťahovanie pomocou viacerých jadier v zariadení, čo nám umožnilo násobne znížiť čas potrebný na stiahnutie všetkých súborov z webového portálu. Toto bolo docielené pomocou knižnice `multiprocessing` a jej triedy `Pool`, ktorá umožňuje asynchrónne vykonávanie funkcií.

Cache na sťahovanie a spracovanie

Ďalšou optimalizácou, ktorú sme sa v rámci sťahovania rozhodli vykonávať je priebežné ukladanie zoznamu súborov, ktoré už boli stiahnuté a spracované. Na ukladanie slúži `cache.json` súbor, ktorý obsahuje dva zoznamy:

- `downloaded` - obsahuje zoznam názvov zip súborov, ktoré už boli stiahnuté a uložené.
- `extract` - obsahuje zoznam ciest k vygenerovaným xml súborov, ktoré boli spracované zo stiahnutých súborov.

V kóde následne po počiatočnom stiahnutí a spracovaní súborov, uložíme ich zoznam. Pri nasledujúcej iniciácí sťahovanie súborov, sa tento zoznam využije aby nedochádzalo k duplicitnému spracovaniu/stiahnutiu súborov.

4.1.2 Ukladanie dát

Po stiahnutí dát nasleduje rozparsovanie dát pomocou knižnice *lxml* a príprava dát do Python štruktúry `Dictionary` pre vloženie do správnej kolekcie podľa typu správy. Aby sa nevkladali do databázy duplicitné záznamy, využíva sa pre vkladanie funkcia od *PyMongo* `replace_one()` s parametrom `upsert=True`, pričom sa kontroluje zhodnosť TRID a PAID. Offset pri jednotlivých lokáciach trasy sa pripočíta k datetime objektu, v ktorom je uložený príchod alebo odchod zo stanice. Pričom offset=počet dní pripočítaných k datetime objektu. Je to vhodné riešenie, pretože jednotlivé odchody a príchody vlaku do lokality sú vyjadrené iba časovým údajom.

4.1.3 Vyhľadávanie dát

Hlavnou časťou je trieda `Queries.py`, ktorej funkcie sú vkladanie dát do databázy a vyhľadávať požadované dáta. Nižšie opísaný spôsob vyhľadávania vlakov bol rozdelený do menších častí, pre možné budúce rozšírenie aplikácie o vyhľadávanie napr. iba zrušených alebo odklonených vlakov, popr. rozšírenie vyhľadávania vlakov idúcich z východzej stanice do cieľovej o vlaky aj odklonené.

Vyhľadávanie vlakov idúcich v daný deň z východzej stanice (`odkiaľ`) do cieľovej (`kam`) prebieha následovne:

