VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

DUOMENŲ MOKSLO PROGRAMA

Kelionių registracijos sistema

Travel registration system

Duomenų bazės modeliavimas

Autorius: Evelina Vaitkevičiūtė
VU el. p.: evelina.vaitkeviciute@mif.vu.lt

Darbo vadovas:

Vilnius
2024

1 Įvadas

Kiekvienais metais kompiuterinių programų surenkamas įvairių duomenų, naudojamų mokslinėse, verslo, transporto valdymo ir kitose srityse kiekis auga. Juos išsaugoti ir efektyviai jais operuoti tampa vis sunkiau, todėl atsiranda poreikis naudoti nereliacines duomenų bazes (NoSQL).

Viena iš tokių sričių – transporto valdymas, kelionių registracijos sistemos. Tradiciškai tokių sistemų kūrimui buvo naudojamos reliacinės duomenų bazės, tačiau dėl didėjančio keliautojų susidomėjimo ir naudojimosi navigacijos įrenginiais, didėjančio duomenų kiekio, kelionių registracijos sistemos vis dažniau susiduria su iššūkiais. Su dideliais kiekiais duomenų realiacinės duomenų bazės tiesiog tampa ne efektyvios ir per lėtos. Taip pat, realiacinėse duomenų bazėse yra sudėtinga saugoti nestruktūrizuotus duomenų tipus.

Minėtas problemas išspręsti ir užtikrinti didesnį sistemos našumą, išplečiamumą bei lankstesnį duomenų modeliavimą gali nereliacinės duomenų bazės. Visgi, norint tinkamai pasirinkti, kokio tipo nereliacinę duomenų bazę panaudoti savo projekte, derėtų įvertinti dalykinės srities keliamus reikalavimus ir savarankiškai išbandyti turimą geriausią technologinį sprendimą bei realizuoti pasirinkimus taikymo sąlygos.

Pagrindinis šio darbo tikslas – pagal pateiktus reikalavimus, pasirinkti tinkamą duomenų bazę kelionių registracijos sistemos realizavimui, suprojektuoti bei realizuoti duomenų modelį.

Tikslui pasiekti darbe buvo sprendžiami šie uždaviniai:

1. Įvertinti kelių žinomų nereliacinių duomenų bazių tinkamumą užduočiai ir pasirinkti tinkamiausią duomenų bazę užduoties realizavimui.
2. Suprojektuoti duomenų modelį, pateikti fizinio duomenų modelio diagramą.
3. Sukurti servisą, leidžiantį navigacijos įrenginiams registruoti keliones.

Darbą sudaro įvadas, du skyriai ir baigiamoji dalis, kurioje pateiktos darbo išvados, taip pat naudotos literatūros sąrašas, priedai. Antrajame skyriuje atliekama nereliacinių duomenų bazių analizė. Trečiasis skyrius skirtas fizinio duomenų modelio pateikimui ir aprašymui.

2 Nereliacinių duomenų bazių palyginimas

Šiame skyriuje bus atliekama nereliacinių duomenų bazių palyginamoji analizė ir taip siekiama išsiaiškinti, kuri NoSQL duomenų bazė efektyviausia realizuojant kelionių registracijos sistemą.

Kadangi darome prielaidą, kad kelionių registracijos sistema gali sulaukti daug vartotojų, todėl duomenų bazė turi būti pajėgi apdoroti didelius duomenų kiekius. Duomenų bazė turi palaikyti įvairius duomenų tipus, įskaitant tekstą, skaičius, datą ir laiką bei koordinates.

Buvo pasirinkta analizuoti Redis, Apache Cassandra, MongoDB, Neo4j duomenų bazes. Duomenų bazių palyginimui buvo atlikta duomenų bazių analizė (1 lentelė), o platesniam tyrimui atlikti yra naudojami analizės metu padaryti pastebėjimai ir išvados. NoSQL duomenų bazes prasminga lyginti pagal šiuos kriterijus: greitį, plečiamumą, užklausų rašymą ir lankstumą.

