Razvoj usluge interaktivne karte za prikaz podataka o kretanju vozila prometnom mrežom

Goldner, Hrvoje

SVEUČILIŠTE U ZAGREBU

FAKULTET PROMETNIH ZNANOSTI

Hrvoje Goldner

RAZVOJ USLUGE INTERAKTIVNE KARTE ZA PRIKAZ PODATAKA O KRETANJU VOZILA PROMETNOM MREŽOM

DIPLOMSKI RAD

Zagreb, 2023.

Sveučilište u Zagrebu

Fakultet prometnih znanosti

DIPLOMSKI RAD

RAZVOJ USLUGE INTERAKTIVNE KARTE ZA PRIKAZ PODATAKA O KRETANJU VOZILA PROMETNOM MREŽOM

DEVELOPMENT OF INTERACTIVE MAP SERVICE FOR DISPLAYING VEHICLE MOVEMENT DATA THROUGH THE TRAFFIC NETWORK

Mentor: dr. sc. Tomislav Erdelić

Student: Hrvoje Goldner

JMBAG: 0135242494

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI
POVJERENSTVO ZA DIPLOMSKI ISPIT

Zagreb, 1. lipnja 2023.

Zavod: **Zavod za inteligentne transportne sustave**
Predmet: **Napredne baze podataka**

DIPLOMSKI ZADATAK br. 7372

Pristupnik: **Hrvoje Goldner (0135242494)**
Studij: Promet
Smjer: Informacijsko-komunikacijski promet

Zadatak: **Razvoj usluge interaktivne karte za prikaz podataka o kretanju vozila prometnom mrežom**

Opis zadatka:

U sklopu rada potrebno je razviti uslugu interaktivne karte (web aplikacija) koja će služiti za prikaz podataka o kretanju vozila prometnom mrežom. Podaci o kretanju vozila prometnom mrežom spremljeni su u komercijalnu bazu pokretnih objekata na poslužitelju, te je potrebno razviti sustav za rad s navedenom bazom podatka, pri čemu je prvenstveno naglasak na dohvatu podataka. Na temelju razvijene interaktivne karte potrebno je provesti analizu spremljenih podataka i opisati sve mogućnosti razvijene interaktivne karte. Potom je potrebno spremiti dio podataka u bazu pokretnih objekata otvorenog koda, koristeći besplatan sustav za upravljanje bazom pokretnih objekata. Na kraju je potrebno usporediti razlike u arhitekturi i vremenu dohvata između besplatnog i komercijalnog sustava za upravljanje bazom podatka pokretnih objekata.

Mentor:

Tomislav Erdelić

dr. sc. Tomislav Erdelić

Predsjednik povjerenstva za diplomski ispit:

RAZVOJ USLUGE INTERAKTIVNE KARTE ZA PRIKAZ PODATAKA O KRETANJU VOZILA PROMETNOM MREŽOM

SAŽETAK

U ovom radu predstavljen je razvoj sučelja za obradu, filtriranje i prikaz velike količine podataka o kretanju vozila kroz prometnu mrežu. Cilj rada je prezentacija informacija o dostupnim rješenjima u polju prostorno-vremenskih baza podataka te korištenje istih u svrhu učinkovite pohrane, dohvaćanja i vizualizacije prostorno-vremenskih podataka, te razvoja aplikacija koje koriste iste. Prikazane su osnove sustava baza podataka, relacijskog modela, relacijskih baza podataka i SQL jezika, kreiranja pokazivača i različitih dostupnih stabala za pohranu istih. Zatim su opisane mogućnosti implementacije prostorno-vremenskih podataka kroz dostupne inačice baza podataka i sustava za upravljanje istim, te usporedbe performansi njihovog rada prilikom dohvata podataka. Implementirano je sučelje za dohvat prostorno-vremenskih podataka iz Mireo Space-Time baze podataka, te prikaz na karti, čime su prikazana moguća rješenja i dostupne mogućnosti za obradu, prikaz, prijenos i pohranu prostorno-vremenskih podataka.

KLJUČNE RIJEČI: relacijske baze podataka, SQL, prostorno-vremenske baze podataka, stabla pokazivača, prikaz podataka o kretanju vozila, Mireo Space-Time, SECONDO, interaktivna karta, Leaflet, GeoJSON

SUMMARY

This paper describes the creation of an interface that will process, filter, and display a sizable amount of traffic network vehicle movement data. The objective is to provide information about the solutions that are available for effective store, data retrieval, and visualization of spatio-temporal data, as well as details on how to use those data to develop applications. The fundamentals of relational model and databases, the development of indices and the various tree types that can be used to store them, as well as the modern technologies that are currently available, are discussed. The implementation of spatio-temporal data through numerous database and control system versions is then covered, along with performance comparisons. Viable solutions and accessible capabilities for processing, visualization, transmission, and storage of spatio-temporal data have been highlighted through the implementation of an interface for retrieving data from the Mireo Space-Time database and displaying it on a map.

KEYWORDS: relation databases, SQL, spatio-temporal databases, indexing trees, displaying vehicle movement data, Mireo Space-Time, SECONDO, interactive map, Leaflet, GeoJSON

Sadržaj

1. Uvod

U današnje vrijeme podaci su postali jedan od najvrjednijih elemenata poslovanja u mnogim granama, što zauzvrat čini područje sustava za obradu i pohranu podataka znatno popularnim i raznovrsnim. Trend porasta veličine i količine prikupljenih podataka nalaže potrebu za konstantnom nadogradnjom postojećih rješenja i razvojem novih. Prikupljeni podaci, obradom pretvoreni u korisne informacije, daju veoma važne uvide u prošlost, sadašnjost te donekle mogućnosti predviđanja stanja u budućnosti.

Podaci o kretanju vozila prometnom mrežom iznimno su bitni i primjenjivi u raznim industrijama, koje uključuju osiguranja, upravljanje flotom, planiranje i projektiranje prometnica, i dr. Kako bi se takvi podaci mogli suvislo iskoristiti u analizama, potrebno ih je prikupljati sa raznovrsnih vozila, dulji period vremena, što čini sustav za pohranu, obradu i dohvat podataka kompleksnim.

Svrha diplomskog rada naziva **Razvoj usluge interaktivne karte za prikaz podataka o kretanju vozila prometnom mrežom** je prikaz dostupnih tehnologija i sustava baza podataka, povijesti i razvoja istih, te mogućnosti dostupnih sustava za kreiranje rješenja za pohranu i obradu podataka s prostornom i vremenskom komponentom. Rad je podijeljen na šest poglavlja:

1. Uvod,
2. Relacijske baze podataka,
3. Prostorno-vremenske baze podataka,
4. Sučelje za dohvat, obradu, interakciju i prikaz podataka na karti,
5. Analiza rada implementiranih prostorno-vremenskih baza podataka,
6. Zaključak.

Drugo poglavlje čini povijest, razvoj, principi i mogućnosti baza podataka, te najzastupljenijeg tipa istih – relacijskih baza podataka. Unutar istog spominje se jezik upita korišten za baze podataka, te nove dostupne tehnologije i njihove prednosti i nedostaci. Navedeni su primjeri popularnih komercijalnih dostupnih rješenja sustava baza podataka.

Treće poglavlje objašnjava koncept prostora i vremena u bazama podataka, odnosno u kontekstu podataka i pohrane istih. Navedeni su standardi i dostupna proširenja sustava baza podataka u svrhu podrške prostorno-vremenskih podataka, te su objašnjene razlike između

prostornih, vremenskih te prostorno-vremenskih baza podataka – i vrste podataka koji se pohranjuju u iste.

Četvrto poglavlje sastoji se od dokumentacije procesa i postupaka izrade sučelja za prikaz podataka o kretanju vozila prometnom mrežom. Navedene su dostupne tehnologije za izradu, te odabrane implementacije, programski jezici, alati i biblioteke. Sučelje se sastoji od grafičkog korisničkog sučelja i među-sloja za dohvat i obradu podataka iz baze podataka. Baza podataka je prostorno-vremenska baza podataka razvijena i konfigurirana od strane tvrtke Mireo d.d., te ustupljena Fakultetu prometnih znanosti za istraživačke svrhe.

Peto poglavlje pruža uvid u razlike u performansama prilikom dohvata prostornih podataka između baza podataka generalne namjene, te baze podataka namijenjene za prostorne podatke. Objašnjeni su procesi pripreme podataka i sustava baza podataka, te razlike u vremenima dohvata za određene pred-definirane upite.

Zaključak rada predstavlja osvrt na mogućnosti kreiranog sučelja unutar ovog rada, mogućnosti i primjene tradicionalnih sustava baza podataka te budućnost istih.

2. Relacijske baze podataka

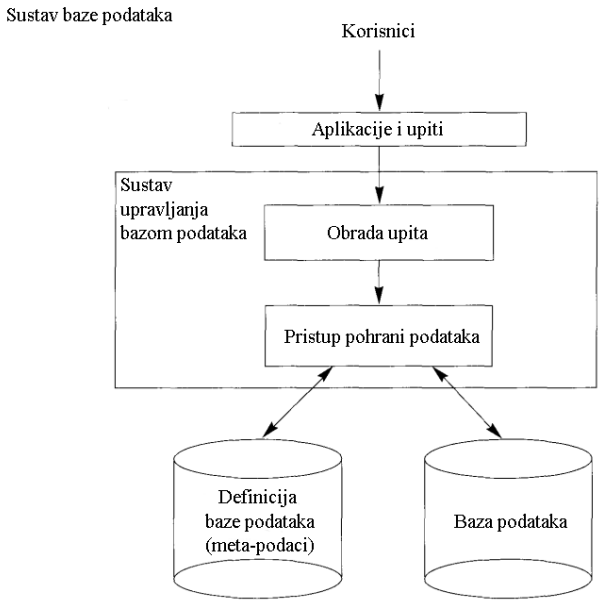
Smatra se da skoro svi poslovni i društveni procesi osnovu rada temelje na podacima. Podatak (engl. „*data*“) je najmanja sirova činjenica koja ima značenje u stvarnom svijetu, te sam po sebi nema korisno značenje. Obrada podataka je proces interpretacije podatka ili skupine podataka u svrhu dobivanja korisne informacije, [1].

Baza podataka predstavlja kompleksan sustav prikupljanja i pohrane međusobno povezanih podataka te posluživanje istih prema krajnjim korisnicima ili organizacijama. Podatak u bazi podataka naziva se podatkovna stavka (engl. „*data item*“) – npr. ime ili prezime zaposlenika, adresa i dr. Grupa shodnih podataka naziva se zapis (engl. „*record*“) te se u bazi podataka promatra kao zasebna jedinka, npr. jedan zaposlenik. Datoteka (engl. „*file*“) predstavlja skup zasebnih zapisa istog tipa – npr. zapisi o više zaposlenika. U relacijskim bazama podataka, podatkovna stavka naziva se stupac (engl. „*column*“) ili atribut, zapis se naziva retkom (engl. „*row*“), a zajedno čine relaciju, čija manifestacija se može vizualno prikazati tablicom, [2].

Relacijska baza podataka osim samih podataka sadrži veze i odnose između grupe podataka, stoga kako bi se skup podataka mogao smatrati bazom podataka, podaci u skupu moraju imati značenje i međusoban odnos – grupa nasumičnih podataka bez zajedničkog značenja ne naziva se bazom podataka, [3].

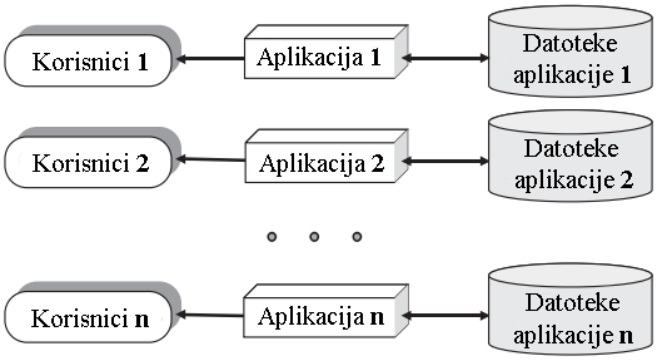
Kreiranje, održavanje te upravljanje bazom podataka vrši se kroz sustav upravljanja bazom podataka (engl. DBMS – „*Database management system*“). DBMS je skup softverskih alata koji omogućavaju definiranje, izradu, rukovanje te dijeljenje baza podataka krajnjim korisnicima i aplikacijama. Baza podataka i sustav upravljanja bazom podataka zajedno se nazivaju sustavom baze podataka (engl. „*Database system*“), [2], [3], prikazan na slici 1.

Sustav baze podataka, osim samih skupova podataka pohranjenih u bazi podataka sadrži metapodatke (engl. „*metadata*“) – skup podataka koji opisuju bazu podataka i podatke pohranjene u istoj. Metapodaci su podaci o podacima – opisuju strukturu baze podataka, tipove, strukture i ograničenja pohranjenih podataka, [3]. Logička struktura baze podataka još se naziva i shema (engl. „*schema*“), [2].



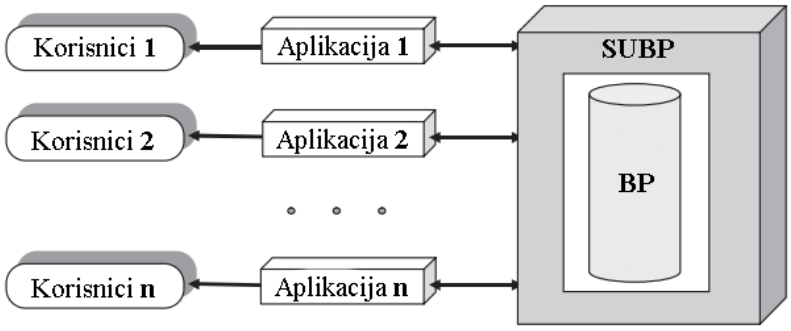
Slika 1: Sustav baze podataka [3]

Tradicionalni način pohrane i pristupa podacima, prikazan slikom 2 temelji se na datotekama – svaki korisnik definira, koristi i ažurira datoteke potrebne za njemu specifičan slučaj uporabe. Problemi kod ovakvog pristupa podacima nastaju ukoliko više korisnika rade na datotekama koje sadrže bitnu količinu istih podataka – u tom slučaju nastaje znatna količina redundancije i nepotrebno utrošenog prostora za pohranu, što otežava ažuriranje zajedničkih podataka pohranjenih u više datoteka, [3].



Slika 2: Sustav pristupa podacima baziran na datotekama [1]

Baze podataka i sustavi baza podataka razvijeni su iz potrebe za dugoročnim pohranjivanjem podataka te istovremenim pristupanjem istima od strane većeg broja korisnika i aplikacija. Prednosti ovakvog pristupa podacima, prikazanog na slici 3, vidljive su kroz središnje (centralizirano) upravljanje podacima, neovisnost između podataka i aplikacija, povećanom integritetu podataka, smanjenom proturječnosti podataka te podrškom za različite poglede na podatke ovisno o potrebama korisnika, [1].

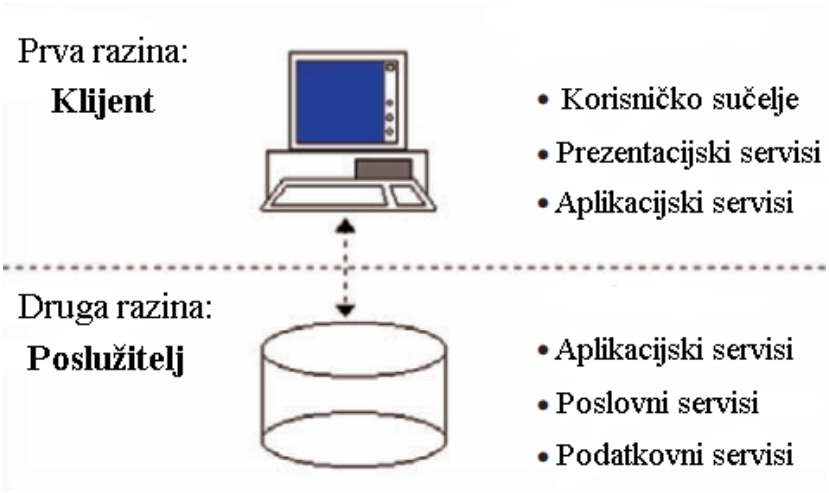


Slika 3: Pristup podacima koristeći sustav baze podataka, [1]

Arhitektura sustava baze podataka može se podijeliti s obzirom na broj logičkih razina na kojima se nalazi programska podrška – razlikuje se arhitektura u dvije te tri ili više razina, [1].

Arhitektura u dvije razine podrazumijeva arhitekturu klijenta i poslužitelja, gdje se na klijentskoj strani nalazi (grafičko) korisničko sučelje (engl. GUI – „*Graphical User Interface*“) te aplikacije i programska podrška korištena za poslovanje, dok se na serverskoj strani nalazi programska podrška za posluživanje upita i obradu transakcija – tj. funkcionalnosti potrebne za dohvat i manipulaciju podacima. Ovakav tip poslužitelja naziva se i poslužitelj upita (engl. „*query server*“) ili poslužitelj transakcija (engl. „*transaction server*“). Alternativni način posluživanja je da klijent osim prethodno navedenih funkcionalnosti sadrži i funkcionalnosti upita i transakcija, dok podatkovni poslužitelj (engl. „*data server*“) sadrži samo podatke te funkcionalnosti za dohvat podataka, [3].

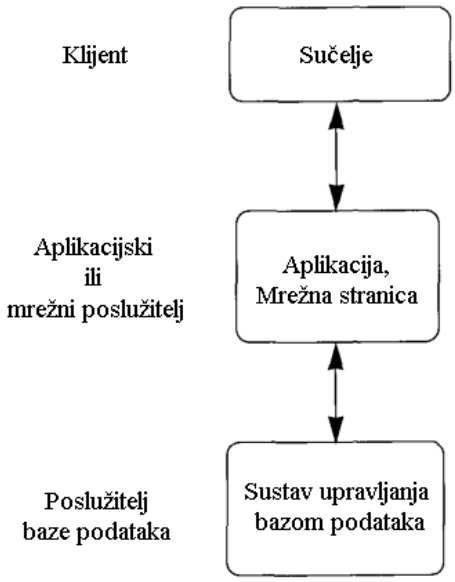
Osnovna izvedba arhitekture u dvije razine, prikazana na slici 4, neovisno o funkciji poslužitelja, podrazumijeva podjelu odgovornosti između klijenta i poslužitelja na način da je glavna uloga klijenta prezentacija podataka, a glavna uloga poslužitelja je pružanje podatkovnih usluga klijentu, [1].



Slika 4: Dvorazinska arhitektura [1]

Dodavanjem razine između poslužitelja i klijenta postižu se veće performanse naspram arhitekture u dvije razine, zbog čega se najčešće primjenjuje. Arhitektura podijeljena na tri ili više logičkih razina, prikazana na slici 5, dodatno pruža veću skalabilnost aplikacija zbog mogućnosti ponovnog korištenja implementiranih programskih komponenti i rješenja, [1].

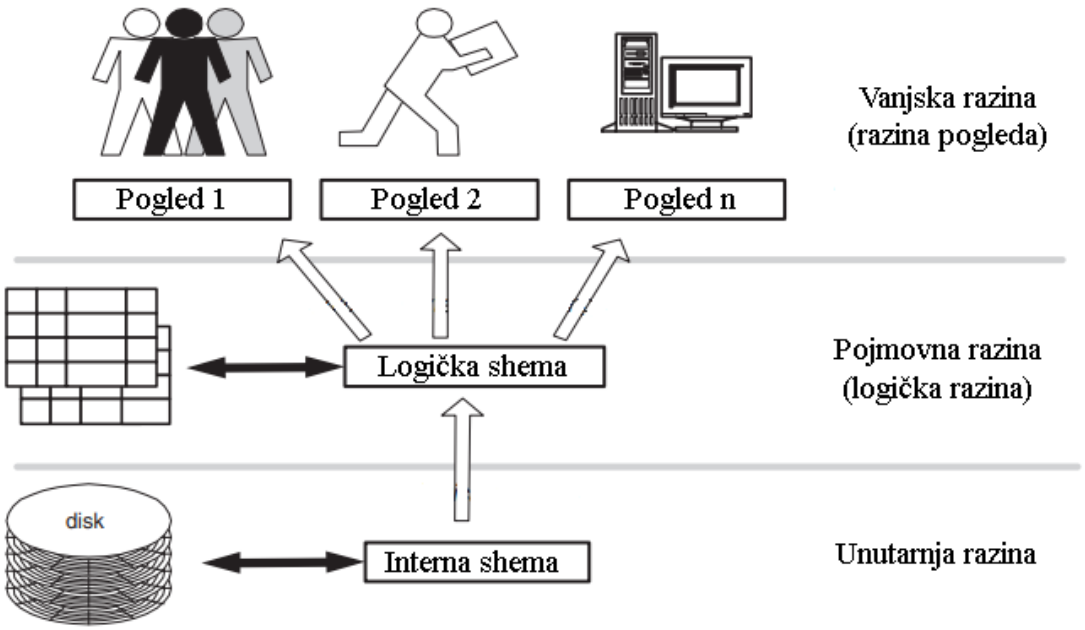
Arhitektura u tri razine izvedena je pomoću dodatne među-razine između sustava baze podataka i razine za prikaz podataka. Svrha ove razine, koja se ovisno o implementaciji i potrebi aplikacije naziva aplikacijski ili web poslužitelj, je da sadrži poslovnu logiku, proceduru i načine dohvata podataka iz baze podataka te opcionalno autentikacije korisnika prije samog pristupa sustavu bazi podataka. Na ovaj način u potpunosti se odvaja razina pohrane, pristupa i izmjene podataka – poslužitelj baze podataka te razina poslovne logike, aplikacija i programa – aplikacijski ili web poslužitelj i klijentska razina za prikaz samih podataka na grafičkom ili web sučelju, [3].