- Vyfiltrujú sa zrušené a odklonené vlaky v daný deň pomocou agregácie.
  1. Vypočíta sa bitový index ako rozdiel počiatočného dátumu kalendáru a hľadaného dňa.
  2. Cez bitový index sa nájde v bitovej mape hľadaný bit.
  3. Cez logický súčin $\$and$ sa nájdu záznamy, pre ktoré kalendáre majú počiatočný dátum $\geq$ ako hľadaný dátum a zároveň konečný dátum platnosti $\leq$ ako hľadaný dátum. A zároveň bit = 1.
  4. Nájdené objekty sa zoradia podľa atribútu `created` v zostupnom poradí.
  5. Objekty sa zoskupia podľa `TR.ID` a vyberie sa iba najnovšia správa.
  6. Vráti sa objekt iba s položkami `TRID`, `PAID` a `path`.
- Vyfiltrované vlaky sa uložia do pomocných kolekcií `self.canceled_trains_in_date` a `self.reroute_trains_in_date`.
- Nasleduje výber platných vlakov pomcou agregácie.
  1. Kroky 1-3. totožné s hľadaním zrušených/odklonených vlakov.
  2. Cez funkciu $\$lookup$ sa vytvorí left outer join cez hodnotu `TRID` pre pomocné kolekcie `self.canceled_trains_in_date` a `self.reroute_trains_in_date`.
  3. Nakoniec sa vyberú iba objekty, ktoré majú prázny prienik pre zrušené aj odklonené vlaky.
- Idúce vlaky v daný deň sa uložia do pomocnej kolekcie `self.valid_trains` a predajú sa pre filtráciu na základe trasy, ktorá postupuje následovne:
  1. Nájdu sa iba vlaky, ktorých element `path.name` obsahuje zároveň miesto `odkiaľ` a element `path.trainActivity` = 0001 a zároveň `path.name` = `kam` a `path.trainActivity` = 0001
  2. Vypočítajú sa indexy na ktorých sa nachádzajú lokácie `kam` a `odkiaľ` v zozname elementu `path`.
  3. Vyberú sa iba vlaky, kde index miesta `kam` je väčší ako index miesta `odkiaľ`.
- Vybrané vlaky sa uložia do pomocnej kolekcie `trains_going_to_location`.
- Nakoniec sa kolekcia `trains_going_to_location` upraví pre výpis, iba na parametre obsahujúce prvky:
  - TRID
  - PAID
  - názov stanice, v ktorej vlak zastavuje - obsahuje kód 0001, príchod do stanice a odchod zo stanice.

4.2 Server

Server je naimplementovaný ako jednoduchý lokálny vývojový server, pomocou knižnice *Flask*. Táto knižnica nám umožňuje špecifikovať RESTful API, skladajúce sa z "endpointov"komunikujúcich s našou MongoDB databázou. Jednotlivé "endpointy"sú deklarované funkciami so špeciálnymi dekorátormi z *Flask* knižnice. Táto funkcia má za sebou istú požadovanú logiku, ktorá prípadne pracuje s databázou a ako výsledok vráti *json* súbor s určitými dátami. Tieto dáta môžu byť prevzané v našej webovej aplikácií a zobrazené užívateľovi. Prípadne užívateľ môže poslať určité dáta, ktoré server použije za určitým účelom(napríklad filtrovanie záznamov) a opäť vráti odpoveď. V rámci našej aplikácie sme potrebovali tri rôzne "endpointy":

- Dekorátor `@application.route('/')` reprezentuje endpoint pre domovskú stránku. Slúži ako test či funguje server.
- Dekorátor `@application.route('/locations')` reprezentuje endpoint, ktorý je volaný ako prvý pri načítani clienta. Vráti zoznam všetkých vlakových staníc. Tento zoznam je následne používany pri filtrovani SearchFieldu v inpute v React aplikácii. Tým podľa definície bez dodatočného payloudu stačí GET.
- Dekorátor `@application.route('/paths')` vyhladá všetky spojenia medzi zadanými stanicami na **celý** deň.
- Dekorátor `@@cross_origin()` je použitý na lokálny vývoj aby clientská aplikácia nemala problem s CORS policy.

4.3 Client

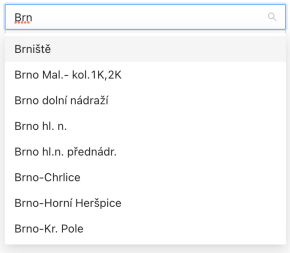
Klientskú aplikáciu sme naimplementovali v *React JS* vo forme jednoduchej webovej stránky s využitím knižnice *ANT Design*, ktorá obsahuje radu komponent. Rozhodli sme sa tak z dôvodu, že webová stránka predstavuje užívateľsky priateľskejší a prehľadnejší spôsob práce s aplikáciou, ako by to bolo v prípade konzolovej aplikácie.