Duomenų bazės plečiamumas (angl. *scalability*) - tai sistemos gebėjimas apdoroti augančius, ar potencialiai ateityje augančius duomenų ir operacijų su šiais duomenimis kiekius su tikslu kuo labiau sumažinti galimas rizikas. Plėsti į plotį reiškia įsigyti papildomų, dažniausiai mažesnio galingumo serverių. Yra du pagrindiniai būdai duomenų bazei plėstis į plotį – tai replikacija ir fragmentacija.

2.1 Redis

Šiame skyriuje pateikiama Redis duomenų bazės privalumai ir trūkumai, pastebėjimai.

2.1.1 Duomenų saugojimo modelis

Redis duomenų bazė yra rakto-reikšmės duomenų bazė ir saugojimo modeliu išsiskiria tuo, jog viskas yra saugoma atmintyje, todėl operacijos vykdomos labai greitai.

Ši duomenų bazė palaiko daug duomenų struktūrų. Tinka realaus laiko koordinatėms saugoti, tačiau labiau tinkamas trumpalaikiams duomenims (angl. *cache*), o ne ilgalaikiam saugojimui. Taip pat netinka kompleksiniams duomenų ryšiams.

Įvertis - Geras - 2 / 3 balų

2.1.2 Replikacijos galimybės duomenims

Redis duomenų bazė taip pat palaiko dviejų tipų replikacijas:

1. Master-slave: vienas pagrindinis serveris, kurį kopijuoja replikų serveriai.
2. Master-master: visi serveriai yra pagrindiniai, ir bet kuriame iš jų gali būti atliekamos tiek rašymo, tiek skaitymo komandos. Šios replikacijos tipas turi trūkumų, nes implementuojamas išorinės programinės įrangos ir nepilnai palaiko visas duomenų bazės komandas [8].

Įvertis - Geras - 2 / 3 balų

2.1.3 Duomenų paskirstymas tarp serverių

Duomenų bazės duomenų paskirstymas tarp serverių nėra tobulas Redis duomenų bazėje ir turi įvairių implementacijų. Pasirinkimas priklauso nuo saugomų duomenų tipo ir svarbumo.

Įvertis - Geras - 2 / 3 balų

2.2 MongoDB

Šiame skyriuje pateikiama MongoDB duomenų bazės privalumai ir trūkumai, pastebėjimai.

2.2.1 Duomenų saugojimo modelis

MongoDB – tai dokumentų talpykla, kurioje dokumentai yra saugomi BSON dokumentų formatu. MongoDB duomenų bazė saugo dokumentus kolekcijose, kur dažniausiai dokumentai turi vienodo tipo struktūrą. Šios užduoties atveju kiekvienas klientas, automobilis ir kelionė galėtų būti saugomi kaip dokumentai. Kadangi MongoDB paremta JSON formatu, ji taip pat gali saugoti ir nestruktūrizuotus dokumentus, kurių struktūra vis pastoviai kinta [6]. Puikiai tinka hierarchinėms struktūroms (pvz., kelionės ir jų pozicijos). Lanksti struktūra leidžia greitai integruoti naujus laukus (naudinga, jei sistema bus plečiama).

Įvertis - Puikus - 3 / 3 balų

2.2.2 Replikacijos galimybės duomenims

MongoDB replikacijai įgyvendinti naudojamas vienas pagrindinis serveris, kuris yra atsakingas už visų gaunamų rašymo operacijų saugojimą į specialią kolekciją. Antrinių serverių darbas yra skaityti tas operacijas iš pirminio serverio kolekcijos ir jas vykdyti. Antriniai serveriai taip pat gali nuskaityti operacijas iš kitų antrinių serverių taip mažindami apkrovimą nuo pirminio.

Taip sistema saugodama naujausius duomenis ne vienam, o keliuose serveriuose yra ne tik patikimesnė, bet gali būti ir daug greitesnė, nes užklausų srautas gali būti paskirstytas į keletą serverių.