Slika 5: Arhitektura podijeljena na tri logičke razine, [3]

Opis baze podataka naziva se shema baze podataka te se sastoji od skupova objekata koji pružaju logičku klasifikaciju objekata u bazi podataka. Shema baze podataka definira se prilikom procesa dizajniranja baze podataka te može biti promjenjiva, ali u praktičnim primjenama se rijetko mijenja. Za razliku od sheme, podaci u bazi podataka se često mijenjaju te se trenutno stanje i vrijednosti podataka u nekom trenutku u vremenu nazivaju stanjem baze podataka, snimka baze podataka (engl. „*snapshot*“) ili instanca baze podataka (engl. „*database instance*“), [1, 3].

Razlika između samih podataka i sheme baze podataka omogućuje neovisnost podataka i aplikacija te višekorisnički rad, stoga je 1978. godine definiran ANSI (engl. „*American National Standard Institute*“) / SPARK (engl. „*Standards Planning and Requirements Committee*“) općeniti okvir za sustave baze podataka koji pruža tri razine apstrakcije za prikaz baze podataka, [1].



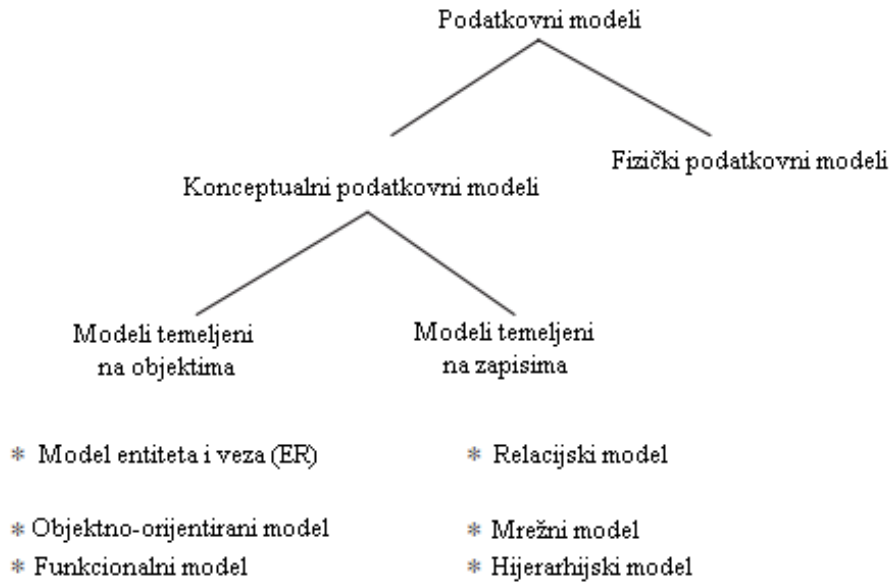
Slika 6: ANSI/SPARK model, [1]

Arhitektura u tri sheme, tj. razine apstrakcije, prikazan na slici 6, sastoji se od sljedećih razina:

- Unutarnja razina – sadrži internu shemu koja opisuje strukturu fizičke pohrane podataka, podatkovni model i putanje za pristup podacima,
- Pojmovna ili logička razina – sadrži konceptualnu shemu koja opisuje entitete, tipove podataka, veze, korisničke operacije i ograničenja,
- Vanjska ili razina pogleda – sadrži korisničke poglede – poglede na dijelove baze podataka potrebne određenom korisniku ili grupi korisnika, [3].

Ovako dizajnirana arhitektura omogućuje neovisnost podataka u dvije razine, [3]:

1. Logička neovisnost podataka – mogućnost izmjene konceptualne sheme bez promjene vanjske sheme ili aplikacije,
2. Fizička neovisnost podataka – mogućnost izmjene interne sheme bez potrebe za promjenom konceptualne sheme.



Slika 7: Podjela podatkovnih modela, [1]

Za opis logičke razine koriste se podatkovni modeli – jezici za opisivanje podataka i baze podataka te definiranje sheme. Osnovna podjela modela, prikazana na slici 7, je na modele više razine – konceptualne (pojmovne) i modele niže razine – fizičke modele. Modeli više razine koncipirani su prema načinu na koji korisnici opažaju podatke, dok modeli niže razine opisuju načine na koji su podaci pohranjeni u računalu, te kao takvi ne moraju biti razumljivi krajnjim korisnicima, [3].

Konceptualni podatkovni modeli za reprezentaciju podataka koriste se sljedećim pojmovima:

- entiteti – predstavljaju objekte ili koncepte iz stvarnog svijeta koji se opisuju u bazi podataka,
- atributi – podrobniji opis korištenih entiteta,
- veze – odnosi između 2 ili više entiteta.

Implementacijski podatkovni modeli, još poznati kao prezentacijski, najčešće su korišteni u tradicionalnim sustavima baza podataka na logičkoj razini ili razini pogleda, a uključuju najčešće korišteni model – relacijski model te mrežni model i hijerarhijski model. Nazivaju se i podatkovnim modelima temeljenim na zapisima, jer prikazuju podatke u strukturama zapisa – svaki podatak prikazuje se kao skup zapisa koji sadrže određen broj polja, odnosno atributa, ograničene veličine, [4].

Hijerarhijski podatkovni model može se grafički prikazati kao preokrenuto stablo, na način da jedan entitet (tablica) baze podataka predstavlja korijen stabla, a druge entitete (tablice) se povezuju na korijen kao grane drveta. Veze između tablica mogu se opisati relacijom roditelja i djece – jedna roditeljska tablica može imati jednu ili više tablica djece, dok svaka tablica može imati isključivo jednu roditeljsku tablicu. Ovakav način eksplicitnih veza između tablica omogućuje veliku brzinu dohvata podataka, te ugrađenu metodologiju automatskog očuvanja referencijalnog integriteta tablice – ukoliko se obriše zapis iz roditeljske tablice, brišu se i svi zapisi iz tablica djece povezane s obrisanim zapisom roditelj tablice. Nedostatak hijerarhijskog podatkovnog modela je kreiranje kompleksnih veza – kao što su veze više na prema više, gdje dolazi do velike redundancije podataka, [5].

Mrežni podatkovni model sastoji se od čvorova (engl. „*nodes*“) i veza (engl. „*set structures*“) između njih. Čvorovi se sastoje od zbirke zapisa, dok veze između njih predstavljaju informaciju koji čvor je vlasnik, a koji čvor član u toj vezi. Ovaj tip veze podržava jedan na prema više veze – čvor koji je vlasnik može imati jedan ili više članova, ali svaki dodani zapis u članskom čvoru se mora vezati na neki od postojećih zapisa u vlasničkom čvoru. Razlika ovog modela naspram hijerarhijskog modela je u vezama – u mrežnom modelu jedan čvor može biti povezan sa više vlasnika, dok je u hijerarhijskom modelu čvor mogao imati isključivo jedan povezani čvor kao roditelj. Ovaj napredak omogućuje znatno brže pregledavanje i dohvat podataka naspram korištenja hijerarhijskog modela, te mogućnosti dohvata podataka iz bilo kojeg čvora i stvaranje kompleksnih upita, što u prijašnjem modelu nije bilo moguće, već se dohvat vršio od izvorne tablice prema povezanim tablicama. Nedostaci ovog pristupa su znatno otežana izmjena veza između čvorova – izmjenom veze potrebno je u potpunosti izmijeniti i aplikacijske programe kojima je potreban pristup podacima u bazi podataka, [5].

Podatkovni model temeljen na objektima relativno je nova kategorija podatkovnih modela više razine, te se kao takav koristi kao konceptualni podatkovni model, najčešće u programskom inženjerstvu, [3].

2.1. Relacijski model

Relacijski model 1969. godine definirao matematičar Dr. Edgar F. Codd, tadašnji istraživač u IBM-u iz nezadovoljstva prema postojećim rješenjima za upravljanje velikom količinom podataka. Model je iznimno brzo postao popularan zbog svoje jednostavnosti i matematičke podloge. Osnova relacijskog modela izvedena je iz dviju grana matematike: teorije skupova i

izraza prvog reda (engl. „*first-order predicate*“). Model za prikaz podataka koristi matematičku relaciju, po čemu je i dobio ime, čiji rezultat se može prikazati kao tablica vrijednosti. Prvu implementaciju relacijskog modela izvršio je Oracle 1980. godine, [3].

Matematička relacija je skup elemenata n-torki (engl. „*tuples*“), od kojih su svi istog tipa, te čiji poredak nije bitan. Ukoliko se definiraju skupovi n-torki S_1, S_2, …, S_n, R je relacija nad tim skupovima ako se prvi element dohvaća iz S1, drugi iz S2 i tako do n-te n-torke, odnosno R je podskup Kartezijevog produkta (umnoška) $S_1 \times S_2 \times \ldots \times S_n$. Relacija R se u relacijskom modelu očituje kao tablica sa sljedećim svojstvima, [6]:

- svaki redak predstavlja jednu n-torku podskupa R,
- poredak redova je beznačajan,
- svi redovi se međusobno razlikuju u informacijskom sadržaju.

Razlike između matematičke relacije i relacije u relacijskom modelu prikazane su tablicom 1:

Tablica 1: Razlike matematičke relacije i relacije u relacijskom modelu, [6]

Matematička relacija	**Relacijski model**
Slobodne, neograničene vrijednosti	Atomične vrijednosti[1]
Neimenovani stupci	Svaki stupac različito imenovan
Stupci se razlikuju po poziciji	Stupci se razlikuju po imenu
Tipično konstanta	Tipično se mijenja s vremenom

Svaka relacija sastoji se od n-torki, odnosno zapisa, te atributa ili polja. U relacijskom modelu, sljedeće dvije karakteristike omogućuju logički pristup i izmjenu podataka neovisno o načinu na koji su fizički pohranjeni, [5]:

1. svaki zapis se identificira pomoću atributa jedinstvene vrijednosti,
2. redoslijed zapisa, tj. redova te atributa, odnosno stupaca nema značenje.

Mogućnosti relacijskog modela mogu se podijeliti na:

- strukturna svojstva – opisuju implementaciju relacijskog modela u bazi podataka kao skup relacija,

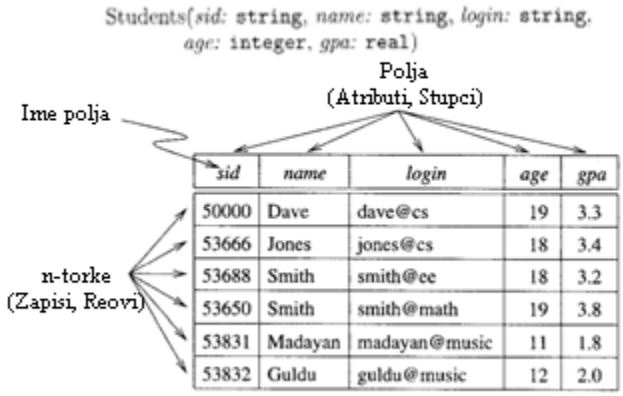
[1] Atomične vrijednosti podrazumijevaju vrijednosti koje se ne mogu nadalje podijeliti na manje dijelove u kontekstu relacijskog modela, osim preko posebnih funkcija sustava za upravljanje bazom podataka. [6]

- svojstva očuvanja integriteta – opisuju postupke očuvanja cjelovitosti baze podataka,
- manipulativna svojstva – skup alata za rukovanje podacima, tj. relacijama, [1].

2.1.1 Strukturna svojstva

Strukturna svojstva relacijskog modela opisuju načine prezentacije podataka u relacijskom modelu – prikaz relacije kao tablice. Relacije se definiraju nad klasama, tj. domenama – skupovima vrijednosti iz kojih atributi u relacijama uzimaju vrijednosti. Domene se mogu definirati i kao skupovi atomičkih vrijednosti, tako da se definira tip podataka kojeg će biti vrijednosti koje se nalaze u definiranoj domeni, [7].

Svaka relacija sastoji se od sheme relacije i instance relacije. Shema relacije naznačuje ime relacije, ime svakog polja, odnosno atributa, te domenu svakog polja. Instanca relacije je tablica, koja se sastoji od zapisa – skupa n-torki, u kojoj svaka n-torka posjeduje isti broj polja kao što je navedeno u shemi relacije. U instanci relacije, redak tablice predstavlja jednu n-torku, a stupac predstavlja polja navedena u shemi relacije. Broj polja navedenih u relacijskoj shemi predstavlja stupanj relacije, a broj n-torki naziva se kardinalitet instance relacije, [8]. Primjer relacije prikazan je slikom 8:



Slika 8: Primjer relacije, [8]

Relacijski model se koristi ključevima – kandidatskim, primarnim te stranim. Kandidatski ključ predstavlja jedinstveni identifikator (oznaku) n-torke, to je često kombinacija atributa pomoću koje je moguće jednoznačno referencirati jednu određenu n-torku, te se sastoji od minimalno jednog atributa, iako jedna n-torka može imati više kandidatskih ključeva. Primarni ključ je jedan kandidatski ključ preko kojeg je moguće jedinstveno označiti n-torku. Strani

ključ je skup atributa u jednoj relaciji čija vrijednost mora biti jednaka vrijednostima kandidatskih ključeva iz neke druge (povezane) relacije ili te iste relacije, [7].

2.1.2 Svojstva očuvanja integriteta

Svojstva očuvanja integriteta baze podataka sastoje se od ograničenja postavljenih na bazu podataka i podataka koji se unose u istu. Prema [3], ograničenja za očuvanje integriteta u relacijskom modelu mogu se podijeliti u tri kategorije:

- ograničenja korištenog modela,
- ograničenja zbog implementirane relacijske sheme, koja čine:
    - domenska ograničenja,
    - ograničenja na ključeve,
    - ograničenja na nepoznate (engl. „*null*“) vrijednosti,
    - ograničenja za očuvanja integriteta entiteta,
    - ograničenja za očuvanje referencijalnog integriteta,
- ograničenja postavljena u aplikacijskom rješenju.

Ograničenja korištenog modela podrazumijevaju sva ograničenja u implementaciji relacijskog modela, kao npr. svojstvo zabrane unosa dupliciranih n-torki.

Domenska ograničenja zahtijevaju da u instanci relacije, vrijednosti kojima se popunjavaju atributi za svaku n-torku proizlaze iz skupa vrijednosti definirane domene za taj stupac. Ovim ograničenjem domena polja se može razmatrati i kao tip tog polja, jer direktno utječe na tip vrijednosti koji se u isti može unositi, [8].

Ograničenja na ključeve pobliže definiraju problem dupliciranih podataka pomoću definiranog super-ključa (engl. „*superkey*“) koji predstavlja kombinaciju svih atributa n-torke. Navedeno ograničenje osigurava da se tako definirani ključevi za svaku n-torku tj. zapis u relaciji moraju razlikovati, odnosno da u relaciji ne smije postojati dva zapisa sa identičnim skupom atributa, [3].

Pomoću ograničenja za nepoznate vrijednosti moguće je definirati atribute, tj. domene kojima u instanci relacije zapisi mogu imati nepoznate vrijednosti, čime se obvezuje popunjavanje obaveznih polja, dok neka polja ne moraju imati unesenu vrijednost, [3].

Ograničenja za očuvanja integriteta entiteta i ograničenja za očuvanje referencijalnog integriteta bave se primarnim i stranim ključevima. Očuvanje integriteta entiteta nalaže da atribut n-torke na temelju kojega se kreira primarni ključ mora imati vrijednost, tj. vrijednost

ne smije biti nepoznata, dok se očuvanje referencijalnog integriteta vrši provjerom stranog ključa u tablici – strani ključ u relaciji mora biti valjana vrijednost koja postoji u relaciji koju referencira, [7].

Nad bazom podataka moguće je postaviti i razna druga ograničenja ovisno o domeni, vrijednostima ili drugim parametrima. Sva ograničenja koja nije moguće predvidjeti ili izvesti u bazi podataka, mogu se postaviti i na samom aplikacijskom rješenju, kroz provjere podataka prilikom unosa, izmjene ili brisanja, [3].

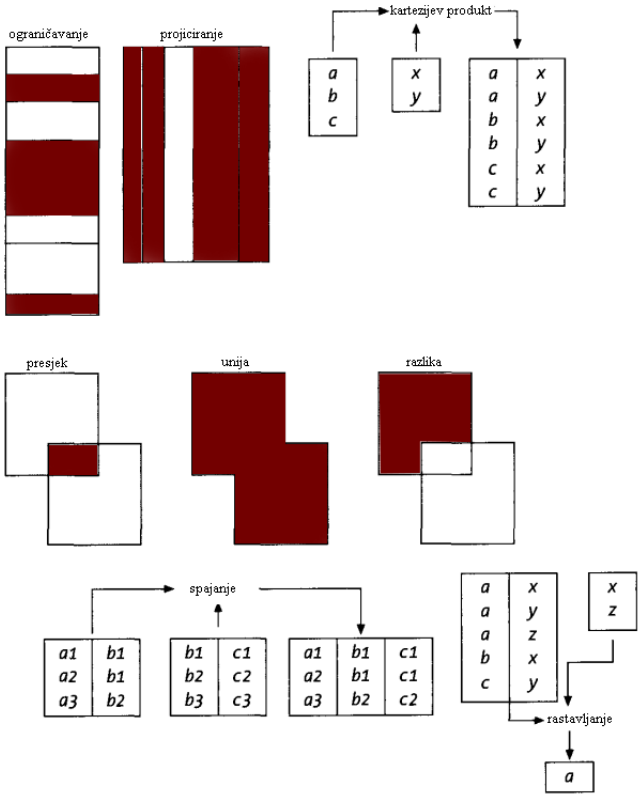
2.1.3 Manipulativna svojstva

Manipulativna svojstva relacijskog modela sastoje se od kombinacije relacijskih operacija, odnosno relacijske algebre i relacijskog dodjeljivanja koje omogućuje dodjelu rezultata izvršenja relacijskih operacija nad drugim relacijama – odnosno tvorbu novih relacija iz starih relacija. Svaka operacija izvršava se nad jednom ili više relacija, te kao rezultat daje isključivo jednu relaciju, čime je omogućeno slaganje višestrukih operacija – jedna operacija može na svom ulazu ili izlazu imati neku drugu operaciju. Codd je u začecima relacijskog modela definirao početnih osam operacija relacijske algebre:

- ograničavanje (engl. „*Restrict*“) – rezultira relacijom koja sadrži samo one n-torke iz željene relacije koje zadovoljavaju dani uvjet,
- projiciranje (engl. „*Project*“) – relacija koja sadrži sve n-torke iz početne relacije ali samo određene atribute tj. stupce,
- produkt (engl. „*Product*“) – znan kao kartezijev produkt ili spajanje te unakrsni produkt ili spajanje, te se kao takav može razmatrati i kao vrstom operacije spajanja – relacija koja sadrži sve moguće n-torke koji su kombinacija dviju n-torki, po jedna iz svake dane relacije,
- presjek (engl. „*Intersect*“) – relacija koja sadrži sve n-torke koje se nalaze u obje dane relacije, može se razmatrati kao vrsta operacije spajanja,
- unija (engl. „*Union*“) – relacija koja sadrži sve n-torke iz prve, druge ili obje dane relacije,
- razlika (engl. „*Difference*“) – relacija koja sadrži n-torke koje se nalaze u prvoj, ali ne i u drugoj danoj relaciji,
- spajanje (engl. „*Join*“) – relacija koja sadrži sve moguće n-torke koje su kombinacija n-torki iz danih relacija na način da svake dvije n-torke iz danih relacija

sadrže zajedničku vrijednost zajedničkog atributa te dvije relacije, te se u rezultatu pojavljuju samo jednom,

- rastavljanje (engl. „*Divide*“) – moguće izvršiti samo na kombinaciji jedne relacije višeg stupnja – više stupaca (atributa) i jedne relacije prvog stupnja – sa samo jednim stupcem, te kao rezultat daje relaciju koja se sastoji od svih vrijednosti jednog atributa u binarnoj relaciji koji odgovara svim vrijednostima u relaciji prvog stupnja – unarnoj relaciji, [7].

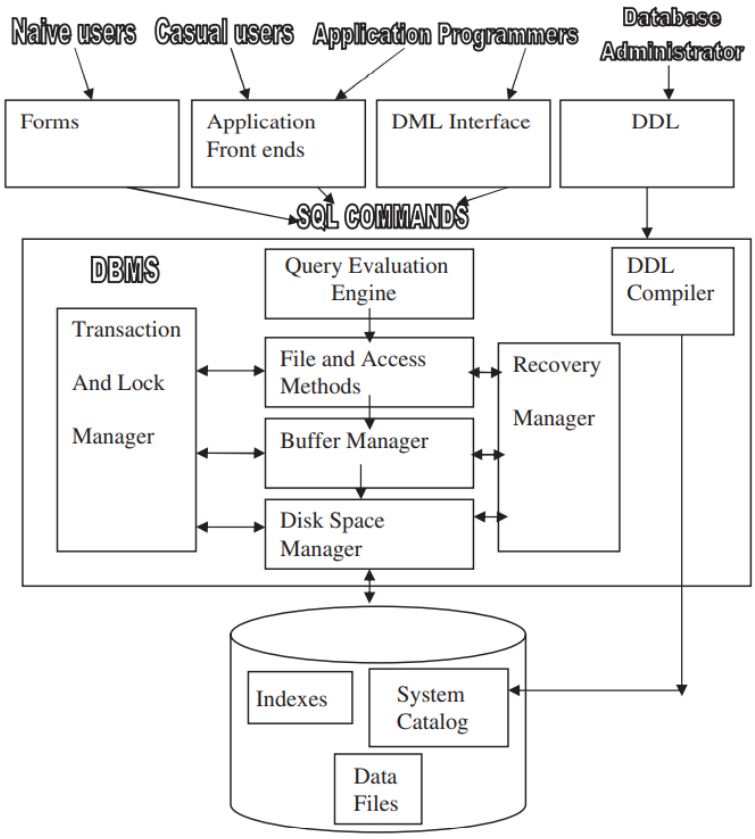


Slika 9: Operacije relacijske algebre, [7]

Na slici 9 vidljiva je vizualna reprezentacija navedenih operacija relacijske algebre. Operacijom ograničavanja izdvajaju se željeni redovi iz relacije nad kojom se izvršava, dok je za izdvajanje stupaca moguće koristiti operaciju projiciranja. Produkt kao rezultat daje novu relaciju koja sadrži sve moguće kombinacije iz dvije relacije koje se kombiniraju. Rastavljanje kao rezultat daje relaciju koja sadrži samo stupce i retke koji sadrže vrijednosti druge relacije, bez stupaca iz druge relacije. Operacije presjeka, unije i spajanja bave se potpunim spajanjem redaka i stupaca relacije, te izlazna relacija sadrži sve stupce iz obje relacije, a broj redaka ovisi o vrsti spajanja tj. o korištenoj operaciji relacijske algebre.