4.3.1 Vizuálna stránka

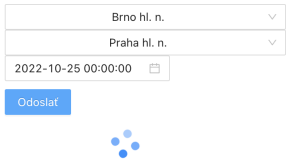
Webová stránka vizualizuje jednotlivé spoje zo spracovaných jízdních řádů vo forme jeden záznam jedna tabuľka. Na tabuľku bola využitá komponenta `Table` z *ANT Design*, ktorá dokáže dynamicky zobrazovať variabilný počet stanic. Tabuľka je plne modifikovateľná, takže bolo jednoduché zvoliť jednotlivé stĺpce, ktoré chceme zobrazovať. V rámci jednej tabuľky a teda jedneho spoja zobrazujeme **príchod odchod** a **stanicu**.

Mimo tabuľky stránka obsahuje dve textové vstupné polia a jedno vstupné pole pre voľbu dátumu a času. Textové polia slúžia pre špecifikáciu názvu počiatočnej a cieľovej stanice. Obsah textového vstupu sa ukladá do stavových premenných. Pričom tieto textové polia podporujú select funkciu zo všetkých dostupných staníc z dát.

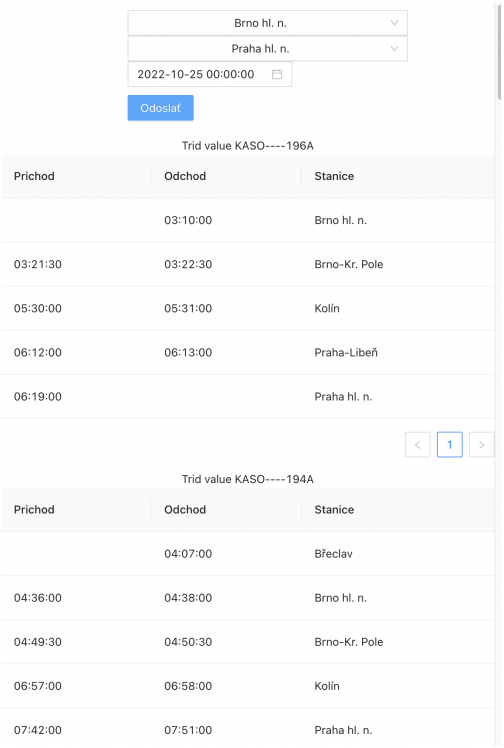
Vstup pre dátum a čas je implementovaný pomocou `DatePicker` komponenty a jej hodnota sa ukladá podobným princípom ako u vstupných poliach, len s iným formátom.



Obrázek 4.1: Našepkávanie pre výber existujúcej stanice



Obrázek 4.2: Vyhladávanie má loader dokým nepríde odpoveď od flask serveru.



Obrázek 4.3: Výsledky po vyhladaní spojov. Na obrázku je sú dve tabuľky, kde sú dva spoje.

4.3.2 Komunikácia so serverom

V zložke `service/api` sú definované všetky API-calls cez knižnicu `axios`. Využívaju dva endpointy a to `/locations` `/paths` z flask aplikácie.