Įvertis - Puikus - 3 / 3 balų

2.2.3 Duomenų paskirstymo galimybės tarp serverių

Duomenų paskirstymas yra implementuojamas pačios MongoDB duomenų bazės.

MongoDB paskirstytas klasteris susideda iš:

- Shard (dalių serveris): serveris su duomenų dalimi;
- Mongos: užklausų paskirstymo serveriai;
- Konfigūracijos serveriai: konfigūracijos serveriai saugo klasterio nustatymus ir meta duomenis.

MongoDB paskirsto duomenis kolekcijų duomenis tarp dalių serverių klasteryje pagal paskirstymo raktą (angl. *shard key*), kuris turi egzistuoti kiekviename dokumente.

Nutrūkus vieno dalių serverio (angl. *shard*) darbui, vis tiek bus pasiekiami kiti dalių serveriai ir bus tęsiamas užklausų vykdymas, jei tas serveris yra pasiekiamas [9].

Įvertis - Geras - 2 / 3 balų

2.3 Apache Cassandra

Šiame skyriuje pateikiama Apache Cassandra duomenų bazės privalumai ir trūkumai, pastebėjimai.

2.3.1 Duomenų saugojimo modelis

Apache Cassandra – tai atviro kodo paskirstyta stulpelio formato duomenų bazė, kuri palaiko daugybę duomenų tipų, įskaitant gimtuosius (angl. *native*), kolekcijų (angl. *collection*) tipus, naudotojo apibrėžtus (angl. *user-defined*) tipus, tuple tipus ir pasirinktinius tipus. Duomenų bazė palaiko duomenų struktūras: maps (rakto:reikšmės saugojimo duomenų struktūros), sets (išrikiuotos unikalių reikšmių kolekcijos) ir lists (išrikiuotos reikšmių kolekcijos (gali būti ne vien unikalios reikšmės)) [2]. Ši duomenų bazė itin tinkama dideliems duomenų kiekiams ir dideliam įrašų dažniui, tačiau silpniau palaikomos sudėtingos užklausos.

Įvertis - Geras - 2 / 3 balų

2.3.2 Replikacijos galimybės duomenims

Siekiant užtikrinti patikimumą ir atsparumą gedimams, ji turi puikią implementaciją duomenų replikacijai. Duomenų replikos laikomos keliuose serveriuose, tam kad užtikrinti pastovų duomenų pasiekiamumą [1].

Įvertis - Puikus - 3 / 3 balų

2.3.3 Duomenų paskirstymo galimybės tarp serverių

Apache Cassandra duomenų bazė standartiškai paskirsto duomenis nuo pradinio jos paleidimo, nes tai yra jos veikimo pricipo idėja. Apache Cassandra viena iš aktyviai palaikomų duomenų bazių kuri implementuoja nuoseklų žiedinį maišos (angl. *consistent hash ring*) algoritmą. Ši maišos funkcija leidžia algoritmui išskirstyti duomenis į tolygaus dydžio skirsnius, pridėjus ar ištrynus serverius, galima lengvai perskirstyti duomenis [3].

Įvertis - Puikus - 3 / 3

2.4 Neo4j

Šiame skyriuje pateikiama Neo4j duomenų bazės privalumai ir trūkumai, pastebėjimai.

2.4.1 Duomenų saugojimo modelis

Neo4j – tai atviro kodo grafų duomenų bazė, specialiai sukurta sudėtingų ryšių modeliavimui ir užklausoms tarp duomenų mazgų. Pagrindinis saugojimo modelis yra mazgai (angl. *nodes*) ir ryšiai (angl. *relationships*), kurie turi atributus (angl. *properties*) [4].

Įvertis - Geras - 2 / 3

2.4.2 Replikacijos ir duomenų paskirstymo galimybės

Neo4j replikacijos funkcionalumas yra labiau orientuotas į grafų užklausas, nei į masinį duomenų paskirstymą. Grafų architektūra apsunkina tolygų duomenų paskirstymą, nes tai reikalauja palaikyti ryšius tarp serveriuose išskirstytų duomenų.