2.2. Sustav upravljanja bazom podataka

Sustavi upravljanja bazom podataka (SUBP) se mogu raščlaniti na pet općenitih komponenti: podaci, hardver, softver, procedure i korisnici. Sučelja od kojih se navedene komponente sastoje prikazana su na slici 10:



Slika 10: Komponente i sučelja SUBP, [1]

Komponente SUBP mogu se podijeliti na dvije kategorije ovisno o općenitoj funkciji koju u sustavu ispunjavaju: upravljanje pohranom i obrada upita. Upravljanje pohranom podrazumijeva pohranu, dohvat i ažuriranje podataka u bazi, a sastoji se od slijedećih komponenti:

- Upravitelj transakcijama – transakciju čini skup operacija koje ispunjavaju jednu logičku svrhu, upravljanje transakcijama nužan je element SUBP koji osigurava dosljednost podataka u SUBP uslijed pojave sustavnih greški, prekinutih transakcija, te u slučaju istovremenih interakcija i transakcija,
- Upravitelj ovlastima i integritetom – provjerava ovlasti korisnika koji pristupa podacima i uvjeta za očuvanjem integriteta podataka,

- Upravitelj datotekama – upravlja raspodjelom prostora za pohranu, vrši smještaj skupova podataka, odnosno zapisa, u datoteke i dohvat, ažuriranje i brisanje iz istih, te kreira i održava popis struktura i pokazatelja (engl. „*indexes*“ ili „*indices*“) definiranih u internoj shemi,
- Upravitelj međuspremnika (engl. „*Buffer manager*“) – za cilj ima povećati iskorištenje sustava pohrane te smanjiti potražnju za procesorskim resursima, glavni zadatak je dohvat podataka iz sustava pohrane i pohranjivanje istih, ili samo adrese istih u glavnu memoriju računala,
- Pokazatelji (engl. „*Indices*“) – pružaju brz pristup podacima koji sadrže određene vrijednosti pomoću popisa numeričkih vrijednosti koje daju redoslijed zapisa u određenoj tablici, [1].

Jezici korišteni za upravljanje i održavanje SUBP mogu se podijeliti na sljedeće kategorije, iako se u modernim sustavima koriste jezici koji obuhvaćaju operacije svih navedenih kategorija:

- Jezici za definiranje podatkovne strukture (engl. DDL – „*Data definition language*“) – za definiranje konceptualne i interne sheme baze podataka, te definiranje veza između istih,
- Jezici za definiranje strukture pohrane podataka (engl. SDL – „*Storage definition language*“) – za definiranje interne sheme u SUBP gdje su konceptualna i fizička razina strogo odvojene,
- Jezici za definiranje pogleda (engl. VDL – „*View definition language*“) – omogućuju definiranje korisničkih pogleda i veza istih s konceptualnom shemom, iako u većini SUBP se za isto koristi ranije navedeni DDL,
- Jezici za manipuliranje podacima (engl. DML – „*Data manipulation language*“) – sastoje se od operacija kao što su dohvaćanje, unos, brisanje i ažuriranje podataka, [3].

Baza podataka, tj. podatkovne datoteke i katalog sustava (engl. „*system catalog*“) tipično su pohranjeni u sustavu pohrane računala. Katalog SUBP-a, koji se još naziva i podatkovnim rječnikom (engl. „*data dictionary*“) sastoji se od ranije definiranih shema te shodno tome nije podložan čestim izmjenama, za razliku od samih podataka. Katalog može sadržavati informacije kao što su nazivi i veličine datoteka, nazivi i tipovi podataka podatkovnih stavki, detalje o pohrani svake datoteke, informacije o preslikavanju između shema, informacije o

postavljenim ograničenjima, i dr. informacije potrebne ostalim komponentama SUBP-a. Upravitelj podatkovnim prostorom (engl. „*Disk space manager*“ ili „*Stored data manager*“) rukuje pristupom sustavu pohrane, uz pomoć osnovnih operacija koje mu pruža pozadinski operacijski sustav, u svrhu prijenosa podataka iz sustava za pohranu u memoriju računala, kako bi se omogućila obrada istih ostalim komponentama i sučeljima SUBP-a. DDL prevoditelj obrađuje definiciju sheme koja mu je predana, te pohranjuje opise shema (meta-podatke) u katalog. Prevoditelj upita (engl. „*Query Compiler*“) preuzima korisničke upite više razine te ih raščlanjuje, analizira, interpretira, optimizira i prevodi u kôd za pristup podacima, kojeg SUBP izvršava, [3].

2.2.1 Pohrana podataka i optimizacija dohvata

B.M. Schueler rekao je: „Baza podataka nije baza podataka – transakcijski dnevnik (engl. „*transaction log*“ ili samo „*log*“) je baza podataka, a baza podataka je samo optimizirana pristupna putanja do aktualne verzije dnevnika.“, [9].

U relacijskim bazama podataka često korišten način organizacije podataka je korištenjem registara pokazatelja (indeksa) – registri se sastoje od kolekcija podatkovnih stavki, te sadrže mehanizme za učinkovito pretraživanje danih ključnih vrijednosti (engl. „*key value*“), bez da se prilikom pretrage prolazi kroz cijelu tablicu, odnosno kolekciju. Ključ za pretraživanje sačinjen je od jednog ili više atributa na temelju kojeg se potražuju zapisi unutar datoteke, [1].

Svaka tablica može sadržavati jedan ili više pokazatelja, a sami pokazatelji dijele se na:

- Primarni pokazatelj (grupirani) – zapisi se unutar datoteke sekvencijalno raspoređuju na temelju primarnog pokazatelja,
- Sekundarni pokazatelj (ne-grupirani) – ubrzavaju upite koji koriste ključeve za pretraživanje koji nisu obuhvaćeni primarnim pokazateljem,
- Gusti pokazatelj (engl. „*dense*“),
- Rijedak pokazatelj (engl. „*sparse*“),
- Bitovna karta (engl. „*bitmap*“),
- Jedno-stupanjski (engl. „*single level*“),
- Više-stupanjski (engl. „*multi-level*“), [1].

U relacijskim bazama podataka, najčešće korištena podatkovna struktura za pohranu indeksa je B-stablo i B+-stablo. Stablo, u terminologiji podatkovnih struktura, sastoji se od čvorova (engl. „*nodes*“). Svaki čvor, osim korijena, sastoji se od jednog roditeljskog čvora, te

može sadržavati i čvorove djecu. Čvor koji ne sadržava niti jedan dječji čvor naziva se završnim čvorom (engl. „*leaf node*“), a čvor koji nije završni naziva se unutrašnjim čvorom (engl. „*internal node*“). B-stabla pohranjuju pokazivače na podatkovni blok u unutrašnjim i završnim čvorovima, dok B+-stabla sadrže pokazivače samo u završnim čvorovima – što omogućuje veći kapacitet pokazivača, manje razina čvorova te brže brisanje. B i B+ stabla su uravnotežena stabla – što znači da su svi putevi od korijena do završnog čvora jednake duljine, [3].

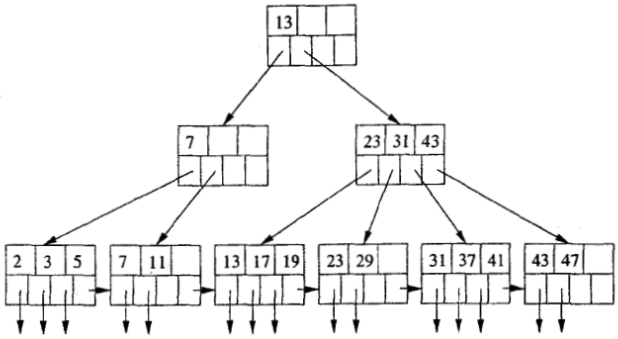
Glavne razlike između B i B+ stabla prikazane su u tablici 2.

Tablica 2: Razlike između B i B+-stabla [1]

B-stablo	B+-stablo
Unutrašnji čvorovi su veći od završnih čvorova	Unutrašnji i završni čvorovi su iste veličine
Brisanje zapisa u B-stablu je kompliciranо	Brisanje zapisa je jednostavno, jer se zapisi uvijek nalaze u završnim čvorovima
Pokazivači na podatkovne zapise nalaze se na svim razinama stabla	Pokazivači na podatkovne zapise nalaze se samo u završnim čvorovima

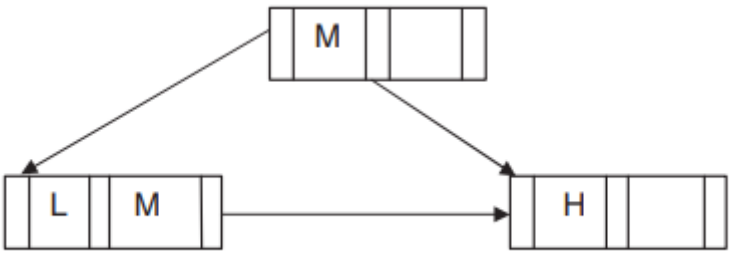
Popunjavanje B-stabla započinje s jednim korijenskim čvorom, na nultoj razini. Kada se korijenski čvor popuni sa *p-1* vrijednosti (gdje *p* predstavlja ukupan broj razina, odnosno red stabla), korijenski čvor se dijeli na dva čvora na prvoj razini – na način da se samo središnja vrijednost zadržava u korijenu, a ostale vrijednosti (s lijeve i desne strane) se dijele na druga dva čvora na razini jedan. Kada se popuni čvor koji nije korijenski, također se dijeli na dva nova čvora, ali u istoj razini u kojoj je taj čvor, a središnja vrijednost se pomiče u roditeljski čvor tog čvora, zajedno sa dva pokazivača na dva novo kreirana čvora. Ukoliko je roditeljski čvor pun, također se dijeli na već definirani način – te se takvo dijeljenje može propagirati sve do korijenskog čvora, prilikom čega će se kreirati nova razina, ukoliko se korijenski sloj podijelio, [3].

Vizualna reprezentacija B-stabla prikazana je slici 11, a dohvat iz istog vrši se uz pretpostavku da su vrijednosti indeksa pohranjene u stablu poredane, te da nema dupliciranih vrijednosti – što odgovara najčešćim slučajevima kod primarnih ključeva u relacijskim bazama podataka, [10].



Slika 11: B-stablo [10]

Popunjavanje B+-stabla vrši se na način da samo krajnji čvorovi sadrže pokazivač na zapis, dok svi ostali čvorovi sadrže pokazatelje na druge čvorove. Ukoliko je čvor prazan, prvo se popunjava lijeva strana, a ukoliko čvor sadrži neku vrijednost, vrijednosti u čvoru se slažu s lijeva na desno, od najmanje do najveće. Ukoliko je čvor popunjen, vrijednosti se klasificiraju kao najmanja, srednja i najveća vrijednost, te se čvor razdvaja, na način prikazan na slici 12, gdje *L* predstavlja najmanju vrijednost, *M* srednju vrijednost, a *H* najveću vrijednost, [1].



Slika 12: Razdvajanje čvorova u B+-stablu [1]

Prednosti korištenja B i B+-stabala u relacijskim bazama podataka su:

- Upotreba prostora za pohranu indeksa je uvijek veća od 50%, te se dinamički dodjeljuje i vraća, zbog čega ne dolazi do narušavanja razine usluge,
- Izrazito brz (logaritamski) nasumičan pristup podacima,
- Učinkovito ubacivanje i brisanje zapisa omogućuje održavanje poretka vrijednosti (ključeva) i uravnoteženost stabla, što omogućuje redoslijednu obradu vrijednosti te brz nasumičan dohvat,
- Omogućuju učinkovitu obradu većih grupa podataka – zbog održavanja poretka vrijednosti, [1].

2.2.2 Korisnici

Kompleksni sustavi mogu sadržavati sustave baza podataka kojima pristupa i čije podatke dijeli veliki broj korisnika sa različitim ulogama i potrebama. Korisnici baza podataka mogu se podijeliti na:

- administratore baze podataka – čiji je zadatak upravljanje resursima, kreiranjem i ažuriranjem pristupnih prava, te optimizacija sigurnosti i performansi rada baze podataka,
- dizajneri baze podataka – odgovorni za odabir odgovarajuće strukture podataka u bazi podataka na temelju zahtjeva i podataka koje je potrebno pohraniti, te kreiranje pogleda ovisno o potrebama grupa korisnika,
- krajnji korisnici – osoblje čiji rad zahtjeva pristup, pregled te ažuriranje podataka u bazi podataka, koji se nadalje mogu podijeliti na:
    - povremeni krajnji korisnici – povremeno pristupaju bazi te svaki put zahtijevaju različite podatke,
    - naivni ili parametarski korisnici – najveća grupa korisnika čiji posao zahtjeva konstantno potraživanje upita iz baze koristeći standardizirane tipove upita i ažuriranja – tzv. transakcije tipa crne kutije (engl. „*canned transactions*“),
    - sofisticirani krajnji korisnici – korisnici koji su upoznati sa radom i strukturom sustava baze podataka te samostalno stvaraju upite,
    - samostalni korisnici – održavaju osobne baze podataka koristeći izrađene programske pakete, [3].

2.2.3 SUBP i relacijski model

Kako bi se sustav upravljanja bazom podataka smatrao relacijskim, mora zadovoljavati određeni skup dvanaest pravila koja je 1985. godine definirao tvorac relacijskog modela, Dr. Edgar F. Codd. Općenito pravilo nalaže da SUBP more sadržavati potpun skup funkcionalnosti za kompletno upravljanje podacima kroz mogućnosti relacijskog modela. SUBP se smatra relacijskim ukoliko zadovoljava općenito pravilo te barem šest od sljedećih, [1]:

1. Pravilo prikaza informacija (engl. „*The Information Rule*“) – Sav informacijski sadržaj u relacijskoj bazi podataka predstavlja se izričito na logičkoj razini i to isključivo na jedan način – kao vrijednosti u tablicama,

2. Pravilo zajamčenog pristupa (engl. „*Guaranteed Access Rule*“) – u relacijskoj bazi podataka, pristup svakom podatku mora biti zajamčen na logičkoj razini uporabom kombinacije naziva tablice, primarnog ključa i naziva stupca,
3. Sustavno postupanje prema nepoznatim vrijednostima (engl. „*Systematic Treatment of Null Values*“) – nepoznate vrijednosti (različite od praznog skupa znakova ili nule) moraju biti podržane u relacijskom SUBP-u za prikaz informacijskog sadržaja, koji nedostaje ili je neprimjenjiv, na sistematičan način neovisno o tipu podatka,
4. Dinamični on-line katalog temeljen na relacijskom modelu (engl. „*Dynamic On-line Catalog Based on the Relational Model*“) – opis baze podataka mora biti predstavljen na logičkoj razini na isti način kao i podaci, kako bi se za kreiranje upita nad shemom mogao primijeniti isti model kao i na podacima – relacijski model,
5. Pravilo sveobuhvatnog podatkovnog jezika (engl. „*Comprehensive Data Sublanguage Rule*“) – relacijski sustav može podržavati više jezika i načina korištenja, ali mora postojati barem jedan jezik koji je izražajan, definiran sintaksom kao niz teksta te koji podržava operacije definiranja podataka, definiranja pogleda, rukovanja podacima, postavljanja transakcijskih ograničenja i ograničenja koja osiguravaju integritet sustava,
6. Pravilo ažuriranja pogleda (engl. „*View Updating Rule*“) – svi pogledi koji se u teoriji mogu ažurirati, moraju se moći ažurirati i kroz SUBP – podaci se moraju moći mijenjati kroz sve poglede dostupne korisniku,
7. Pravilo konceptualnog rukovanja podacima (engl. „*High-level Insert, Update, and Delete*“) – SUBP mora omogućiti rukovanje relacijama kao da su jedan operand, odnosno omogućiti dohvat, ubacivanje, izmjenu i brisanje podataka kroz jednu naredbu,
8. Neovisnost fizičkih podataka (engl. „*Physical Data Independence*“) – aplikacije i ostale programske implementacije moraju ostati logički netaknute u slučaju izmjene načina pohrane ili pristupa podacima,
9. Neovisnost logičkih podataka (engl. „*Logical Data Independence*“) – aplikacije i ostale programske implementacije moraju ostati logički netaknute u slučaju izmjene strukture baze podataka i njenih tablica,
10. Neovisnost cjelovitosti (engl. „*Integrity Independence*“) – ograničenja za očuvanje cjelovitosti (integriteta) sustava moraju se moći definirati kroz relacijski podatkovni

jezik, te pohraniti u katalog baze podataka, a ne u aplikaciju ili drugu programsku implementaciju.

11. Neovisnost distribucije (engl. „*Distribution Independence*“) – SUBP mora omogućiti distribuciju podataka kroz više sustava, bez da korisnik mora biti svjestan iste,
12. Pravilo subverzije[2] (engl. „*Non-subversion Rule*“) – ukoliko relacijski sustav koristi i jezik niže razine, taj jezik se ne smije koristiti na način koji bi narušio ograničenja za očuvanje cjelovitosti definirana relacijskim jezikom više razine.

Neki od najpoznatijih primjera relacijskih SUBP rješenja, ili kompanija su: Oracle, MySQL, Microsoft SQL Server, PostgreSQL, IBM DB2, Microsoft Access, SQLite, MariaDB, IBM Informix te Azure SQL, [11].

2.3. Strukturirani jezik upita

Strukturirani jezik upita – SQL (engl. „*Structured Query Language*“) je posredni jezik za komunikaciju s relacijskim SUBP-om, koji je danas de-facto standard za razne implementacije relacijskih baza podataka. Sastoji se od naredbi pisanih kao izjave na engleskom jezikom, što ga čini lakim za shvaćanje, pisanje i učenje. Osnovne naredbe omogućuju dohvat, unos, ažuriranje i brisanje podataka. SQL je ne-proceduralni jezik – što znači da pomoću istog nije potrebno definirati na koji način se podaci dohvaćaju, već samo koje podatke je potrebno dohvatiti, [1].

Povijest SQL standarda počinje 1970. godine – razvojem koncepta relacijskog modela te iz sve većeg porasta interesa i istraživanja prema implementaciji istog u raznim sustavima, dolazi do potrebe za specifikacijom zahtjeva i ciljeva koje univerzalni relacijski jezik mora ispunjavati. Relacijski jezik je jezik koji mora ispunjavati neke ili sve mogućnosti relacijskog modela. Od više razvijenih jezika najviše se istaknuo SEQUEL (engl. „*Structured English Query Language*“) razvijen od strane IBM-a 1974. godine. Kroz iduće godine naziv SEQUEL izmijenjen je u SQL iz pravnih razloga, te je 1976.-1977. objavljena nadopuna istog zvana SEQUEL/2. 1986. godine. Važnost uspostave norme za relacijske jezike prepoznalo je i američko standardizacijsko tijelo ANSI, čime IBM-ov SQL, uz manje preinake, postaje standardnim relacijskim jezikom, a 1987. godine i međunarodnim standardom proglašenim od

[2] Subverzija – rušenje važećih vrijednosti ili postojećeg stanja u društvu iznutra, lat. subversio ≃ subversivus: prevratnički ← subvertere: zbaciti, srušiti [50]

strane Međunarodne organizacije za standardizaciju ISO (engl. „*International Organization for Standardization*“), [12].

Idućih godina slijedile su razne nadopune i revizije SQL standarda, od kojih su najveće revizije bile 1992. godine kao SQL2 ili SQL/92, 1993. – SQL3, te SQL99 objavljen 1999. godine. Posljednja inačica standarda objavljena je 2016. godine, [13].

SQL je sačinjen od više zasebnih grupa funkcionalnosti, [8]:

- Jezik za definiranje podatkovne strukture[3] – DDL – podrška za kreiranje, brisanje i izmjenu definicija (shema) tablica i pogleda, te postavljanje ograničenja za očuvanje integriteta,
- Jezik za manipuliranje podacima – DML – upiti za dohvaćanje, unošenje, izmjenu i brisanje redaka,
- Okidači i napredna ograničenja za očuvanje integriteta – podrška za izvršavanje radnji ovisno o događajima (izmjenama) u SUBP definiranim u uvjetima okidača,
- Ugrađeni i promjenjivi (dinamični) SQL – ugrađeni SQL omogućuje izvršavanje SQL programskog kôda iz programskog jezika računala kao što je C ili COBOL, dok promjenjivi SQL omogućuje kreiranje i izvršavanje upita u vremenu izvršavanja (engl. *run-time*),
- Udaljeni pristup bazi i veza klijent-poslužitelj – funkcionalnosti preko kojih je moguće definirati na koji se način poslužiteljska aplikacijska implementacija povezuje na poslužitelj SQL baze podataka, te na koji se način pristupa podacima preko mreže,
- Upravljanje transakcijama – omogućuje upravljanje načinom na koji se transakcije u SUBP-u obrađuju i izvršavaju,
- Sigurnost – mehanizmi za upravljanje korisničkog pristupa podacima, odnosno tablicama i pogledima,

[3] Prema ANSI definiciji SQL standarda, DDL se naziva i jezikom za definiranje sheme (engl. „*Schema Definition Language*“) [51]

- Napredne funkcionalnosti – objektno-orijentirane funkcionalnosti, rekurzivni upiti, upiti potpore odlučivanju, funkcionalnosti podrške rudarenju podataka, upravljanje prostornim, tekstualnim i XML[4] (engl. „*Extensible Markup Language*“) podacima.