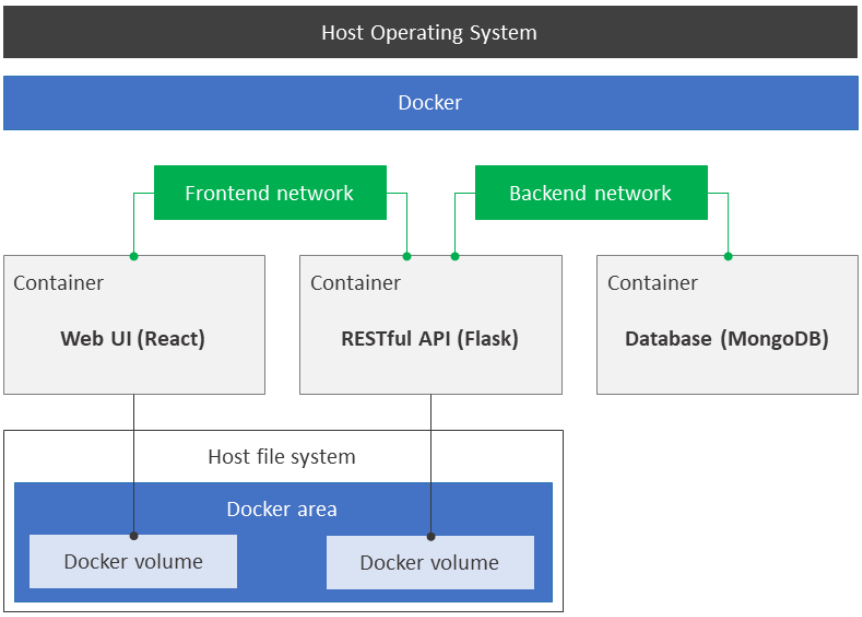
4.4 Dockerizácia

Pri implementácií jednotlivých častí projektu sme sa stretli s problémami u určitých verzí použitých balíčkov a knižníc. Preto sme sa rozhodli jednotlivé časti našej aplikácie kontajnerizovať pomocou platformy *Docker*. Týmto pádom sme sa vyhli aj prípadným problémom pri spojazdnení projektu pri vyhodnotení. Keďže naša aplikácia sa skladá z troch častí, vytvorili sme tri kontajnery:

- Kontajner pre našu databázu v MongoDB. `http://localhost:27017/`
- Kontajner pre náš Flask API server. `http://localhost:5001/`
- Kontajner pre našu klientsku webovú aplikáciu. `http://localhost:3000/`

Kontajnery sme pripojili dvoma sieťami(viz Obrázek. 4.4). Prvou sieťou sme prepojili kontajneri našej databáze a serveru(RESTful API), čím sme znížili risk neautorizovaného prístupu do databáze. Druhá sieť sa stará o komunikáciu

medzi našim kontajnerom webovej aplikácie a kontajneru s Flask API. Za zmienku sa taktiež oplatí zmieniť, že rovno v `docker-compose.yml` sa pri inicializácii MongoDB sa nastaví aj root účet a aj root heslo k prístupu ku db.



Obrázek 4.4: Štruktúra našej kontajnerizovanej aplikácie.

Kapitola 5

Způsob použití

5.1 Spôsob spustenia projektu

Pre spustenie projektu je potrebné aby bolo na systéme nainštalovaný .

1. `make`
2. `docker`
3. `docker-compose`
4. `python3.10` alebo aj `python3.9` s podporou venv. Slúži pre sťahovanie súborov a není súčastou docker-compose.

Projekt stači rozbaliť alebo stiahnut cez git clone z adresy `https://github.com/Mylanos/UPA2022.git`

1. `make start` inicializuje cely docker environment.
2. `pip install virtualenv` ak neni naištalovaný virtualenv
3. `virtualenv venv` vytvorenie virtuálneho environment-u .
4. `source venv/bin/activate` aktivovanie venv.
5. `pip install -r requirements.txt`
6. `cd data`
7. `python3 main.py` stiahne dáta a insertuje do databáze.

Nasledne si može pozrieť webovú aplikáciu na `http://localhost:3000/`

Kapitola 6

Experimenty

6.1 Časová náročnosť stiahnutia a spracovania dát

Meranie času behu programu na spracovanie a stiahnutie dát bolo vykonané na počítači skladajúci sa z CPU s 6 jadrami a 12 vláknami na frekvencii 3.600GHz. Čo stojí za poznamenanie je čas potrebný na stiahnutie, pred zavedením multiprocesovej obsluhy stiahnutia sa čas potrebný na stiahnutie všetkých súborov pohyboval v desiatkach minút. Po zavedení sme dosiahli násobné zrýchlenie a stiahnutie na referenčnom stroji trvalo len 43 sekúnd.

```
$ python3 -m timeit data/main.py
    Time it takes to initialize DB and Queries objects: 0.089295 seconds.
    Time it takes to download all the files: 42.611331 seconds.
    Time it takes to process all the downloaded files: 6.885434 seconds.
    Time it takes to insert all the data into the database: 483.820754 seconds.
```

Obrázek 6.1: Výstup z časovaného behu skriptu na spracovanie dát.