Įvertis - Geras - 2 / 3

1 lentelė: Nereliacinių duomenų bazių palyginimas

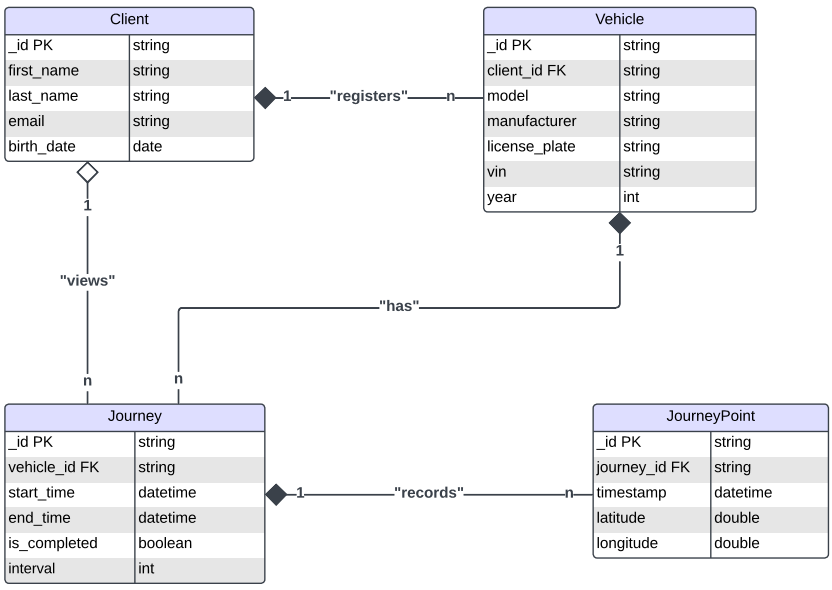
Duomenų bazė	Privalumai	Trūkumai	Tinkamumas
Redis	Greitas, palaikoma duomenų struktūrų įvairovė	Reikalauja daug atminties, netinkamas ilgalaikiam duomenų saugojimui, nesudėtingos užklausos	Tinka realaus laiko koordinatėms, bet ne istoriniams duomenims
MongoDB	Lankstus modelis, geografiniai indeksai, pilno teksto paieškos galimybė	Didelis saugojimo vietos poreikis, ribotas ACID	Tinka kelionių, koordinatėms ir pilno teksto paieškai dėl lankstumo
Cassandra	Greitas rašymas, replikacija, plėtimasis	Netinka sudėtingoms užklausoms, sudėtingas duomenų modeliavimas	Tinka labai dideliam duomenų kiekiui (rašymo operacija)
Neo4j	Ryšių modeliavimas, greitos užklausos	Lėtas masyviems duomenims, ribotas plėtimasis	Tinka tik ryšių analizei

Apibendrinant, MongoDB būtų tinkamiausia šios sistemos realizavimui dėl savo lankstumo, integracijų su geografiniais duomenimis ir pilno teksto (angl. *full text*) paieškos palaikymo.

Paskirstymas tarp serverių leis greitai ir efektyviai apdoroti užklausas. Pasirinkus tinkamus paskirstymo rakto laukus (pvz., `client_id`, `vehicle_id`), duomenys bus paskirstyti pagal svarbiausius sistemos komponentus ir užtikrins optimalų našumą. Replikacija užtikrins aukštą sistemos prieinamumą ir atsparumą, įvykus gedimams.

3 Fizinis duomenų modelis

Šiame poskyryje bus pateikiamas fizinis duomenų modelis – tai toks duomenų modelis, kurį kuriant yra žinomas ne tik duomenų bazės tipas, bet ir konkreti duomenų bazių valdymo sistema (DBVS).