Definiranje podatkovne strukture sastoji se od SQL naredbi za kreiranje, brisanje i izmjenu tablica, njenih stupaca i ograničenja na iste. Pojam tablica podrazumijeva osnovne tablice, tablice izvedene iz osnovnih tablica, poglede – imenovane izvedene tablice te tablice i poglede koje koriste grupiranje podataka po vrijednostima. Prilikom kreiranja i izmjena stupaca tablice, potrebno je definirati tip podatka (domenu) stupca. [14]

Prema ISO/ANSI standardu, tipovi podataka korišteni u SQL-u mogu se klasificirati kao:

- unaprijed definirani – tipovi podataka određeni standardom, mogu sadržavati isključivo atomične vrijednosti – navedeni u Tablica 3,
- izvedeni – tipovi podataka koji se sastoje od više atomičnih vrijednosti, ili više kompozitnih vrijednosti – vrijednosti sastavljenih od nula ili više vrijednosti određenog tipa,
- korisnički definirani – tipovi podataka imenovani i definirani od strane korisnika u shemi baze podataka,

Unaprijed definirani tipovi podataka i njihove karakteristike kao npr. veličina podataka koje isti mogu pohranjivati, ovise o tehničkoj implementaciji SUBP-a, te mogu sadržavati više od navedenih unaprijed definiranih tipova podataka u tablici 3, [15].

Tablica 3: Predefinirani tipovi podataka u SQL standardu prema ISO 9075-1:2016, [15]

Tekstualni nizovi	Određene duljine	CHARACTER	Niz znakova duljine n
		NATIONAL CHARACTER	UNICODE niz znakova duljine n
	Promjenjive duljine	CHARACTER VARYING	Niz znakova varijabilne duljine
		NATIONAL CHARACTER VARYING	UNICODE niz znakova varijabilne duljine

[4] XML – engl. „*Extensible Markup Language*“ je format za prikaz podataka i struktura podataka koji mogu imati različite oblike i vrijednosti (engl. „*arbitrary data*“), te koji nisu ograničeni određenim formatom ili strukturom, XML definira set pravila za prikaz takvih podataka na način čitljiv i ljudima i strojevima, [52].

Brojčani podaci	Egzaktan broj	DECIMAL	Decimalni broj sa mogućnosti određivanja preciznosti i skale
		INTEGER	Cijeli broj
	Približan broj	BINARY FLOATING POINT	Približne decimalne vrijednosti, sa mogućnosti određivanja preciznosti i skale
		DECIMAL FLOATING POINT	Približne decimalne vrijednosti, sa mogućnosti određivanja preciznosti i skale
Datum i vrijeme	Samo datum	DATE	Sadrži vrijednosti godina, mjesec i dan
	Samo vrijeme	TIME	Sadrži vrijednost vremena u danu
	Datum i vrijeme	TIMESTAMP	Sadrži vrijednosti godine, mjeseca, dana, sata, minute i sekunde
Intervali	Dugački intervali	INTERVAL YEAR, INTERVAL MONTH, INTERVAL YEAR TO MONTH	Sadrži vrijednost koja predstavlja razliku godina, mjeseci ili godine i mjeseci između dva datuma

	Kratki intervali	INTERVAL DAY, INTERVAL HOUR, INTERVAL MINUTE, INTERVAL SECOND, INTERVAL DAY TO HOUR, INTERVAL DAY TO MINUTE, INTERVAL DAY TO SECOND, INTERVAL HOUR TO MINUTE , INTERVAL HOUR TO SECOND, INTERVAL MINUTE TO SECOND	Sadrži vrijednost koja predstavlja razliku dana, sati, minuta, sekundi ili kombinacije navedenih između dva datuma
Booleov tip	Istina / laž	BOOLEAN	Vrijednost koja može biti istina ili laž
XML tip		XML	Tip za pohranu XML objekata
Binarni	Binarni niz znakova	BINARY STRING TYPE	Sadrži vrijednost niza okteta, može biti određene ili neodređene veličine
		BINARY LARGE OBJECT STRING	Sadrži vrijednost niza okteta do maksimalne veličine definirane implementacijom

Osim definicija podatkovnih tipova, DDL sadrži i naredbe *CREATE*, *ALTER* i *DROP* – čije se funkcionalnosti odnose na sljedeće strukture, [1]:

- tablice,
- poglede – virtualne tablice,
- brojevne nizove – cijeli broj koji varira ovisno o danoj konstantnoj vrijednosti,
- okidače – okidači izvršavaju određene naredbe kada su zadovoljeni određeni uvjeti,
- pokazivače (indekse) – korišteni za optimizaciju dohvata podataka.

U SQL jeziku dostupne su četiri osnovne naredbe za manipuliranje podacima: *SELECT*, *INSERT*, *UPDATE* i *DELETE*.

2.4. *NoSQL*, *NewSQL* i *Cloud* implementacije

Baze podataka generalno ispunjavaju dva tipa radnih zadataka – mrežnu obradu transakcija (engl. OLTP – „*On-Line Transactional Processing*“) ili mrežnu analitičku obradu (engl. OLAP – „*On-Line Analytical Processing*“) čije razlike i zahtjevi su prikazani u tablici 4, iz koje se može zaključiti da su OLTP sustavi uglavnom sustavi usmjereni na ažuriranje podataka i zapis novih podataka, dok su OLAP sustavi usmjereni na dohvat i obradu velike količine povijesnih podataka za analitičke svrhe, [16].

Tablica 4: Razlika između sustava namijenjenih za OLTP i OLAP zadatke, [16]

OLTP	OLAP
Izvršavanje velikog broja transakcija od strane velikog broja korisnika u stvarnom vremenu	Izvršavanje upita nad velikom količinom podataka, nerijetko i nad svim podacima u bazi, u analitičke svrhe
Zahtjeva veoma kratko vrijeme odziva	Vrijeme odziva može biti znatno veće nego što je zahtijevano kod transakcija
Česta izmjena manje količine podataka, podjednaka količina čitanja i pisanja u bazu	Zadaci uglavnom podrazumijevaju samo čitanje iz baze, bez izmjene postojećih podataka u istoj
Koristi se pokazivačima kako bi se smanjilo vrijeme odziva	Pohranjuje podatke u stupčastom formatu kako bi se olakšao pristup većem broju zapisa
Zahtjeva česte izrade sigurnosnih kopija	Sigurnosne kopije mogu se izrađivati znatno rjeđe

Mali zahtjevi nad prostorom pohrane	Zbog pohrane velike količine povijesnih podataka, zahtjeva veliku količinu prostora za pohranu
Upiti najčešće uključuju samo nekoliko zapisa	Upiti su često vrlo kompleksni te uključuju veliki broj zapisa

Za navedene zahtjeve OLTP sustava, razvijene su relacijske baze podataka. Osim što omogućavaju i podržavaju navedeno u tablici 4, zadovoljavaju i ACID zahtjeve, [17]:

- Atomarnost (engl. „***A**tomicity*“) – svaka operacija u transakciji (čitanje, pisanje, ažuriranje ili brisanje podataka) tretira se kao zasebna jedinica, prilikom izvršavanja transakcije izvršiti će se sve zahtijevane operacije, u protivnom se neće izvršiti niti jedna,
- Konzistentnost (engl. „***C**onsistency*“) – osigurava da transakcije čine predvidljive i predefinirane izmjene nad tablicama, te da se u slučaju pojave grešaka unutar podataka ne narušava integritet tablice,
- Izolacija (engl. „***I**solation*“) – ukoliko više korisnika istovremeno dohvaća ili ažurira podatke u tablici, izolacija transakcija osigurava da istodobne transakcije međusobno ne utječu jedna na drugu, te da se izvršavaju redoslijedno,
- Trajnost (engl. „***D**urability*“) – osigurava da se izmjene podataka definirane u izvršenoj transakciji pohrane u bazi podataka, čak i u slučaju sustavnih grešaka.

OLAP sustavi usmjereni su na analizu velike količine poslovnih podataka prikupljenih iz više različitih izvora, u svrhu pružanja pomoći prilikom strateškog planiranja. Za potrebe agregiranja podataka koji će se koristiti u OLAP sustavima, korišteni su podatkovna skladišta (engl. „*data warehouse*“) – koja sadrže strukturirane podatke prikupljene iz različitih izvora, uključujući i relacijske baze podataka, [18].

Dolaskom ere „Big Data“[5] – koju klasificiraju veliki, kompleksni i nestrukturirani skupovi podataka – podatkovna skladišta postaju podatkovnim jezerima (engl. „*data lake*“). Podatkovna jezera sadržavaju sve funkcionalnosti i mogućnosti podatkovnih skladišta, ali uz dodatak mogućnosti pohrane podataka u njihovom ne-strukturiranom, sirovom obliku, [19].

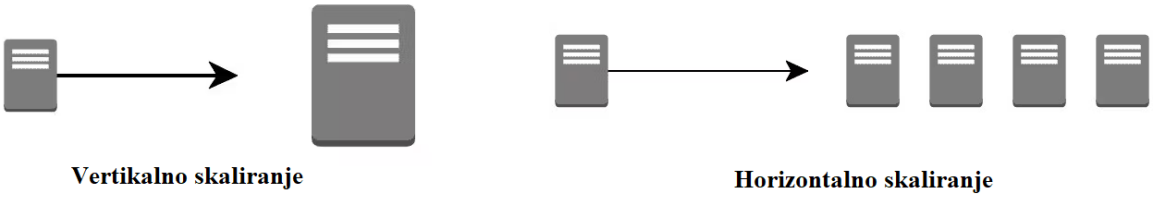
[5] „Big Data“ pojam može se definirati pomoću tri slova V – podaci imaju određenu raznolikost (engl. „***V**ariety*“), veliki opseg i veličinu (engl. „***V**olume*“) te nastaju znatno brže (engl. „***V**elocity*“), [56].

Trend porasta povećanja veličine, volumena i brzine generiranja podataka uzrokovao je potrebu za novim konceptom baze podataka – NoSQL baze podataka. NoSQL tumači se kao „*non-SQL*“, „*non-relational*“ odnosno „*not only SQL*“, što karakterizira NoSQL baze podataka kao ne-relacijske baze, koje mogu pohranjivati više tipova strukturiranih ili ne-strukturiranih podataka. Osnovi tipovi koje je moguće pohraniti u takve baze podataka su dokumenti, parovi ključeva i vrijednosti, grafikoni, i „*wide column*“ tipovi – koji mogu sadržavati tablice, redove, i dinamičke stupce. Dok je osnova pohrane podataka u relacijskim bazama podataka definicija sheme, u NoSQL baze podataka moguće je pohranjivati podatke bez definiranja sheme istih, [20].

NoSQL baze podataka optimizirane su specifično za slučajeve upotrebe sa velikom količinom podataka, niskim vremenom odgovora, te fleksibilnim podatkovnim strukturama, odnosno modelima, [21]. Poznate implementacije NoSQL baza podataka su: MongoDB, Apache Cassandra, Apache CouchDB, ScyllaDB, Redis, Neo4j, Oracle NoSQL, ArangoDB, Riak, Aerospike, CouchBase, MySQL Cluster i dr.

Prednosti koje omogućuje NoSQL naspram tradicionalnih relacijskih baza podataka, navedene u [20], su:

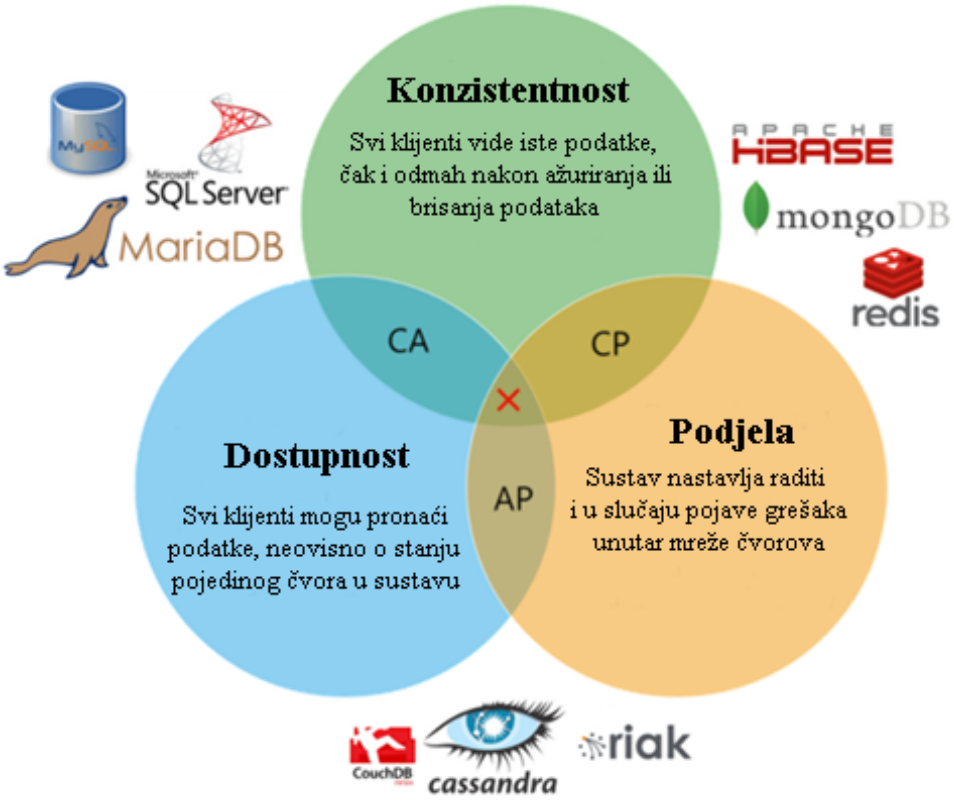
- Fleksibilna shema – prilikom kreiranja tablica u relacijskoj bazi podataka, potrebno je definirati shemu istih, dok u NoSQL bazama podataka nije potrebno da sve podatke unutar skupa podataka definira ista shema niti da imaju iste atribute,
- Horizontalno skaliranje – prilikom povećanja opsega korištenja sustava, u relacijskom modelu, jedina mogućnost skaliranja je vertikalno skaliranje – povećanje procesorske moći poslužitelja, u NoSQL sustavima, osim vertikalnog skaliranja moguće je i horizontalno skaliranje – dodavanje čvorova poslužitelja te raspodjela opterećenja između istih, prikazano slikom 13,
- Brzi upiti nad velikom količinom podataka – veze između tablica u relacijskom modelu, u NoSQL sustavima izvedene su denormalizirano – na način da se svi podaci iz povezanih tablica ugnježđuju u sam traženi podatak, čime se smanjuje broj operacija na bazi podataka.



Slika 13: Razlika vertikalnog skaliranja (lijevo) i horizontalnog skaliranja (desno), [22]

Horizontalno skaliranje sustava čini sustav baze podataka distribuiranim, te je kao takav podložan CAP teoremu (slika 14), koji diktira kako distribuirana baza podataka može isporučiti samo dva od sljedeća tri svojstva, prikazano , [23]:

- Konzistentnost (engl. „***C**onsistency*“) – prilikom dohvata podataka, svi čvorovi sadrže isti podatak, odnosno podatak zadnje operacije pisanja,
- Dostupnost (engl. „***A**vailability*“) – svaki zahtjev mora dobiti odgovor u 100% slučajeva, neovisno o stanju pojedinog čvora u sustavu,
- Tolerancija na podjelu (engl. „***P**artition Tolerance*“) – sustav mora funkcionirati neovisno o kašnjenju poruka između čvorova, replikacijom podataka na više čvorova osigurava se rad sustava unatoč stanju pojedinog čvora.



Slika 14: CAP teorem uz primjer baza podataka koje žrtvuju jedno od tri svojstva, [24]

Dok se relacijske baze podataka definiraju kroz ranije navedene ACID zahtjeve, NoSQL baze podataka najčešće prate BASE model, koji se temelji na svojstvima dostupnosti i tolerancije na podjelu preuzetih iz CAP teorema, žrtvujući trenutnu konzistentnost, [25]:

- Načelna dostupnost (engl. „***B**asically **A**vailable*“) – umjesto primoravanja trenutne konzistentnosti, NoSQL baze podataka garantiraju dostupnost podataka replikacijom i rasprostranjivanjem istih na više čvorova u grupi čvorova baze podataka, te garantiraju odgovor na svaki upit,
- Meko stanje (engl. „***S**oft State*“) – zbog nedostatka garancije trenutne konzistentnosti, dohvati podataka s vremenom mogu biti različiti za isti podatak, odgovornost konzistentnosti podataka se delegira iz sustava baze podataka na aplikacijski sloj,
- Krajnja konzistentnost (engl. „***E**ventually Consistent*“) – iako su podaci u sustavu u datom trenutku nekonzistentni, s vremenom će sustav ostvariti konzistentnost.

NewSQL baze podataka su moderne relacijske baze podataka koje pružaju mogućnosti horizontalnog skaliranja kao i NoSQL baza podataka za OLTP primjene, uz zadržavanje principa relacijskog modela, te zadovoljavanje ranije navedenih ACID zahtjeva. NewSQL baze mogu se razmatrati kao najbolja moguća kombinacija svojstava tradicionalnih relacijskih (SQL) baza i ne-relacijskih (NoSQL) baza. Dok SQL baze koriste ACID – model najsnažnije razine konzistentnosti, te NoSQL baze koriste BASE, model slabe razine konzistentnosti, NewSQL baze koriste model između ACID i BASE – BASIC, [24]:

- Načelna dostupnost (engl. „***B**asic **A**vailability*“) – sustav uvijek odgovara na upite,
- Skalabilnost (engl. „***S**calability*”) – mogućnost dodavanja resursa u slučaju povećanja opterećenja,
- Trenutna konzistentnost (engl. „***I**nstant **C**onsistency*”) – osigurava da prilikom uzastopnih operacija pisanja i čitanja, podaci dohvaćeni operacijom čitanja moraju biti jednaki podacima zapisanim operacijom pisanja.

Sažetak razlika u svojstvima SQL, NoSQL i NewSQL baza podataka prikazan je tablicom 5:

Tablica 5: Usporedba svojstava SQL, NoSQL i NewSQL baza podataka, [24]

Svojstvo	Tradicionalne relacijske baze podataka (SQL)	Ne-relacijske baze podataka (NoSQL)	NewSQL baze podataka
Relacijski model	DA	NE	DA
Konzistentnost podataka	Visoka razina konzistentnosti ACID	Niska razina konzistentnosti BASE	Visoka ili viša razina konzistentnosti ACID / BASIC
Skaliranje	Vertikalno	Horizontalno	Horizontalno
Kompleksnost upita	Niska	Visoka	Vrlo visoka
SQL podrška	DA	NE	DA

Implementacije NewSQL baza podataka su: MariaDB Xpand – poznat i kao Clustrix, NuoDB, CockroachDB, Pivotal GemFire XD, Altibase, SingleStoreDB, VoltDB, c-treeACE, Percona TokuDB, Apache Trafodion, TIBCO ActiveSpaces, ActorDB, TiDB, YugabyteDB, i dr.

Razvojem NoSQL i NewSQL baza podataka, odnosno baza podataka sa mogućnošću distribucije, skaliranja i pohrane strukturiranih ili ne-strukturiranih podataka, napravljen je uvod u prebacivanje baza podataka sa lokalne infrastrukture, na *Cloud* rješenja. Cloud tehnologija omogućuje odvajanje sloja pohrane od računalnog sloja – moguće je zasebno skalirati veličinu pohrane i same performanse sustava, [26]. Primjeri Cloud rješenja baza podataka su: Oracle MySQL HeatWave, Amazon DynamoDB, Google BigTable, Google AlloyDB, Amazon Aurora, Google Spanner, MongoDB Atlas, Azure Cosmos DB, i dr.

3. Prostorno-vremenske baze podataka

3.1. Vremenske baze podataka

Vremenske baze podataka, ili vremenski sustavi baza podataka su sustavi koji sadržavaju posebnu podršku za pohranjivanje, dohvat i ažuriranje povijesnih i/ili budućih podataka. Uobičajeni SUBP-ovi namijenjeni su pohrani i obradi podataka o trenutnom stanju, te je implementacija vremenskih podataka unutar istih znatno kompleksnija, a dohvati željenih podataka unutar istog spori i ograničenih mogućnosti – takvi sustavi mogu se nazvati i ne-vremenskim bazama podataka, [9].

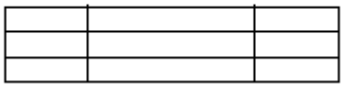
Promatranje i bilježenje vremena vrši se kroz tri dimenzije, [27]:

- Korisničko vrijeme – korisnička interpretacija vremenske vrijednosti,
- Valjano vrijeme (engl. „*valid time*“) – kada je činjenica bila istinita u modelu stvarnosti,
- Transakcijsko vrijeme – kada se činjenica pohranila u bazu podataka.

Podrška za navedene dimenzije vremena ovisi o implementaciji SUBP-a, gdje je najčešće podržano korisničko vrijeme, dok se ostale dimenzije obrađuju kroz aplikacijski sloj. Potreba za različitim definicijama vremena proizlazi iz činjenice da je u tradicionalnim bazama podataka moguće mijenjati sve dostupne podatke – jer takva baza podataka predstavlja trenutno stanje, dok u vremenskim bazama podataka mogu postojati podaci iz prošlosti ili budućnosti, čije izmjene ne bi smjele biti moguće, ukoliko nisu u valjanom vremenu, [9].