1 pav.: Duomenų bazės struktūros diagrama

"originatingScript":"m2","payload":"guid":"9721a1ea-8b95-4c1b-8c51-df44f74acd1c58e8bb","muid":"f6cc52
d132-40d0-abd9-4e55d25257e68bc043","sid":"fe738736-9333-4296-8ca3-a909ea261bf268eb0b""originatingScript"
8b95-4c1b-8c51-df44f74acd1c58e8bb","muid":"f6cc52fc-d132-40d0-abd9-4e55d25257e68bc043","sid":"fe738736-
9333-4296-8ca3-a909ea261bf268eb0b""originatingScript":"m2","payload":"guid":"9721a1ea-8b95-4c1b-
8c51-df44f74acd1c58e8bb","muid":"f6cc52fc-d132-40d0-abd9-4e55d25257e68bc043","sid":"c94552d7-1c52-
4340-9c39-7ec56c730af9ccc65a""originatingScript":"m2","payload":"guid":"9721a1ea-8b95-4c1b-8c51-df44f74ac
d132-40d0-abd9-4e55d25257e68bc043","sid":"c94552d7-1c52-4340-9c39-7ec56c730af9ccc65a" Duomenų bazėje yra keturios lentelės: Client (Klientų), Vehicle (Transporto priemonių), Journey (Kelionių) ir JourneyPoints (Kelionės pozicijų) (1 pav.). Visų lentelių pagrindinis raktas (angl. *primary key, PK*) yra id.

Kiekvienas klientas turi unikalų identifikatorių _id, vardą, pavardę, el. paštą, gimimo datą. Klientas registruoja (sąryšis „1:n“) kelias transporto priemones (Vehicle).

Transporto priemonė turi unikalų identifikatorių _id, klientą, kuriam ji priklauso (client_id – *foreign* raktas), modelį, gamintoją, valstybinius numerius, VIN numerį bei gamybos metus. Viena transporto priemonė turi (sąryšis „1:n“) daug kelionių (Journey).

Kelionė turi unikalų identifikatorių _id, transporto priemonės identifikatorių (vehicle_id – *foreign* raktas), pradžios ir pabaigos laiką, užbaigtumo būseną (is_completed), intervalą. Viena kelionė įrašo (sąryšis „1:n“) daug kelionės taškų (JourneyPoint).

Kelionės taškas turi unikalų identifikatorių _id, kelionės identifikatorių (journey_id – *foreign*), laiko žymą, platumą bei ilgumą.

Pagrindiniai MongoDB duomenų modelio komponentai: dokumentas ir kolekcija. Kolekcija yra grupė dokumentų, kurie yra susiję tarpusavyje. Kelionės ir kelionės taškai modeliuojamos atskirai, nes šių duomenų kiekis gali labai išaugti, todėl taip užtikrinamas geresnis našumas atliekant užklausas su dideliais duomenų rinkiniais.

Kliento kolekcija (angl. *client collection*), kliento duomenys saugomi su transporto priemonių duomenimis:

```
{
  "_id": "client_id_1",
  "first_name": "John",
  "last_name": "Doe",
  "email": "john.doe@example.com",
  "birth_date": "1990-01-01"
}
```

Transporto priemonių kolekcija (angl. *vehicle collection*), saugo informaciją apie vartotojo registruotas transporto priemones:

```
{
  "_id": "vehicle_id_1",
  "client_id": "client_id_1",
  "model": "Model S",
  "manufacturer": "Tesla",
```

```
  "license_plate": "XYZ1234",
  "vin": "5YJSA1E16GF120233",
  "year": 2020
}
```

Kelionių kolekcija (angl. *journeys collection*), kelionės duomenys saugomi nuorodomis į transporto priemones pagal (`vehicle_id`) ir įterpiamos kelionės pozicijos:

```
{
  "_id": "journey_id_1",
  "vehicle_id": "vehicle_id_1",
  "start_time": "2024-01-01T08:00:00Z",
  "end_time": "2024-01-01T10:00:00Z",
  "is_completed": true
}
```

Kelionės taškų kolekcija (angl. *journey points collection*), saugo informaciją apie transporto priemonės buvimo vietą tam tikrais intervalais:

```
{
  "_id": "point_id_1",
  "journey_id": "journey_id_1",
  "timestamp": "2024-01-01T08:10:00Z",
  "latitude": 37.7749,
  "longitude": -122.4194
}
```

4 Kelionių registracijos serviso kūrimas

Šiame skyriuje bus pateikiama web serviso, leidžiančio navigacijos įrenginiams registruoti keliones, kūrimo eiga, naudotos užklausos, pagal sudarytą API aprašą [7].