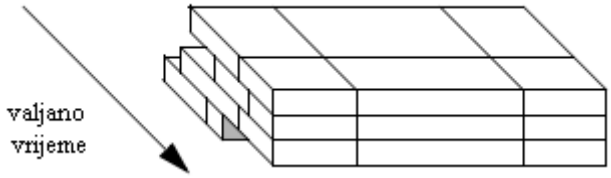
Ovisno o podršci SUBP-a prema određenim dimenzijama promatranog vremena, prema [28], baze podataka mogu se podijeliti na:

- Trenutne baze (engl. „*snapshot*“) – tradicionalne baze podataka koje sadrže isključivo podatke koji opisuju trenutno stanje sustava (slika 15),



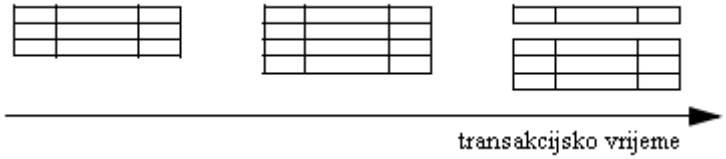
Slika 15: Vizualni prikaz relacije trenutnog vremena, odnosno *snapshot* relacije, [28]

- Povijesne baze (engl. „*valid-time*“ ili „*historical*“) – baze podataka koje omogućuju rukovanje valjanim vremenom, te mogu sadržavati podatke o prijašnjem, trenutnom i budućem stanju (slika 16),



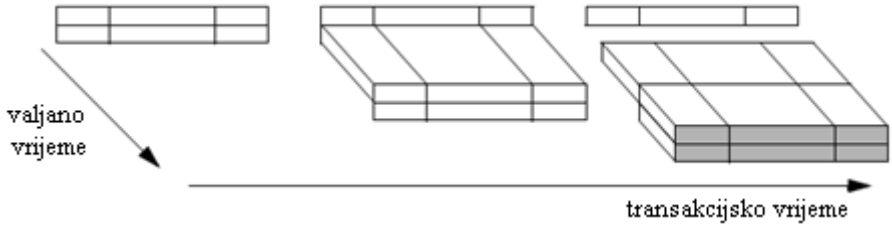
Slika 16: Vizualni prikaz relacije valjanog vremena, [28]

- Transakcijske baze (engl. „*transaction-time*“ ili „*rollback*“) – baze koje rade s podacima isključivo u transakcijskom vremenu – vremenu kad je zapis izmijenjen odnosno u vremenu u kojem postoji određeno stanje baze podataka (slika 17),



Slika 17: Vizualni prikaz relacije u transakcijskom vremenu, [28]

- Bi-temporalne baze (engl. „*bitemporal*“) – baze koje podržavaju rad sa valjanim i transakcijskim vremenom (slika 18),



Slika 18: Vizualni prikaz bi-temporalne relacije, [28]

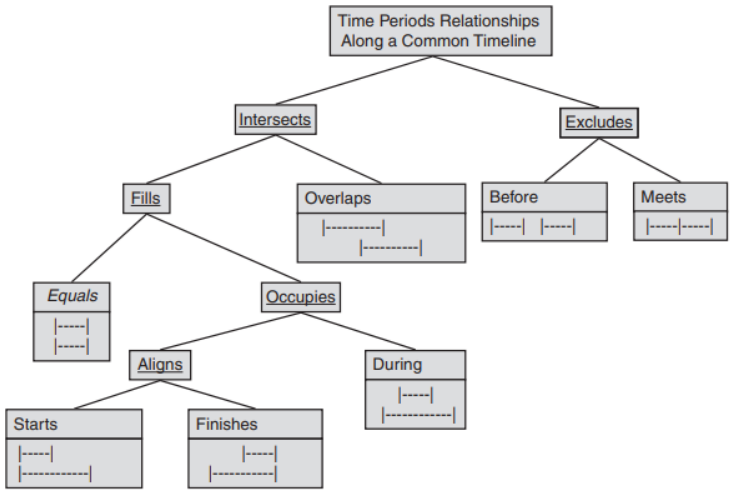
- Temporalne baze – baze koje podržavaju sve navedene tipove vremena, tj. ranije navedene relacije.

Postoje tri osnovna tipa vremenskih podataka, [27]:

- Trenutak, ili točka (engl. „*instant*“ ili „*point*“) – trenutak u vremenu, npr. 19. travnja 2023. godine u 15:10:30,
- Interval – određena duljina vremena, npr. tri mjeseca,
- Period – određeno trajanje između dva vremena, npr. od siječnja do travnja 2023.

James F. Allen 1983. godine koncipirao je pojam intervalne algebre, te je definirao sve moguće relacije dva vremenska intervala na istoj vremenskoj liniji, prikazane na slici 19. Za

takva dva vremenska perioda, ukoliko se dodiruju u barem jednoj točki, može se reći da se sijeku (engl. „*intersect*“), a ukoliko ne, onda se isključuju (engl. „*exclude*“). Ukoliko je utvrđeno da se sijeku, mogu se ispunjavati (engl. „*fills*“), odnosno preklapati (engl. „*overlaps*“). Ukoliko prvi u potpunosti ispunjava drugog, i obratno, onda su međusobno jednaki (engl. „*equals*“), a ukoliko se ne ispunjavaju u potpunosti, jedan zauzima (engl. „*occupies*“) drugog (ili obratno) – te mogu biti ili poravnati (engl. „*aligns*“) – po početku (engl. „*starts*“) ili kraju (engl. „*ends*“) ili jedan može biti usred drugoga (engl. „*during*“). Ukoliko je utvrđeno da se ne dodiruju već isključuju, tada je jedan prije drugoga (engl. „*before*“), ili im se krajnja i početna točka dodiruju (engl. „*meets*“). Navedene engleske termini za objašnjene temporalne odnose također predstavljaju i naredbe koje je moguće koristiti u temporalnim SUBP-ovima nad periodnim tipovima podataka, [29].



Slika 19: Moguće relacije dva vremenska intervala na istoj vremenskoj liniji, [29]

Iako je već 1992. godine u SQL92 standard dodana podrška za vremenske tipove podataka, ista nije bila dovoljno opširna te se iz tog razloga razvio dodatak na SQL standard, koji se isključivo bavi dohvatom i obradom temporalnih podataka, nazvan TSQL 2 – *Temporal Query Language*, uz par definiranih pravila nad istim, [30]:

- Mora biti relacijski jezik upita,
- Mora biti dosljedan postojećim standardima – SQL-92,
- Nije namijenjen da sam postane standardom, već je osnovica za daljnja istraživanja i implementacije,
- Nije objektno-orijentirani jezik upita,

- Mora biti sveobuhvatan kao i SQL, te sadržavati sve funkcionalnosti SQL-a, ali uz ključnu riječ „*temporal*“ koja označava da se naredba odnosi na temporalan tip podatka,
- Mora podržavati valjano i transakcijsko vrijeme,
- Mora sadržavati odgovarajuću algebru kao pozadinu operacija,
- TSQL2 je jezični plan – ne smije se baviti implementacijskim svojstvima SUBP-a kao što su podatkovne strukture, strukture indeksiranja, pristupnim metodama i optimizacijskim tehnikama.

TSQL2 prepoznaje sljedeće vrste relacija, [28]:

- Relacije trenutnog vremena – *snapshot* relacije,
- Relacije valjanog vremena, koje se dijele na:
    - Relacije stanja,
    - Relacije događaja,
- Relacije transakcijskog vremena,
- Bi-temporalne relacije, koje se dijele na:
    - Relacije stanja,
    - Relacije događaja.

Relacije u povijesnim bazama, odnosno relacije valjanog stanja, sastoje se od podataka koji su valjani u određenom periodu u vremenu, te se mogu podijeliti na dva tipa, relacije stanja – koje sadrže vremenski podatak unutar kojih je navedeni zapis valjan – najčešće period, te relacije događaja – koje sadrže jedan vremenski podatak koji označava trenutak izvršavanja događaja – najčešće trenutak u vremenu.

U tablici 6 prikazana je relacija valjanog stanja, iz koje se iščitavaju zapisi o trajanju stipendija za određenog studenta, te je moguće vidjeti kako postoje dva zapisa za studenta sa ID-jem 1; jedna istekla stipendija u prošlosti, te jedna koja će tek vrijediti u budućnosti, dok za studenta sa ID-jem 7 postoji važeća stipendija, u trenutku pisanja rada.

Tablica 7 predstavlja relaciju događaja, odnosno skup zapisa koji predstavljaju trenutak u kojemu je isplaćena stipendija određenom studentu.

Kada bi se u relacije opisane tablicom 6 i tablicom 7 dodao podatak o trenutku u kojem je nastao zapis ili kada je isti ažuriran – transakcijsko vrijeme – relacija bi predstavljala bi-temporalnu relaciju.

Tablica 6: Relacija stanja na primjeru studentskih stipendija

StudentID	Iznos	Od	Do
1	2000	1.10.2020.	1.10.2021.
7	3200	1.10.2022.	1.10.2023.
1	1500	1.10.2023.	1.10.2024.

Tablica 7: Relacija događaja na primjeru isplaćivanja studentskih stipendija

StudentID	Iznos	Datum
1	2000	10.10.2020. 14:02
1	2000	10.11.2020. 15:00
7	3200	10.10.2022. 12:00

3.2. Prostorne baze podataka

Cilj prostornih baza podataka je proširiti opseg podatkovnih modela SUBP-a te jezika za kreiranje upita kako bi obuhvatili prikaz i dohvat geometrijskih podataka. Prostorne baze podataka nastale su kao podrška geografskim informacijskim sustavima (GIS), te danas skoro svi komercijalni sustavi baza podataka, kao što su Oracle, Microsoft SQL Server, POSTGRES i dr., implementiraju određenu razinu funkcionalnosti za rad sa prostornim podacima, [28].

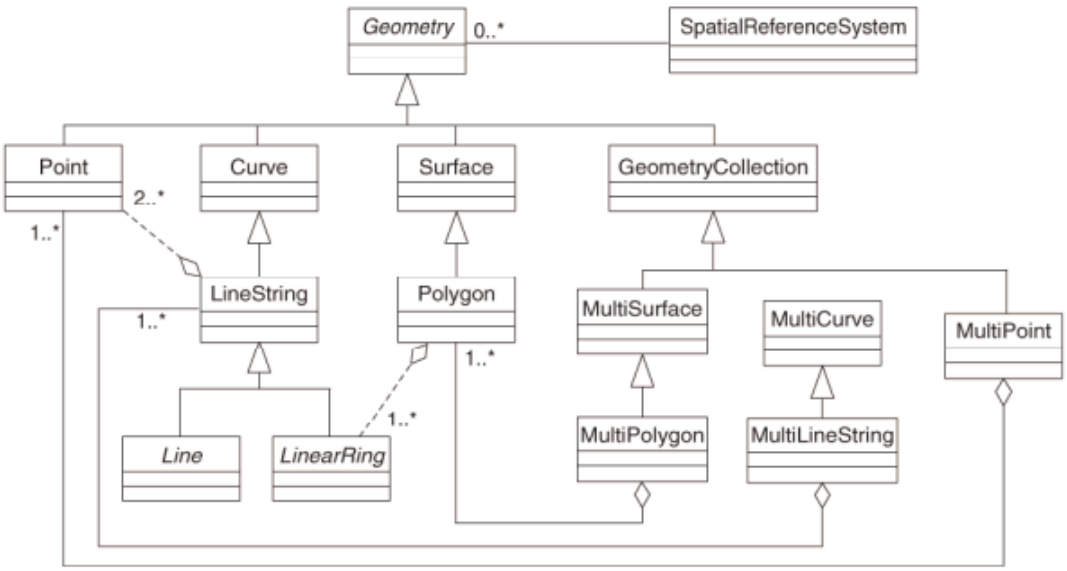
Open Geospatial Consortium – OGC – utemeljen je 1994. godine kako bi univerzalno definirao pravila i prakse za rad sa geografskim informacijama. Glavni cilj je kreirati standarde koji omogućuju razvojnim programerima kreiranje informacijskih sustava za razmjenjivanje geografskih i prostornih podataka među drugim informacijskim sustavima, [31].

Prostorna baza podataka je obična komercijalna baza podataka sa dodatnim mogućnostima i funkcijama koje podržavaju prostorne podatke, kao što su, [32]:

- Prostorni tipovi podataka – pohrana prostornih podataka definiranih kao u *Open Geospatial Consortium*, ili kao BLOB – *binary large object,*
- Prostorni pokazatelji (indeksi) – mehanizmi za ubrzavanje pristupa prostornim podacima korištenjem R-stabla, quad-stabla i B-stabla,
- Prostorni operatori – funkcije za obradu prostornih podataka koje je moguće pozivati korištenjem SQL jezik za dohvat željenih rezultata,
- Prostorne aplikacijske rutine – niz softverskih komponenti za specifične funkcije kao što su učitavanje prostornih podataka, upravljanje performansama, kreiranje sigurnosnih kopija, replikaciju, upravljanje verzijama, kontrolu transakcija i dr.

Prostorni tipovi podataka, odnosno geometrije, čiji međusobni odnosi su prikazani slikom 20, definirani su OGC-om, [33], kako slijedi:

- Točka – engl. „*Point*“,
- Linija – koju je moguće definirati tipovima: engl. „ *LineString*“, „*Line*“, „*LinearRing*“,
- Površina – engl. „*Surface*“,
- Mnogokut – engl. „*Polygon*“,
- Skup geometrijskih oblike – engl. „*GeometryCollection*“,
- Skup površina – engl. „*MultiSurface*“,
- Skup mnogokuta – engl. „*MultiPolygon*“,
- Skup krivulja – engl. „*MultiCurve*“,
- Skup linija – engl. „*MultiLineString*“,
- Skup točaka – engl. „*MultiPoint*“.



Slika 20: Geometrije definirane OGC-om, [33]

Za potrebe pohrane i prijenosa navedenih geometrijskih tipova, OGC definira tekstualni (WKT – engl. „*Well-known Text*“) i binarni (WBT – engl. „*Well-known Binary*“) tip prikaza geometrijskih podataka. Primjeri WKT prikaza predstavljeni su u tablici 8, dok WBT predstavlja geometrijski lik kao niz bajtova.

Tablica 8: Primjeri WKT tekstualnog prikaza geometrijskih podataka

Prostorni tip (geometrija)	**WKT prikaz**	**Opis**
Točka (*Point*)	'POINT (10 10)'	Točka s x i y koordinatama (10,10)
Linija (*LineString*)	'LINESTRING (10 10, 20 20)'	Linija koja spaja točke (10,10) i (20,20)
Mnogokut (*Polygon*)	'POLYGON ((10 10, 10 20, 20 20, 20 15, 10 10))'	Mnogokut sačinjen od točaka (10, 10), (10, 20), (20, 20), (20, 15), (10, 10) posljednja točka mnogokuta mora biti jednaka početnoj.
Skup točaka (*Multipoint*)	'MULTIPOINT (10 10, 20 20)'	Skup točaka (10, 10), (20, 20).
Skup linija (*MultiLineString*)	'MULTILINESTRING ((10 10, 20 20), (15 15, 30 15))'	Skup dvije linije sastavljen od linije koja povezuje točke (10, 10), (20, 20) i linije koja povezuje (15,15), (30, 15).
Skup mnogokuta (*MultiPolygon*)	'MULTIPOLYGON (((10 10, 10 20, 20 20, 20 15, 10 10)), ((60 60, 70 70, 80 60, 60 60)))'	Skup dva mnogokuta sastavljenih od 5 točaka.
Skup geometrija (*GeomCollection*)	'GEOMETRYCOLLECTION (POINT (10 10), POINT (30 30), LINESTRING (15 15, 20 20))'	Skup geometrija sastavljen od dvije točke, i jedne linije koja povezuje druge dvije točke.

Osim samih tipova podataka, definirane su i metode uporabljive nad geometrijskim objektima, kao što su npr. *INTERSECTS* – koja kao rezultat daje 1 (istinu) ukoliko se geometrije sijeku, *CONTAINS* koja kao rezultat daje 1 (istinu) ukoliko se geometrija u cijelosti nalazi unutar druge geometrije, te mnoge druge, [33].

Za kreiranje prostornih indeksa koriste se podatkovne strukture kao što su quad-stabla, R-stabla i k-d-stabla.

3.3. Prostorno-vremenske baze podataka

Prostorno-vremenske baze podataka smatraju se bazama podatka koje implementiraju ranije navedene prostorne i vremenske koncepte kako bi pohranili prostorne, vremenske ili prostorno-vremenske podatke. Mnoge inačice modela prostorno-vremenskih podataka, razvijaju se kao nadogradnja vremenske komponente kako bi podržavala prostornu, ili kao nadogradnja prostorne komponente kako bi uključivala vremensku, stoga i baze podataka prate isto. Većina komercijalnih (relacijskih) baza podataka može se prilagoditi za pohranu i kreiranje upita nad prostorno-vremenskim podacima, ali mali broj takvih sustava je razvijen kao primarno prostorno-vremenski, [34]. Dakle, prostorno-vremenski sustavi upravljanja bazama podataka mogu se promatrati na sljedeći način:

- SUBP koji podržavaju pohranu, obradu i dohvat prostornih i vremenskih atributa podataka – statični podaci,
- SUBP koji podržavaju pohranu, obradu i dohvat podataka koji kontinuirano mijenjaju svoje prostorne i/ili vremenske atribute – dinamični podaci.

Tip prostorno-vremenskih baza podataka koji se bavi pokretnim objektima u vremenu naziva se i baza pokretnih objekata (MOD – „*Moving Objects Database*“).

3.4. Sustavi upravljanja prostorno-vremenskim bazama podataka

3.4.1 SECONDO i BBoxDB

SECONDO je proširi sustav upravljanja bazom podataka, otvorenog kôda, namijenjen ne-standardnim primjenama. Razvijen je na FernUniversitat-u u Hagen-u, te je koncipiran kao platforma za istraživanje različitih podatkovnih modela, koje najčešće nije moguće pronaći u komercijalnim implementacijama. Smatra se jednom od prvih implementacija prostorno-vremenskih SUBP. Proširivost sustava ostvarena je kroz mogućnost implementacije jednog od dodatnih algebarskih modula, [35]:

- Standardna algebra – operacije nad brojčanim, Bool-eovim i tekstualnim tipovima podataka,
- Relacijska algebra – operacije nad tipovima podataka koji prate definicije relacijskog modela,
- Prostorna algebra – podrška za prostorne tipove podataka i operacije nad istima,

- Vremenska algebra – podrška za pokretne objekte u bazi podataka.

SECONDO za upite i izvršavanje operacija osim SQL jezika koristi i jezik izvršavanja.

BBoxDB je distribuirana i skalabilna baza podataka otvorenog kôda za pohranu velike količine višedimenzionalnih podataka. Temeljena je na tradicionalnom principu pohrane ključa i vrijednosti (engl. „*key-value*“) proširenom sa okvirom (engl. „*bounding box*“) prostornih podataka zbog što učinkovitije pohrane i dohvata podataka. Podržava rad s podacima u stvarnom vremenu, te kreiranje kontinuiranih upita nad mijenjajućim prostornim podacima.

3.4.2 Mireo Space-Time

Space-Time softverska platforma tvrtke Mireo d.d. sadrži sve potrebne komponente i funkcionalnosti za pohranu i analizu povijesnih podataka i podataka u stvarnom vremenu o kretanju vozila, [36].

SpaceTime NewSQL baza podataka je distribuirana, skalabilna te razvijena za veliku brzinu unosa prostorno-vremenskih podataka, uz simultane velike analitičke dohvate. Prati ACID svojstva, te koristi vlastito rješenje za kreiranje upita i kreiranje indeksa, što čini ovu bazu visoko dostupnom, otpornom na greške, te horizontalno skalabilnom. Pozadinski kôd baze podataka pisan je u C++-u, što znatno povećava iskoristivost procesorske moći, te omogućuje obradu i do 50.000.000 zapisa po sekundi. Prednost ove baze podataka je raznovrsnost hardverskih platforma na kojima ju je moguće pokrenuti – od običnih osobnih računala, do poslužitelja i Cloud rješenja, [37].

3.4.3 PostgreSQL, PostGIS i MobilityDB

Iako načelno nije prostorno-vremenska baza podataka, PostgreSQL je „najnaprednija relacijska baza otvorenog kôda na svijetu“, [38], te kao takva sadrži mnoga proširenja – uključujući PostGIS i MobilityDB.

PostGIS omogućuje pohranu prostornih podataka, te sadrži prostorne tipove podataka i funkcije definirane OGC-om, te funkcionalnosti za obradu i indeksiranje takvih podataka. Indeksiranje se vrši kroz tri vrste indeksiranja, [39]:

- GiST – engl. „*Generalized Search Tree*“ – generičko indeksiranje višedimenzionalnih podataka, za kreiranje strukture pokazivača koristi se R-stablo, najčešće korišteno te u većinu slučajeva pruža vrlo dobre performanse pretraživanja,

- BRIN – engl. „*Block Range Index*“ – metoda indeksiranja s gubicima – potrebna je dodatna provjera kako bi se potvrdio rezultat pretrage, rezultat joj je znatno manja veličina pokazivača, kraće vrijeme kreiranja indeksa, te dovoljno dobre performanse čitanja,
- SP-GiST – engl. „*Space-Partitioned Generalized Search Tree*“ – tip indeksa koji podržava strukture kao što su quad-stabla, k-d stabla, radix stabla te razdijeljena stabla (engl. „*partitioned trees*“).

MobilityDB je dodatak za PostgreSQL i PostGIS koji omogućuje pohranu pokretnih objekata, kao što su GPS tragovi. Sadrži podršku za vremenske i prostorno-vremenske objekte, te navodi sljedeće karakteristike, [40]:

- Kompaktna pohrana prostornih podataka i putanja,
- Sadržano mnoštvo analitičkih funkcionalnosti,
- Podržana velika količina podataka uz zadržavanje performansi,
- Baziran na SQL sučelju,
- Kompatibilan sa PostgreSQL ekosustavom,
- Suglasan sa OGC standardima za prostorne podatke i pokretne podatke,
- Prihvaćen od strane zajednice za prostorne sustave otvorenog koda (OSGeo – engl. „Open Source Geospatial Foundation“),
- Sadrži razvijene dodatke za rad kroz Python i QGIS[6].

3.4.4 GeoMesa

Iako nije baza podataka, GeoMesa sadrži set alata otvorenog kôda za kreiranje upita i analizu prostorno-vremenskih podataka, povijesnih i u stvarnom vremenu, sa velikog spektra distribuiranih NoSQL i NewSQL baza podataka, kao što su: Accumulo, HBase, Google Bigtable i Cassandra, [41]. Neke od značajki ovog rješenja su, [42]:

- Pohrana velike količine prostornih podataka – više od desetak milijardi,
- Dohvat preko 10 milijuna točaka u sekundi,
- Unos podataka brzine preko 10.000 zapisa u sekundi, po čvoru,
- Mogućnost jednostavnog horizontalnog skaliranja,
- Potpuna podrška OGC standarda za prostorne podatke i dr.

[6] QGIS je besplatan geografski informacijski sustav otvorenog kôda.

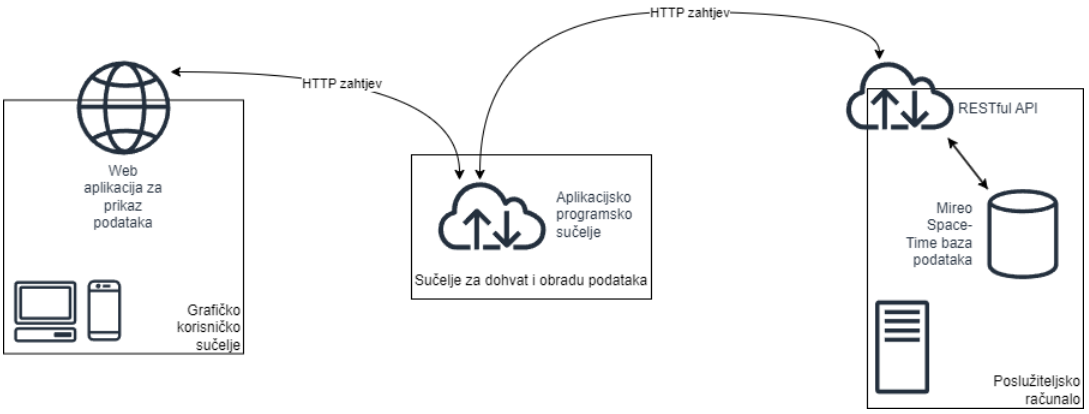
4. Sučelje za dohvat, obradu, interakciju i prikaz podataka na karti

Za potrebe prikaza podataka o kretanju vozila prometnom mrežom, izrađena je web aplikacija, kojoj je svrha dohvat i prikaz podataka iz baze pokretnih podataka. Podaci o kretanju vozila prometnom mrežom spremljeni su u bazu pokretnih objekata Mireo Space-Time, na računalu lociranom u prostoru Fakulteta prometnih znanosti.

Za potrebe istraživanja na računalu Fakulteta, ugrađen je brzi NVME 1 TB SSD disk spojen preko PCI Express 3.0, kako bi se osigurao brzi dohvat i pohrana podataka, te dodatno jedan tvrdi disk od 1 TB na kojem je instaliran operacijski sustav Linux verzije Debian 9, koji omogućava rad s Mireo Space-Time bazom podataka. Tvrtka Mireo d.d. je na navedenom SSD disku postavila svoju prostornu-vremensku bazu podataka, a u navedenom operacijskom sustavu instalirala SUBP i poslužitelj za rad s bazom podataka koji omogućava pristup bazi podataka putem RESTful API-ja. Fakultet prometnih znanosti nema direktan uvid u sam dizajn i implementaciju SUBP-a, te jedino upravlja mogućnošću paljenja, odnosno gašenja servera. Pristup te korištenje navedene baze podataka omogućeno je isključivo unutar mreže Fakulteta prometnih znanosti.

Razmatranjem arhitekture i potrebnih programskih komponenti za razvoj sučelja za dohvat, obradu, interakciju i prikaz podataka na karti, zaključeno je da će se isto sastojati od tri zasebne funkcionalne cjeline, prikazane na slici 21:

1. Baza podataka – sadrži podatke o kretanju vozila te mogućnosti za selektivni dohvat istih,
2. Aplikacijsko programsko sučelje – API (engl. „*Application Programming Interface*“) – zaduženog za dohvat podataka iz baze podataka, te njihovu obradu i transformaciju u format pogodan za prijenos i prikaz u sučelju,
3. Grafičko korisničko sučelje – GUI (engl. „*Graphical User Interface*“) – mrežna stranica koja sadrži funkcionalnosti za filtriranje podataka i prikaz istih na karti.



Slika 21: Arhitektura sučelja za dohvat, obradu, interakciju i prikaz podataka na karti

Za kreiranje komponente aplikacijskog programskog sučelja – sučelja posrednika između same baze podataka i grafičkog sučelja, razmatrane su sljedeće tehnologije:

- .NET Core – je besplatna razvojna platforma otvorenog tipa s podrškom za Windows, Linux i Mac poslužitelje, temeljena na C# programskom jeziku,
- Node.js – razvojna platforma visokih performansi temeljena na JavaScript-u namijenjena za kreiranje alata i aplikacija na poslužiteljima, uz Express mrežni razvojni okvir omogućuje brzi razvoj mrežnih aplikacija sa podrškom za HTTP zahtjeve i rute,
- Ruby on Rails – sadrži skup funkcionalnosti kao što su HTML predlošci, rad s bazama podataka, slanje i primanje e-pošte, i mnoge druge funkcionalne i sigurnosne mogućnosti, koristi se Ruby programski jezik, te Rails mrežno-aplikacijski razvojni okvir koji prati MVC[7] razvojni obrazac,
- Django – razvojni mrežni okvir visoke razine temeljen na Python programskom jeziku, namijenjen za brz i skalabilan razvoj mrežnih aplikacija,
- Flask – popularan razvojni okvir za mrežne aplikacije temeljen na Python-u, klasificira se kao mikro-okvir jer ne zahtijeva korištenje određenih alata i biblioteka,

[7] MVC – engl. „*Model-View-Controller*“ – je razvojni obrazac u projektiranju softvera koji se koristi za implementaciju korisničkih sučelja, podataka i upravljačke logike – sa naglaskom na odjeljivanje briga koje donosi svaka od navedenih komponenti – čime se olakšava održavanje, te smanjuje kompleksnost ukupnog sustava. [55]

- Laravel – besplatna razvojna platforma otvorenog tipa za kreiranje mrežnih aplikacija koristeći PHP programski jezik, sadrži mnoštvo alata i paketa potrebnih za kreiranje modernih mrežnih aplikacija.

S obzirom na usmjerenost grafičkog sučelja na prikaz dohvaćenih podataka o kretanju vozila na karti, nisu razmatrane napredne i kompleksnije tehnologije za kreiranje istog, već je rješenje kreirano koristeći HTML (engl. „*HyperText Markup Language*“) i JavaScript (JS), te biblioteku za prikaz i interakciju sa kartom.

Postoje razne JS biblioteke za prikaz podataka na karti i interakciju s istom, od kojih su najpopularnije sljedeće:

- Leaflet.js – besplatna vodeća JS biblioteka otvorenog tipa za kreiranje interaktivnih karti zbog svoje jednostavnosti i uporabljivosti, mnoštva dostupnih dodataka, opširne i jasno napisane dokumentacije te podrške za pregled na mobilnim uređajima,
- OpenLayers – besplatna JS biblioteka otvorenog tipa za kreiranje mrežnih stranica sa dinamičkom kartom, opširnija od prethodno navedene, ali sa sličnim performansama i značajkama,
- Mapbox GL JS – JS biblioteka za kreiranje interaktivnih mrežnih stranica sa vektorskim kartama, s podrškom za 3D karte, različite dizajne, uzorke terena i projekcije te mnoštvo dodatnih funkcionalnosti, naplaćuje se ovisno o korištenju istih.

Važno je napomenuti da ovo rješenje nije namijenjeno za korištenje u produkcijske svrhe, već je izrađeno isključivo kao dokaz koncepta za potrebe ovog rada, stoga je moguće da prilikom izrade istog nisu primijenjene određene uobičajene programske prakse.

4.1. Baza podataka

Komponenta baze podataka je SUBP Mireo Space-Time, koja sadrži podatke o ukupno 64,757,704 putovanja vozila u periodu od 1.1.2014. do 1.7.2020., te ukupne pređene udaljenosti preko 8,492,840,000 kilometara.

Osnovna konfiguracija, sadrži tri tablice s prostorno-vremenskim podacima:

- Segmenti – *st.segments* – tablica sa najvećim, i najvrjednijim skupom podataka – sadrži zapise o tragovima sa GNSS-a (engl. „*Global Navigation Satellite System*“)

prijemnika koji su uz pomoć algoritama, topoloških analiza i rekreacije putanje povezani na cestovne segmente,

- Putovanja – *st.trips* – sadrži zapise o putovanjima vozila, čija se početna točka definira kao pozicija vozila u kojoj je upaljen pogon, a krajnja točka je pozicija u kojoj je ugašen, sadrži manje podataka od prijašnje tablice, stoga je pogodnija za generalne upite o prijeđenoj udaljenosti ili duljini vožnje, te za opći prikaz kretanja vozila u danu, gdje zbog preglednosti i brzine dohvata nije optimalno koristiti segmente,
- Intervali – *st.intervals* – sadrži podatke o stanju opreme ugrađene u vozilo, za potrebe ovog rada nisu korišteni podaci iz iste.

Struktura tablica *st.segments* i *st.trips* prikazani su u tablici 9 i tablici 10, [43]:

Tablica 9: Struktura tablice *st.segments*

Naziv atributa	Tip podatka	Opis
vid	int32	Unikatni identifikator vozila
t	rint32	Polje sastavljeno od dva cjelobrojna elementa, koja predstavljaju Unix epohe[8] početka i kraja segmenta
x	rint32	Polje od dva cjelobrojna elementa, koja predstavljaju geografsku dužinu početka i kraja segmenta, u decimalnim stupnjevima, pohranjenu u Mireovom posebnom cjelobrojnom formatu
y	rint32	Polje od dva cjelobrojna elementa, koja predstavljaju geografsku širinu početka i kraja segmenta, u decimalnim stupnjevima, pohranjenu u Mireovom posebnom cjelobrojnom formatu
v	ruint16	Polje od dva cjelobrojna elementa koja predstavljaju brzine vozila na početku i kraju segmenta, [km/h]
sl	uint8	Cjelobrojna vrijednost koja predstavlja graničenje brzine na segmentu, [km/h]

[8] Unix epoha, još poznata kao Unix vrijeme, POSIX vrijeme, ili Unix vremenska oznaka, predstavlja broj sekundi od 1.1.1970. godine, bez ubrojenih prestupnih sekundi pomoću kojih se kompenziraju varijacije Zemljine rotacije. [53]

flags	uint8	Cjelobrojna vrijednost koja predstavlja kategoriju ceste kojoj segment pripada, definirana po Mireovoj internoj specifikaciji npr.: • autoceste – 1010, • avenije – 1020, • manje prometnice – 1080, i dr.
style	uint16	Kategorija cestovnog pravca, proširiva posebnom Mireo tablicom (ista nije dostupna u korištenoj verziji) – npr. tuneli, vijadukti i dr.

Tablica 10: Struktura tablice *st.trips*

Naziv atributa	Tip podatka	Opis
vid	int32	Unikatni identifikator vozila
t	rint32	Polje sastavljeno od dva cjelobrojna elementa, koja predstavljaju Unix epohe[8] početka i kraja puta
x0	rint32	Cjelobrojna vrijednost koja predstavlja geografsku dužinu početka puta, u decimalnim stupnjevima, pohranjenu u Mireovom posebnom cjelobrojnom formatu
y0	rint32	Cjelobrojna vrijednost koja predstavlja geografsku širinu početka puta, u decimalnim stupnjevima, pohranjenu u Mireovom posebnom cjelobrojnom formatu
x1	ruint16	Cjelobrojna vrijednost koja predstavlja geografsku dužinu kraja puta, u decimalnim stupnjevima, pohranjenu u Mireovom posebnom cjelobrojnom formatu
yl	uint8	Cjelobrojna vrijednost koja predstavlja geografsku širinu kraja puta, u decimalnim stupnjevima, pohranjenu u Mireovom posebnom cjelobrojnom formatu
len	uint8	Duljina puta [m]

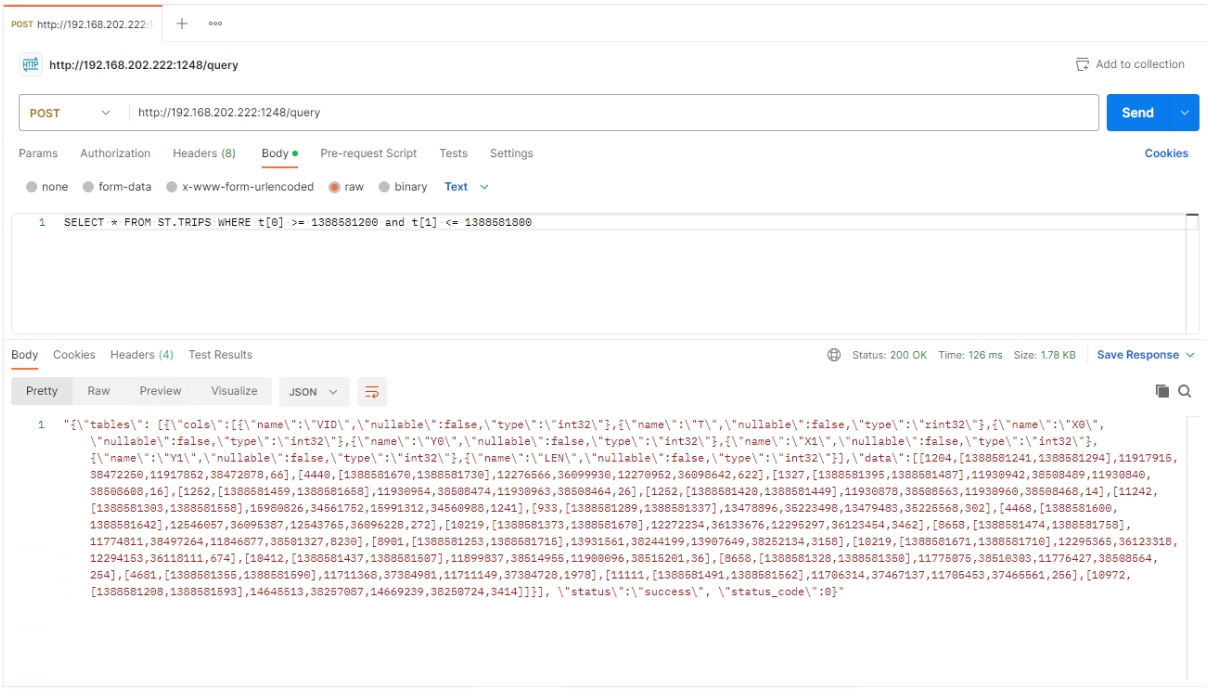
Podacima iz navedenog SUBP pristupa se preko RESTful (engl. REST – „*Representational State Transfer*“) API-ja. REST arhitektura podrazumijeva sljedeća svojstva:

- Arhitektura poslužitelj-klijent koja se sastoji od klijenata, poslužitelja, resursa i zahtjeva kojima se upravlja HTTP (engl. „*Hypertext Transfer Protocol*“) protokolom,
- Svaka komunikacija između klijenta i poslužitelja je odvojena i nepovezana, nema pohrane informacija između zasebnih zahtjeva (engl. „*stateless communication*“),
- Uniformno sučelje između komponenti koje omogućuje prijenos podataka u izvornom obliku,
- Traženi resursi logički su odvojeni od njihovog prikaza koji se šalje klijentu, koji na temelju istog ima dovoljno informacija za manipulaciju podacima tj. resursima,
- Poslužitelj uz same podatke mora poslati opisne poruke koje služe kao uputa korisniku na koji način je potrebno obraditi primljene podatke, [44].

REST API sadrži četiri HTTP metode za dohvat i manipulaciju podacima: *GET* – za dohvat informacija o resursu, *POST* – za kreiranje resursa, *PUT* – za ažuriranje resursa te *DELETE* – za brisanje resursa, [45].

Za dohvat podataka iz Mireo baze podataka koristi se POST zahtjev, u čije se tijelo (engl. „*request body*“) postavlja tekst SQL upita. S obzirom da navedeni API preko kojeg se pristupa podacima izvršava cijeli upit poslan kroz zahtjev, potrebno je kod implementacije aplikacijskog sloja između baze podataka i GUI sanirati SQL upit kako bi se onemogućilo maliciozno korištenje istog – tzv. *SQL Injection* napad.

POST zahtjev, sa SQL upitom za dohvat svih puteva koji su počeli i završili u razdoblju od 13:00 do 13:10 dana 1.1.2014., iz tablice st.trips, opisane tablicom 10, te odgovor na isti, prikazan je na slici 22.



Slika 22: POST zahtjev za dohvat puteva napravljen kroz *Postman*[9] aplikaciju te zaprimljeni odgovor na isti

U odgovoru na POST zahtjev, prikazanom na slici 22, podaci se dostavljaju kao JSON[10] objekt, koji sadrži podatke o stupcima tablice, te same podatke. Kako bi se isti obradili u samoj aplikaciji, potrebno ih je pretvoriti u relevantne modele ovisno o vrsti podataka koji se dohvaćaju.

JSON objekt iz odgovora, prikazan na slici 23, sa podacima sadrži niz podatka *tables,* u kojemu se nalaze dva podatkovna niza, *cols* – koji sadrži objekte koji opisuju atribute stupaca – ime, tip i ništavnost podatka kojeg sadržava navedeni stupac, te *data* – niz podataka koji sadrži same podatke, na način da jedan objekt predstavlja jedan redak dohvaćenih podataka, a podaci unutar objekta su poredani istim redoslijedom kao i stupci u ranije objašnjenom nizu *cols*.

[9] Postman je API platforma za kreiranje i korištenje aplikacijskih programskih sučelja, olakšava korištenje i slanje HTTP Rest zahtjeva.

[10] JSON – engl. „*JavaScript Object Notation*“ – tekstualni format za razmjenu podataka, jednostavan za čitanje ljudima i strojevima, baziran na JavaScript programskom jeziku, ali potpuno neovisan o implementacijskom programskom jeziku . Sastoji se od kolekcije parova imena i vrijednosti – engl. „*name-value pairs*“.

```
{
    "tables": [
        {
            "cols": [
                {
                    "name": "VID",
                    "nullable": false,
                    "type": "int32"
                },
                {
                    "name": "T",
                    "nullable": false,
                    "type": "rint32"
                },
                {
                    "name": "X0",
                    "nullable": false,
                    "type": "int32"
                },
                {
                    "name": "Y0",
                    "nullable": false,
                    "type": "int32"
                },
                {
                    "name": "X1",
                    "nullable": false,
                    "type": "int32"
                },
                {
                    "name": "Y1",
                    "nullable": false,
                    "type": "int32"
                },
                {
                    "name": "LEN",
                    "nullable": false,
                    "type": "int32"
                }
            ],
            "data": [
                [
                    1204,
                    [
                        1388581241,
                        1388581294
                    ],
                    11917915,
                    38472250,
                    11917852,
                    38472878,
                    66
                ],
                [
                    4440,
                    [
                        1388581670,
                        1388581730
                    ],
                    12276566,
                    36099930,
                    12270952,
                    36098642,
                    622
                ],
                ...
            ]
        }
    ],
    "status": "success",
    "status_code": 0
}
```

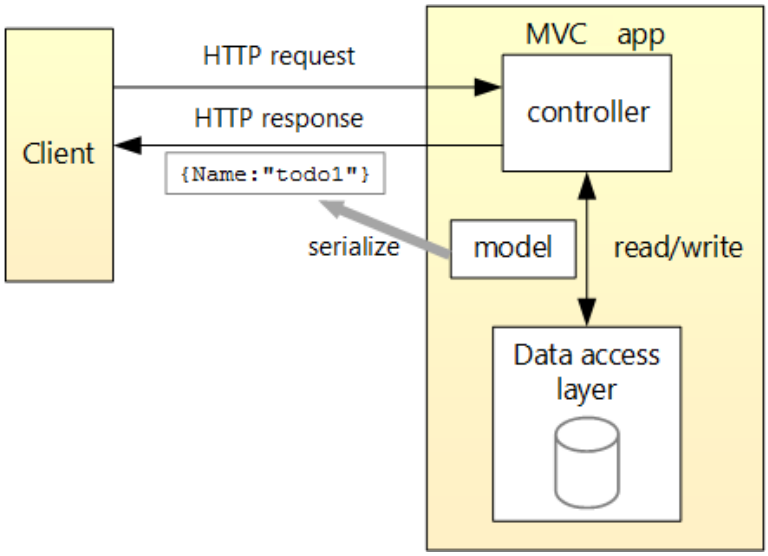

Slika 23: Odgovor na zahtjev prikazan kao JSON objekt

4.2. Aplikacijsko programsko sučelje

Aplikacijsko programsko sučelje služi kao sloj između baze podataka i grafičkog korisničkog sučelja, te sadrži definicije modela koji se dohvaćaju iz baze podataka, te metode za dohvat i obradu istih za prikaz.