4.1 Funkcija `register_client`

```
@app.route('/clients', methods=['PUT'])
def register_client():
```

Funkcija `register_client` yra API maršruto valdiklis, kuris leidžia registruoti naują klientą į duomenų bazę. Ši funkcija naudoja PUT HTTP metodą ir saugo kliento informaciją JSON formatu.

Įvesties (angl. *input*) duomenys:

- **first_name** (string): Kliento vardas.
- **last_name** (string): Kliento pavardė.
- **email** (string): Kliento el. pašto adresas.
- **birth_date** (string): Kliento gimimo data formatu YYYY-MM-DD.

4.2 Funkcija `get_client_details`

```
@app.route('/clients/<string:clientId>', methods=['GET'])
def get_client_details(clientId):
```

Funkcija `get_client_details` leidžia gauti konkretaus kliento detalią informaciją pagal jo unikalų identifikatorių.

Parametras: `client_id` (string): Kliento identifikatorius. Turi būti ObjectId formato, naudojamo MongoDB.

4.3 Funkcija `register_vehicles`

```
app.route("/vehicles", methods=["PUT"])
def register_vehicle():
```

Funkcija `register_vehicle` yra API užklausos taškas (/vehicles), kuris leidžia registruoti naują transporto priemonę kliento vardu. Funkcija patikrina, ar pateikti duomenys apie klientą ir transporto priemonę yra teisingi, o tada prideda transporto priemonę į duomenų bazę.

Įvesties (angl. *input*) duomenys:

- **client_id** (string): Kliento ID, kuris turi būti jau egzistuojantis duomenų bazėje.
- **model** (string): Transporto priemonės modelis.
- **manufacturer** (string): Transporto priemonės gamintojas.
- **license_plate** (string): Transporto priemonės valstybinis numeris.
- **vin** (string): Unikalus transporto priemonės identifikavimo numeris (VIN).
- **year** (integer): Transporto priemonės pagaminimo metai.

4.4 Funkcija `get_vehicles_by_client`

```
@app.route("/clients/<client_id>/vehicles", methods=["GET"])
def get_vehicles_by_client(client_id):
```

Ši funkcija skirta gauti visas transporto priemones, kurios yra susijusios su nurodytu klientu pagal `client_id`.

Parametras: `client_id` (string): Kliento unikalus identifikatorius (ObjectId formatu).

4.5 Funkcija `start_journey`

```
@app.route("/journeys", methods=["POST"])
def start_journey():
```

Funkcija `start_journey` pradeda kelionę konkrečiai transporto priemonei.

Įvesties duomenys (angl. *input*):

- **vehicle_id** (string): Transporto priemonės ID, kuris jau turi būti duomenų bazėje.
- **interval** (integer): Koordinačių registravimo intervalas (sekundėmis) — turi būti ne trumpesnis nei 5 sekundės.

4.6 Funkcija `log_coordinates`

```
@app.route("/journeys/<string:journey_id>/coordinates", methods=["POST"])
def log_coordinates(journey_id):
```

Funkcija `log_coordinates` registruoja transporto priemonės esamas koordinates pagal unikalų kelionės identifikatorių.

Įvesties duomenys (angl. *input*) šiai funkcijai:

- **journey_id** (string): Kelionės ID, kuris jau turi būti duomenų bazėje.
- **latitude** (number): Transporto priemonės platumos koordinatė.
- **longitude** (number): Transporto priemonės ilgumos koordinatė.
- **timestamp** (string): Užregistruotos pozicijos laiko žyma formatu YYYY-MM-DDTHH:MM:SSZ.

4.7 Funkcija `log_coordinates_periodically`

```
def log_coordinates_periodically(journey_id, interval):
```

Funkcija `log_coordinates_periodically` periodiškai registruoja koordinates konkrečiai kelionei.

4.8 Funkcija `get_journey_details`

```
@app.route("/journeys/<string:journey_id>", methods=["GET"])
def get_journey_details(journey_id):
```

Funkcija `get_journey_details` skirta išgauti kelionės detales: kelionės trukmę laiku, kelionės atstumą (kelionės atstumas yra visų atkarpų tarp dviejų iš eilės einančių taškų suma).

Skaičiavimai atliekami MongoDB Aggregation Pipeline pagalba [5].

Naudojamas unikalus identifikatorius `journeyId` (string).

4.9 Funkcija `get_vehicle_statistics`

```
@app.route("/vehicles/<string:vehicle_id>/statistics", methods=["GET"])
def get_vehicle_statistics(vehicle_id):
```

Funkcija `get_vehicle_statistics` skirta išgauti informaciją apie transporto priemonės užregistruotas keliones, klientai gali gauti bendrą konkretaus automobilio kelionių trukmę ir atstumą.

Skaičiavimai atliekami MongoDB Aggregation Pipeline pagalba [5].

Naudojamas unikalus identifikatorius `vehicleId` (string).

4.10 Funkcija `end_journey`

```
@app.route("/journeys/<journey_id>/end", methods=["PUT"])
def end_journey(journey_id):
```

Funkcija `end_journey` skirta baigti kelionę.

Naudojamas unikalus identifikatorius `journeyId` (string).

4.11 Funkcija `flush_all`

```
@app.route('/cleanup', methods=['POST'])
def flush_all():
```

Funkcija `flush_all` ištrina `travel_registration_system` duomenų bazę, įskaitant visas jos kolekcijas (`clients` ir `vehicles`, `journeys`, `journey points`). Tam sukuriamas API užklausos taškas (angl. *servise endpoint*) `POST /cleanup`.

4.12 Funkcija `full_text_search()`

```
@app.route('/search', methods=['GET'])
def full_text_search():
```

Funkcija `full_text_search()` leidžia įgyvendinti pilno teksto paieškos funkcionalumą trijose MongoDB kolekcijose: `clients_collection`, `vehicles_collection`, ir `journeys_collection`.

5 Išvados

1. Įvertinus žinomas nereliacines duomenų bazes, jų trūkumus ir privalumus, siekiant realizuoti kelionių registracijos sistemą geriausia naudoti MongoDB duomenų bazę.

2. Buvo sukurtas fizinis duomenų modelis, sudarytos keturios kolekcijos: kliento, transporto priemonių, kelionių ir kelionės taškų kolekcija.

3. Naudojant API aprašą buvo sukurtas servisas, leidžiantis navigacijos įrenginiams registruoti keliones.

Literatūra

[1] Data replication | apache cassandra 3.0. https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/architecture/archDataDistributeReplication.html, 2024. Accessed: 2024-12-15.

[2] Data types | apache cassandra documentation. https://cassandra.apache.org/doc/stable/cassandra/cql/types.html, 2024. Accessed: 2024-12-15.

[3] Dynamo | apache cassandra documentation. https://cassandra.apache.org/doc/latest/cassandra/architecture/dynamo.html, 2024. Accessed: 2024-12-15.

[4] Graph database concepts - getting started. https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/, 2024. Accessed: 2024-12-15.

[5] Mongodb aggregation pipeline | mongodb. https://www.mongodb.com/resources/products/capabilities/aggregation-pipeline, 2024. Accessed: 2024-12-15.

[6] Mongodb documentation. https://www.mongodb.com/docs/, 2024. Accessed: 2024-12-15.

[7] Navigation device travel registration system. https://evelinavait.github.io/NoSQL-Uzd5/, 2024. Accessed: 2024-12-12.

[8] Redis. https://redis.io/docs/latest/, 2024. Accessed: 2024-12-15.

[9] Sharding - mongodb manual v8.0. https://www.mongodb.com/docs/manual/sharding/, 2024. Accessed: 2024-12-15.