Zbog jednostavnosti, te prijašnjeg iskustva s istom odabrana je .NET Core tehnologija te *Web API* tip projekta i jezik C#. Arhitektura općenitog Web API projekta prikazana je na slici 24, te se sastoji od:

- upravljačkog sloja (engl. „*controller*“) – zaduženog za zaprimanje HTTP zahtjeva od klijenta, te kreiranje i slanje odgovora,
- modela – koji sadrži definicije podatkovnih modela, metode za pretvorbu podataka dohvaćenih iz baze podataka u objekte za korištenje u aplikacijskom sloju, te pretvorbu takvih podataka u oblik pogodan za prijenos HTTP odgovorom do klijenta,
- sloja za dohvat podataka – metode za pristup, dohvat i obradu podataka iz baze podataka.



Slika 24: Arhitektura .NET Core Web API projekta, [46]

4.2.1 Postavljanje razvojnog okruženja

Zbog lokacije poslužitelja baze podataka na mrežnoj infrastrukturi Fakulteta prometnih znanosti, te potrebe da aplikacijsko programsko sučelje ima pristup podacima na istom, razvojno okruženje korišteno za kreiranje ovog rješenja se također mora nalaziti unutar infrastrukture fakulteta.

Stoga, za potrebe izrade rada omogućen je udaljeni pristup na računalo na kojem će se vršiti razvoj. Udaljeni pristup omogućen je korištenjem protokola za pristup udaljenim radnim površinama, razvijenog od strane Microsoft-a (engl. RDP – „*Remote Desktop Protocol*“).

Projekt .NET Core Web API moguće je postaviti kroz *Visual Studio* IDE (engl. IDE – „*Integrated development environment*“) grafičkim putem, te u *Visual Studio Code*-u kroz terminal. Za kreiranje ovog rješenja korišten je *Visual Studio* 2022 IDE, instaliran na ranije navedenom računalu.

Koraci za kreiranje ovog projekta kroz *Visual Studio* IDE su kako slijedi:

1. Na izborničkoj traci, odabrati opciju Datoteka, te novi projekt,
2. Odabrati predložak ASP.NET Core Web API te unijeti ime projekta,
3. Odabrati dodatne opcije, kao što su verzija razvojnog okvira (za potrebe ovog rada korištena je verzija .NET 7.0), te opcija koristi upravljače (engl. – „*Use controllers*“), [46].

4.2.2 Dohvat i obrada podataka

Upravljački sloj sastoji se od više upravljača, svaki za svoj model podataka, te svaki sadrži ranije navedene REST HTTP metode za pristup istima – koje se još nazivaju i krajnjim točkama (engl. „*endpoint*“).

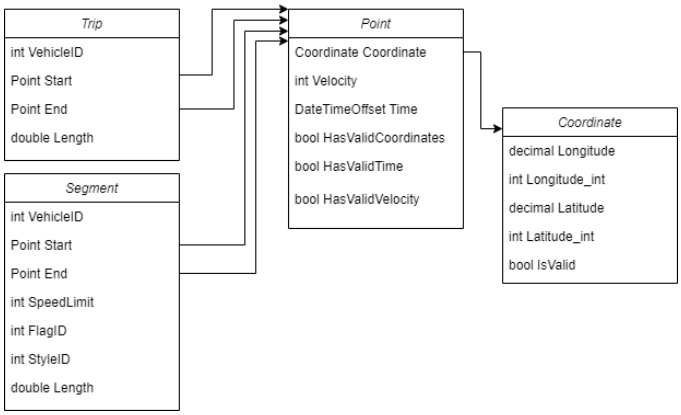
Metode upravljača se direktno izlažu klijentu, te se kao takve pozivaju iz prezentacijskog sloja, u ovom slučaju iz grafičkog korisničkog sučelja. Iako je generalna praksa da svaki model sadrži zasebni kontroler s metodama za dohvat, kreiranje, ažuriranje i brisanje istog (*GET*, *PUT*, *POST*, *DELETE*), za potrebe ovog rada kreirane su samo metode za dohvat podataka.

Korištena su četiri upravljača:

- *Segments Controller* – dohvat segmenata,
- *Trips Controller* – dohvat puteva,
- *Graph Controller* – dohvat podataka za generiranje grafikona i
- *Misc Controller* – pomoćne funkcije.

Osnovni korišteni modeli prikazani su na slici 25 te služe za privremenu pohranu podataka iz baze podataka, kako bi se omogućila olakšana programska obrada istih. *Trip* i *Segment* klase zamišljene su kao preslika shema odgovarajućih tablica (*st.trips* i *st.segments*), uz razliku što koriste pomoćne klase *Point* i *Coordinate*.

Klasa *Point* namijenjena je za pohranu podataka o početnoj i krajnjoj točki puta ili segmenta, te sadrži koordinate, brzinu i vrijeme. *Coordinate* klasa sadrži atribute geografske širine i duljine u dva formata – decimalnom i cijelom, zbog načina na koji Mireo pohranjuje koordinate u bazi – kao cijeli broj, dok prikaz u GUI-ju očekuje klasični – decimalni format. *Coordinate* klasa u svojem konstruktoru sadrži pozive metoda za pretvorbu iz decimalnog u cijeli, i obrnuto, ovisno o tome koji podatak je poznat prilikom kreiranja objekta. Funkcije za pretvorbu koordinati iz decimalnog u cijeli, i obrnuto prikazane su na slici 26.



Slika 25 : Osnovni korišteni modeli podataka

```
public static int ConvertLongitude(decimal longitude)
{
  try
  {
    return Convert.ToInt32(Math.Round((decimal)longitude / (decimal)180.0 * Convert.ToInt32("08000000", 16), 0));
  }
  catch (Exception)
  {
    return 0;
  }
}

public static decimal ConvertLongitude(int longitude)
{
  try
  {
    return (decimal)longitude / Convert.ToInt32("08000000", 16) * (decimal)180.0;
  }
  catch (Exception)
  {
    return 0;
  }
}

public static int ConvertLatitude(decimal latitude)
{
  try
  {
    decimal p = (latitude * DecimalMath.Pi / (decimal)180.0 + DecimalMath.Pi / 2) / (decimal)2.0;
    decimal value = Convert.ToInt32("08000000", 16) / DecimalMath.Pi * DecimalMath.Log(DecimalMath.Tan(p));
    return Convert.ToInt32(Math.Round(value, 0));
  }
  catch (Exception)
  {
    return 0;
  }

}

public static decimal ConvertLatitude(int latitude)
{
  try
  {
    return ((decimal)2 * DecimalMath.ATan(DecimalMath.Exp((decimal)latitude / Convert.ToInt32("08000000", 16) * DecimalMath.Pi)) - DecimalMath.Pi / 2) *
(decimal)180.0 / DecimalMath.Pi;
  }
  catch (Exception)
  {
    return 0;
  }
}
```

Slika 26: Pretvorba koordinata unutar klase *Coordinate*

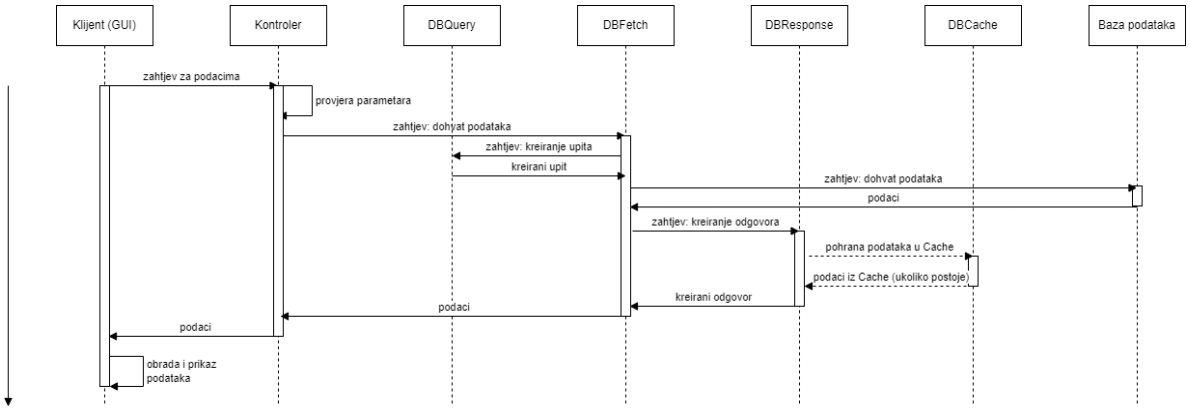
Osim modela i kontrolera, koriste se i pet pomoćnih klasa za dohvat, obradu i pohranu podataka, te za kreiranje odgovora:

- DBQuery – sadrži metode za dinamično kreiranje SQL upita,
- DBFetch – sadrži metode za dohvat podataka iz baze podataka,
- DBResponse – sadrži metode za obradu odgovora iz baze podataka, te za kreiranje odgovora koji se šalje klijentu,
- DBCache – sadrži metode za pohranu često korištenih podataka u memoriji,
- DBOperations – sadrži metode za spajanje rezultata kompleksnih višestrukih upita.

Generalni hodogram ovog aplikacijskog rješenja, prikazan na slici 27, je:

1. Klijent kreira zahtjev pozivajući određenu REST metodu unutar željenog upravljača,
2. Unutar kontrolera obrađuje se zahtjev i parametri predani uz isti,

3. Poziva se određena metoda unutar DBQuery klase za kreiranje SQL upita za dohvat traženih podataka iz baze,
4. Poziva se određena metoda unutar DBFetch klase koja obrađuje kreirani SQL upit iz prošlog koraka, te poziva REST API POST metodu Mireo baze podataka, u čije tijelo postavlja obrađeni SQL upit,
5. DBResponse obrađuje zaprimljeni odgovor iz baze podataka, te isti pretvara u listu objekata pomoću jednog od ranije navedenih modela,
6. DBResponse prolazi kroz sve objekte liste, te ukoliko je potrebno iste pohranjuje u DBCache, i priprema odgovor koji se šalje klijentu,
7. Upravljač, koji je zaprimio zahtjev u prvom koraku, vraća klijentu odgovor generiran u prijašnjem koraku.



Slika 27: Hodogram dohvata podataka kroz aplikacijsko korisničko sučelje

DBQuery klasa za dinamično kreiranje SQL upita koristi SQLKata NuGet paket. SQLKata podržava SqlServer, MySql, PostgreSql, Oracle, SQLite i Firebird proizvođače SUBP-a, te kreiranje kompleksnih dohvata s ugniježđenim WHERE uvjetima i višestrukim spajanjima. Korištenje ovog rješenja u potpunosti onemogućuje SQL *injection* napade zbog metode vezivanja parametara, [47].

S obzirom da Mireo Space-Time baza podataka koristi svoju, minimalno izmijenjenu verziju SQL jezika, te zbog korištenje višestrukih prostornih funkcija i parametara u upitima, nažalost nije moguće iskoristiti sve prednosti SQLKata paketa, te je u kompleksnim upitima potrebno ručno generirati upite. Ovaj nedostatak mogao bi se riješiti definiranjem prilagođenog prevodioca za Mireo SQL jezik, čime bi se dodatno ubrzalo kreiranje upita, a time i sami dohvat podataka, te programski razvoj. No sama implementacija i definicija Mireo SQL jezika nije ustupljena, tako da je u ovom radu bilo potrebno ručno pisati kompleksnije upite.

Na slici 28 prikazan je primjer programskog kôda za generiranje SQL upita za dohvat puteva unutar mnogokuta. Ova metoda omogućuje korištenje jednog upita za dohvat svih puteva unutar nekog vremena, za određeno vozilo, te kojemu su početna i/ili završna točka unutar definiranog mnogokuta. Metoda prima objekt tipa *Trip*, koji u sebi sadrži *VehicleID* (identifikator vozila) te početno i završno vrijeme, te objekt tipa *Polygon*, koji sadrži listu *Coordinate* objekata. Unutar metode, za *Trip* objekt provjerava se valjanost parametara, te ukoliko su isti uneseni u *Trip* objektu, koriste se kao *WHERE* uvjet u *SELECT* naredbi, ali bitno je napomenuti da iste nije potrebno unijeti, čime se određeni uvjet neće postaviti. Za *Polygon* objekt, prolazi se kroz sve *Coordinate* objekte u listi, te se iz istih generira niz teksta koji predstavlja naredbu za kreiranje mnogokuta[11], koji će se koristiti u SQL upitu. U ovom primjeru, predefinirano je da su oba vremena uključujuća, iako zbog jednostavnosti implementacije ne postoji razlog da se i operatori ne postavljaju dinamički. Koristeći ovaj paket nije potrebno definirati više predefiniranih tekstualnih upita ovisno o valjanosti ili postojanju parametara, te se izbjegava mogućnost SQL *injection*[12] napada, jer se prilikom serijalizacije parametara provjeravaju vrijednosti atributa istih.

[11] Naredba za kreiranje mnogokuta: „polygon(koordinata1, koordinata 2, koordinata 3, …, koordinata N)“

[12] Mogućnost SQL injection napada u ovom primjeru djelomično je smanjen zbog ograničenja na korištenje aplikacije unutar domene Fakulteta prometnih znanosti.

```
public static Query GetTripsQuery(Trip trip)
{
  var query = new Query("st.trips").Select("*").As("Trips");

  if (trip is not null)
  {
    if (trip.VehicleID > 0)
    {
      query = query.Where("vid", trip.VehicleID.ToString());
    }
    if (trip.Start.Time is not null && trip.Start.HasValidTime.HasValue && trip.Start.HasValidTime.Value)
    {
      query = query.Where("t[0]", ">=", trip.Start.Time.Value.ToUnixTimeSeconds().ToString());
    }
    if (trip.End.Time is not null && trip.End.HasValidTime.HasValue && trip.End.HasValidTime.Value)
    {
      query = query.Where("t[1]", "<=", trip.End.Time.Value.ToUnixTimeSeconds().ToString());
    }
  }

  return query;
}

public static Query GetTripsInPolyQuery(Trip trip, Polygon polygon)
{
  var query = new Query().From(GetTripsQuery(trip));
  string polyStr = $"";
  if (polygon.Coordinates.Count >= 3)
  {
    polyStr = $"";

    for (int i = 0; i < polygon.Coordinates.Count; i++)
    {
      if(i==0)
      {
        polyStr += $"{polygon.Coordinates[i].Longitude_Int.ToString()} {polygon.Coordinates[i].Latitude_Int.ToString()}";
      }
      else
      {
        polyStr += $", {polygon.Coordinates[i].Longitude_Int.ToString()} {polygon.Coordinates[i].Latitude_Int.ToString()}";
      }
    }
  }

  return query.WhereRaw($"ST_Intersects(ST_GeomFromText('POLYGON (({polyStr}))'), ST_Line(x0,y0,x1,y1))");
}
```

Slika 28: Generiranje SQL upita za dohvat specifičnih putovanja unutar mnogokuta koristeći SQLKata paket

Kreirani upit se koristi u DBFetch, gdje se dodatno obrađuje zbog specifičnosti SQL jezika korištenog u Mireo Space-Time bazi podataka, te se šalje u tijelu POST zahtjeva na REST API Mireo baze, prikazano slikom 29, nakon čega se odgovor baze predaje DBResponse na obradu i kreiranje odgovora.

```
public static async Task<string> GetTripsInPolygon(Trip trip, Polygon polygon)
{
  var compiler = new MyCompiler();

  var query = DBQuery.GetTripsInPolyQuery(trip, polygon);

  SqlResult sql = compiler.Compile(query);

  using var client = new HttpClient();

  var url = "http://192.168.202.222:1248/query";
  var data = new StringContent(sql.ToString().Replace("{","").Replace("}","").Replace("'","").Replace("POLYGON", "'POLYGON").Replace(")))",")))')"));

  var response = await client.PostAsync(url, data);

  if (!(response is null))
  {
    if (response.IsSuccessStatusCode && !(response.Content is null))
    {
      string resultt = await response.Content.ReadAsStringAsync();

      var result = DBResponse.CreateGeoJSONResponseTrips(resultt);
      if (!(result is null))
      {
        return result;
      }
    }
  }

  return string.Empty;
}
```

Slika 29: Obrada SQL upita i dohvat podataka sa REST API-ja Mireo Space-Time baze

DBResponse ima višestruku, možda i najvažniju ulogu. S obzirom na dolazni format podataka iz baze, prikazan slikom 23, odnosno u ovom kontekstu iz DBFetch, isti nije moguće jednoznačno pretvoriti u modele navedene na slici 25, iz razloga što se upiti mogu razlikovati; neki upiti mogu imati različiti redoslijed stupaca, ili uopće ne sadržavati neke stupce.

Iz tog razloga DBResponse sadrži metode za eksplicitnu pretvorbu dolaznih podataka u objekte, odnosno ranije navedene modele. Pretvorba se vrši na način da se prvo prolazi kroz sve stupce navedene u zaprimljenom *JSON*-u, te se formira *DataTable* objekt sa stupcima koji odgovaraju imenu i tipu stupca navedenih u *cols* dijelu JSON-a. Formiranje stupaca *DataTable* tablice prikazano je slici 30, gdje je vidljivo da se prolazi kroz sve stupce navedene u podatkovnom nizu *cols* dolaznog *JSON* objekta, te ukoliko je tip podatka navedenog stupca *rint* ili *ruint*, čija je definicija navedena u tablici 9 i tablici 10 – podatkovni niz cijelih brojeva – kreiraju se višestruki stupci, npr. ukoliko je stupac X tipa cjelobrojnog podatkovnog niza, te sadrži dva podatka u odgovarajućem polju u *cols* nizu *JSON* objekta, kreirati će se dva stupca cjelobrojnog tipa – X0 i X1.

```
dynamic results = JsonConvert.DeserializeObject<dynamic>(inData);

DataTable result = new DataTable();

for(int columnIndex = 0; columnIndex < results.tables[0].cols.Count; columnIndex++)
{
  var column = results.tables[0].cols[columnIndex];
  string coltype = column.type.ToString();

  if (coltype.Contains("rint") || coltype.Contains("ruint"))
  {
    int rowArraySize = results.tables[0].data[0][columnIndex].Count;
    for (int i = 0; i < rowArraySize; i++)
    {
      DataColumn dtcolumn = new DataColumn();
      dtcolumn.DataType = System.Type.GetType("System.Int32");
      dtcolumn.ColumnName = column.name.ToString().ToUpper() + i;
      result.Columns.Add(dtcolumn);
    }

  }
  else if (coltype.Contains("int"))
  {
    DataColumn dtcolumn = new DataColumn();
    dtcolumn.ColumnName = column.name.ToString().ToUpper();
    dtcolumn.DataType = System.Type.GetType("System.Int32");
    result.Columns.Add(dtcolumn);
  }
  else if (coltype.Contains("float"))
  {
    DataColumn dtcolumn = new DataColumn();
    dtcolumn.ColumnName = column.name.ToString().ToUpper();
    dtcolumn.DataType = System.Type.GetType("System.Double");
    result.Columns.Add(dtcolumn);
  }
  else if (coltype.Contains("char"))
  {
    DataColumn dtcolumn = new DataColumn();
    dtcolumn.ColumnName = column.name.ToString().ToUpper();
    dtcolumn.DataType = System.Type.GetType("System.String");
    result.Columns.Add(dtcolumn);
  }
}
```

Slika 30: Popunjavanje stupaca *DataTable* objekta

Nakon što su uspostavljeni odgovarajući tipovi podataka stupaca, potrebno je popuniti *DataTable* objekt podacima, odnosno kreirati redove. S obzirom da su stupci u tablici te podaci u *data* nizu *JSON* objekta jednako poredani, potrebno je samo paziti na redoslijed stupaca koji su pretvoreni iz jednog stupca koji je tipa podatkovni niz, u N stupaca. Primjer programskog kôda za popunjavanje redaka *DataTable* objekta iz *JSON* objekta prikazan je na slici 31:

```
foreach (var datarow in results.tables[0].data)
{
  DataRow row = result.NewRow();
  int iColumn = 0;
  for (int i = 0; i < results.tables[0].cols.Count; i++)
  {
    if (datarow[i].Type.ToString() == "Array")
    {
      for (int j = 0; j < datarow[i].Count; j++)
      {
        if (result.Columns[iColumn].DataType == typeof(Int32))
        {
          row[iColumn] = Int32.TryParse(datarow[i][j].ToString(), out int datafromrow) ? datafromrow : -1;
        }
        else if (result.Columns[i].DataType == typeof(String))
        {
          row[iColumn] = datarow[i][j].ToString();
        }
        iColumn++;
      }
    }
    else
    {
      if (result.Columns[iColumn].DataType == typeof(Int32))
      {
        row[iColumn] = Int32.TryParse(datarow[i].ToString(), out int datafromrow) ? datafromrow : -1;
      }
      else if (result.Columns[iColumn].DataType == typeof(String))
      {
        row[iColumn] = datarow[i].ToString();
      }
      else if (result.Columns[iColumn].DataType == typeof(Double))
      {
        row[iColumn] = double.TryParse(datarow[i].ToString(), out double datafromrow) ? datafromrow : -1;
      }
      iColumn++;
    }
  }
  result.Rows.Add(row);
}
```

Slika 31: Popunjavanje redaka *DataTable* objekta

Navedene dvije operacije kao rezultat imaju popunjenu *DataTable* tablicu s podacima iz JSON objekta dohvaćenog iz baze podataka, neovisno o kreiranom zahtjevu, odabranim stupcima te tipu podataka, što omogućuje ponovno korištenje ove metode za sve dohvate iz baze, bez obzira radi li se o segmentima, putevima ili prilagođenim upitima.

Nakon što je *DataTable* tablica popunjena, proces kreiranja liste željenih objekata (instanci modela) je trivijalan – potrebno je za svaki redak dohvatiti podatke iz odgovarajućih stupaca, provjeriti njihove vrijednosti, te pozvati konstruktor željenog objekta. Na slici 32 prikazan je programski kôd za kreiranje liste puteva iz *DataTable* objekta popunjenog pomoću kôda sa slike 30 i slike 31. Za *Trip* objekt potrebno je kreirati dva *Point* objekta – jedan za početnu, a drugi za završnu točku, te za svaki *Point* objekt jedan *Coordinate* objekt. Konstruktor

Coordinate objekta se poziva za cjelobrojne vrijednosti koordinata, dobivenih iz baze podataka, te unutar istog se iste pretvaraju u konvencionalne decimalne vrijednosti, prikazano ranije na slici 26.

```
public static List<Trip> CreateTripList(string input)
{
  DataTable result = CreateResponse(input);

  List<Trip> trips = new List<Trip>();

  if (result.Rows.Count == 0)
  {
    return trips;
  }

  foreach (DataRow row in result.Rows)
  {
    int vid = Int32.TryParse(row["VID"].ToString(), out int VehicleID) ? VehicleID : -1;
    int t0 = Int32.TryParse(row["T0"].ToString(), out int time0) ? time0 : -1;
    int t1 = Int32.TryParse(row["T1"].ToString(), out int time1) ? time1 : -1;
    int x0 = Int32.TryParse(row["X0"].ToString(), out int startx) ? startx : -1;
    int y0 = Int32.TryParse(row["Y0"].ToString(), out int starty) ? starty : -1;
    int x1 = Int32.TryParse(row["X1"].ToString(), out int endx) ? endx : -1;
    int y1 = Int32.TryParse(row["Y1"].ToString(), out int endy) ? endy : -1;
    double len = double.TryParse(row["LEN"].ToString(), out double lenght) ? lenght : -1;

    Coordinate startCoordinate = new Coordinate(null, x0, null, y0);
    Coordinate endCoordinate = new Coordinate(null, x1, null, y1);

    DateTimeOffset startTime = DateTimeOffset.FromUnixTimeSeconds(t0);
    DateTimeOffset endTime = DateTimeOffset.FromUnixTimeSeconds(t1);


    Point startPoint = new Point(startCoordinate, null, startTime);
    Point endPoint = new Point(endCoordinate, null, endTime);

    Trip trip = new Trip(vid, startPoint, endPoint, len/1000);

    trips.Add(trip);

  }

  trips = trips.OrderBy(trip => trip.Start.Time).ToList();

  DBCache.LastTripsLoad = trips.ToList();

  return trips;
}
```

Slika 32: Kreiranje liste *Trip* objekata iz *DataTable* tablice

Kreiranjem liste puteva, ili bilo kojih drugih traženih objekata – s obzirom da je proces jednak za sve dohvate, završava proces obrade dolaznih podataka iz baze, te započinje proces kreiranje odgovora i pohrane često korištenih podataka u memoriju.

Liste puteva se često dohvaćaju, te često sadrže iste instance puteva koji se tada nepotrebno propagiraju između API-ja i korisničkog sučelja – iz tog razloga kreirana je klasa DBCache, koja sadrži funkcionalnosti za pohranu dohvaćenih puteva i jednoznačnu identifikaciju istih, te metodu za provjeru je li dohvaćeni *Trip* objekt već poslan u nekom odgovoru prema klijentu.

Pohrana podataka vrši se na način da se lista posljednje dohvaćenih puteva sprema u memoriju, kako bi se dalje mogla koristiti za kreiranje grafikona i druge eventualne obrade bez potrebe za ponovnim učitavanjem istih podataka iz baze podataka. Također, prilikom kreiranja odgovora, provjerava se svaki zaseban objekt *Trip* – ukoliko je već pohranjen u memoriji, isti se ne uključuje u odgovor koji se šalje prema klijentu, već se preskače jer već postoji na korisničkom sučelju.

Jedan put može sadržati značajno velik broj segmenata, čiji dohvat iz baze i obrada, mogu potrajati – stoga je mehanizam pohrane proširen i za pohranu segmenata – na način da se pohranjuju dohvaćeni segmenti za svaki put. Prilikom iniciranog zahtjeva sa klijentske strane za dohvat svih segmenata određenog puta, provjerava se postoje li isti u pohrani, te se na taj način za već dohvaćene segmente preskače kreiranje i izvršavanje upita na bazi, već se za iste izrađuje odgovor na temelju podataka već pohranjenih u memoriji, što znatno ubrzava prikaz istih na korisničkom sučelju – ukoliko su već barem jednom bili učitani.

4.2.3 Kreiranje odgovora – GeoJSON

Za kreiranje odgovora prema klijentu, zbog jednostavnosti prikaza i razmjene geografskih podataka između API-ja i GUI-ja, koristi se *GeoJSON* format – prostorno-vremenski format za razmjenu podataka temeljen na JavaScript objektnom sustavu označavanja (engl. JSON – *JavaScript Object Notation*). *GeoJSON* format kreiran je 2016. godine od strane IETF – *Internet Engineering Task Force*, kao prijedlog standarda RFC-7946 (engl. RFC – *Request for Comments*). *GeoJSON* standard koristi više tipova JSON objekata koji predstavljaju podatke o geografskim svojstvima, njihove značajke i prostorne veličine (engl. „*Extents*“). Za prikaz geografskih podataka koriste se stupnjevi u decimalnom obliku, te referentni geografski koordinatni sustav WGS84[13], [48].

GeoJSON specifikacija podržava sljedeće geometrijske objekte: točka (engl. „*Point*”), linija (engl. „*LineString”*), mnogokutnik (engl. „*Polygon*”), višestruka točka (engl. „*MultiPoint*”), višestruka linija (engl. „*MultiLineString*”), višestruki mnogokutnik (engl. „*MultiPolygon*”). Ukoliko se geometrijskim objektima želi dodati i dodatna svojstva, ista se stavljaju u objekt naziva svojstvo (engl. „*Feature*”), te se ista mogu kombinirati u jedan objekt tipa kolekcija svojstva (engl. „*FeatureCollection*”), [49].

[13] *World Geodetic System* 1984 – je referentni koordinatni sustav koji se sastoji od referentnog elipsoid, standardnog koordinatnog sustava, podataka o visini i geoida, namjena mu je aproksimacija Zemljinog stvarnog oblika. [54]

Primjer jednog *GeoJSON* objekta, koji sadrži jedan *Feature* objekt, geografskog tipa *Point*, te dodatne opisne atribute, prikazan je na slici 33:

Slika 33: Primjer *GeoJSON* objekta

DBResponse klasa za kreiranje odgovora koristi GeoJSON.Net *NuGet* paket, koji omogućuje jednostavno kreiranje i korištenje ranije navedenih *GeoJSON* objekata. Primjer programskog kôda za generiranje *GeoJSON* odgovora za kreiranu listu puteva iz primjera u prošlom odlomku dan je na slici 34. Za svaki put u listi generiraju se *GeoJSON* pomoćni objekti tipa *Position* – potreban za kreiranje svih ostalih geografskih objekata, npr. linije, točke, i dr. *Position* objekt sadrži atribute za geografsku širinu, duljinu i nadmorsku visinu. Nakon što su kreirane točke i linije, pomoću ranije kreiranih *Position* objekata za početak i kraj, kreiraju se *Feature* objekti, kako bi se na iste dodala dodatna opisna svojstva. Na svaki objekt, tj. točke početka i kraja, te liniju koja ih spaja, kao opisno svojstvo dodaju se vrijednosti svojstava *Trip* objekta na kojeg se iste odnose (samo referenca), te *Type* parametar koji označuje je li dodana točka početak ili kraj navedenog puta. Naposljetku, tri navedena *Feature* objekta dodaju se u kolekciju *FeatureCollection*, te se isto ponavlja za idući *Trip* objekt u listi puteva. Ukoliko je za određeni *Trip* objekt već generiran *GeoJSON* objekt – ako se navedeni objekt nalazi u DBCache, isti se preskače jer se smatra već učitanim na grafičkom sučelju.

```
public static string CreateGeoJSONResponseTrips(List<Trip> trips)
{
  DBCache.LastTripsLoad = trips.ToList();

  GeoJSON.Net.Feature.FeatureCollection features = new GeoJSON.Net.Feature.FeatureCollection();

  foreach (Trip trip in trips)
  {
    if (trip is null || !DBCache.CheckTripUnique(trip))
        continue;

    Position start = new Position((double)trip.Start.Coordinate.Latitude.Value, (double)trip.Start.Coordinate.Longitude.Value);
    Position end = new Position((double)trip.End.Coordinate.Latitude.Value, (double)trip.End.Coordinate.Longitude.Value);

    GeoJSON.Net.Geometry.Point startPoint = new GeoJSON.Net.Geometry.Point(start);
    GeoJSON.Net.Geometry.Point endPoint = new GeoJSON.Net.Geometry.Point(end);

    GeoJSON.Net.Geometry.LineString line = new GeoJSON.Net.Geometry.LineString(new List<IPosition> { start, end });

    var featureStart = new GeoJSON.Net.Feature.Feature(startPoint, new { trip, type = "start" });
    var featureEnd = new GeoJSON.Net.Feature.Feature(endPoint, new { trip, type = "end" });
    var featureLine = new GeoJSON.Net.Feature.Feature(line, new { trip });

    features.Features.AddRange(new List<GeoJSON.Net.Feature.Feature> { featureStart, featureEnd, featureLine });
  }

  return JsonConvert.SerializeObject(features);
}
```

Slika 34: Generiranje *GeoJSON* objekta za listu puteva

4.2.4 Povezani i prostorni upiti

Osim liste svih puteva na nekom području u nekom vremenu, za neko specifično vozilo, moguće je dohvatiti i sve puteve koji zadovoljavaju određene uvjete. Podržani dohvati su:

- *INTERSECT* – putevi čiji segmenti presijecaju određeno definirano područje (geografski objekt),
- *START* – putevi čija je početna točka unutar definiranog područja,
- *END* – putevi čija je krajnja točka unutar definiranog područja,
- *BOTH* – putevi čije su početne i krajnje točke unutar definiranog područja.

Sve navedene dohvate moguće je još dodatno međusobno povezati, definiranjem više područja u zahtjevu za dohvat, te povezati iste klasičnim logičkim operatorima: i (engl. „*AND*“) te ili (engl. „*OR*“). Tako je moguće kreirati zahtjev za dohvat puteva čija je početna točka unutar jednog područja, te koji presijecaju drugo područje, a završavaju unutar trećeg područja. U teoriji, ne postoji limit na broj ulančanih uvjeta koje je moguće postaviti – uz pretpostavku da u bazi postoje podaci koji zadovoljavaju takav upit.

Na slici 35 prikazan je primjer *JSON* objekta koji se šalje uz zahtjev za dohvat puteva čija se početna točka nalazi unutar prvog mnogokuta, te koji presijecaju drugi mnogokut. Kao i ostali zahtjevi za dohvat puteva, ovaj zahtjev sadrži *Trip* objekt, koji predstavlja *Trip* model, te unutar kojeg se postavljaju uvjeti relevantni za podatke o samom Tripu, kao što su npr. ID

vozila, ili u ovom primjeru, vremena između kojih se nalaze željeni putevi za dohvat. Osim *Trip* objekta, zahtjev sadrži *GeoJSON* objekt, koji sadrži podatke o geografskim objektima, te *Settings* objekt unutar kojega je definiran način dohvata za pojedini geografski objekt – tako se u ovom primjeru za prvi objekt dohvaćaju svi putevi čija se početna točka nalazi unutar istog, a istovremeno segmenti presijecaju područje drugog objekta.

```
{
    "trip": {
        "start": {
            "time": "2014-01-01T00:00"
        },
        "end": {
            "time": "2014-01-01T23:59"
        },
    },
    "geojson": {
        "type": "FeatureCollection",
        "features": [
            {
                "type": "Feature",
                "properties": {},
                "geometry": {
                    "type": "Polygon",
                    "coordinates": [
                        //cordinates array
                    ]
                }
            },
            {
                "type": "Feature",
                "properties": {},
                "geometry": {
                    "type": "Polygon",
                    "coordinates": [
                          //cordinates array
                    ]
                }
            }
        ]
    },
    "settings": [
        {
            "id": 0,
            "operation": "Start",
            "connector": "None"
        },
        {
            "id": 1,
            "operation": "Intersect",
            "connector": "AND"
        }
    ]
}
```

Slika 35: Primjer zahtjeva za dohvat podataka unutar više područja

U trenutku zaprimanja ovakvog zahtjeva, upravljač serijalizira podatke iz istog u odgovarajuće ranije navedene modele, te provjerava koliko *GeoJSON* objekata se nalazi u zahtjevu. Ukoliko postoji samo jedan objekt, kreira se samo jedan dohvat, na način objašnjen u prijašnjem poglavlju. Ukoliko je navedeno više od jednog *GeoJSON* objekta u zahtjevu, postupak je kako slijedi, a programski kôd unutar klase DBFetch za dohvat istog prikazan je na slici 37:

1. Za svaki *GeoJSON Feature* objekt provjerava se odgovarajući zapis u *Settings* objektu,
2. Na temelju atributa *Operation* u objektu *Settings* pomoću DBQuery i DBFetch dohvaćaju se podaci za isti – vrijednosti *Operation* svojstva mogu biti: *start*, *end*, *both* ili *intersect*, o čemu direktno ovisi izlazni SQL upit iz DBQuery,
3. Za svaku dohvaćenu listu *Trip* podataka (koji se dohvaćaju za svaki *Feature* kako je navedeno u prethodnom koraku), vrši se spajanje lista – na način da se provjerava vrijednost C*onnector* atributa – koja može biti AND ili OR – ukoliko je vrijednost AND, liste se spajaju na način da izlazna lista sadrži samo *Trip* objekte koji se nalaze u obje liste, a u slučaju OR, izlazna lista sadrži sve objekte iz obje liste.
4. *Trip* objekti iz izlazne liste se pretvaraju u *GeoJSON* objekte, te se kao takvi šalju u odgovoru nazad klijentu, kao što je objašnjeno u prethodnom poglavlju.

Spajanje lista na ovaj način nije optimalno, jer za svaki *GeoJSON* objekt iz dohvata se kreira zaseban upit i zahtjev nad bazom podataka, ali zbog ranijih navedenih nedostataka prilikom korištenja SQLKata paketa za generiranje upita, te zbog korištenje prostornih funkcija, je dovoljno brzo rješenje za točan dohvat istih.

SQL upiti za dohvat ovih podataka kreirani su ručno, u ovoj implementaciji samo za *GeoJSON* tip poligon, te za sve od ranije navedenih operacija. Kreiranje SQL upita vrši se u klasi DBQuery, te se isti generira za svaki *GeoJSON Feature* objekt dobiven u zahtjevu. Ručno napisani SQL upiti za sve operacije, prikazani su na slici 37.

U upitima se koriste dvije prostorne funkcije, ST_CONTAINS i ST_INTERSECTS, koje obje primaju dva parametra – dva geometrijska objekta, te vraćaju vrijednost istine ili laži. ST_CONTAINS vraća istinu ukoliko objekt dan kao drugi parametar sadrži objekt dan kao prvi parametar, dok ST_INTERSECTS vraća istinu ukoliko se geometrijski objekti dani kao parametri dodiruju.

Za kreiranje geometrijskih objekata koriste se prostorni konstruktori ST_POINT – za kreiranje točke, te ST_LINE za kreiranje linije.

```
switch (settings.Operation)
{
  case Operation.Both:
    return $"select * from st.trips where t[0] >= {trip.Start.Time.Value.ToUnixTimeSeconds()} AND t[1] <= {trip.End.Time.Value.ToUnixTimeSeconds()} AND ST_CONTAINS({coords}, ST_Point(x0,y0)) AND ST_CONTAINS({coords}, ST_Point(x1,y1))";
  case Operation.Start:
    return $"select * from st.trips where t[0] >= {trip.Start.Time.Value.ToUnixTimeSeconds()} AND t[1] <= {trip.End.Time.Value.ToUnixTimeSeconds()} AND ST_CONTAINS({coords}, ST_Point(x0,y0))";
  case Operation.End:
    return $"select * from st.trips where t[0] >= {trip.Start.Time.Value.ToUnixTimeSeconds()} AND t[1] <= {trip.End.Time.Value.ToUnixTimeSeconds()} AND ST_CONTAINS({coords}, ST_Point(x1, y1))";
  case Operation.Intersect:
    return ($"(select b.vid, b.t, b.x0, b.y0, b.x1, b.y1, b.len from st.trips as b, (SELECT * from st.segments where ST_INTERSECTS({coords}, ST_Line(x[0],y[0],x[1],y[1])) AND t[0] >= {trip.Start.Time.Value.ToUnixTimeSeconds()} AND t[1] <= {trip.End.Time.Value.ToUnixTimeSeconds()}) as a where b.vid = a.vid AND b.t[0] >= {trip.Start.Time.Value.ToUnixTimeSeconds()} AND b.t[1] <= {trip.End.Time.Value.ToUnixTimeSeconds()} and ((a.t[0] >= b.t[0] AND a.t[0] <= b.t[1]) and (a.t[1] >= b.t[0] AND a.t[1] <= b.t[1])) group by b.vid, b.t, b.x0, b.y0, b.x1, b.y1, b.len)");
  default:
    break;
}
```

Slika 36: SQL upiti za dohvat puteva ovisno o operaciji

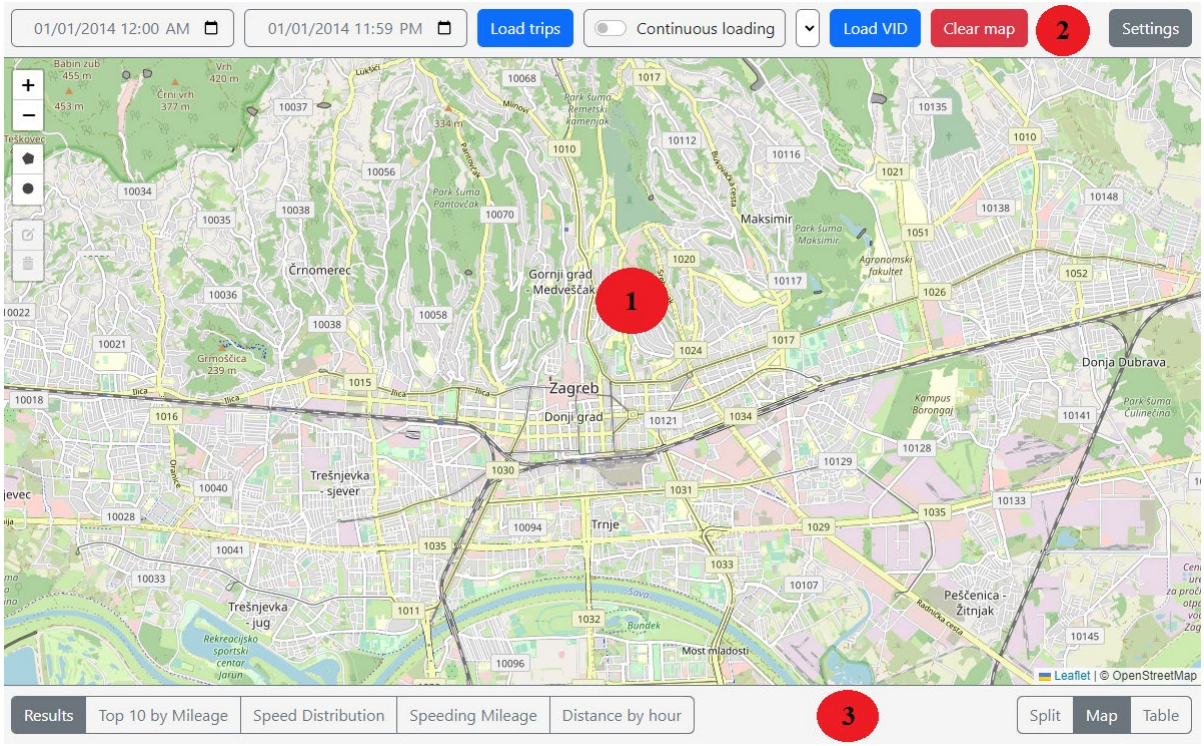
```
        public static async Task<string> GetTripsInGeoJSON(Trip trip, GeoJSON.Net.Feature.FeatureCollection geoFeature, Dictionary<int,LoadSettings> loadProperties)
        {
            DBOperations db = null;
            List<List<Trip>> results = new List<List<Trip>>();
            for (int i = 0; i < geoFeature.Features.Count; i++)
            {
                var query = DBQuery.GetTripsInGeoJSON(trip, geoFeature.Features[i], loadProperties[i]);

                using var client = new HttpClient();

                var url = "http://192.168.202.222:1248/query";

                var data = new StringContent(query);

                var response = await client.PostAsync(url, data);

                if (response is not null)
                {
                    if (response.IsSuccessStatusCode && !(response.Content is null))
                    {
                        string resultt = await response.Content.ReadAsStringAsync();

                        var result = DBResponse.CreateTripList(resultt);

                        results.Add(result);
                    }
                }

                if (loadProperties[i].Connector is Connector.None)
                {
                    db = new DBOperations(results[i]);
                }

            }

            db ??= new DBOperations(results[0]);

            for (int i = 0; i < results.Count; i++)
            {
                switch (loadProperties[i].Connector)
                {
                    case Connector.None:
                        db.Or(results[i]);
                        break;
                    case Connector.And:
                        db.And(results[i]);
                        break;
                    case Connector.Or:
                        db.Or(results[i]);
                        break;
                    default:
                        db.Or(results[i]);
                        break;
                }
            }

            DBCache.Reset();
            return DBResponse.CreateGeoJSONResponseTrips(db.Get());
        }
```

Slika 37: Programski kôd za dohvat i spajanje puteva iz višestrukih prostornih zahtjeva

4.3. Grafičko korisničko sučelje

Grafičko korisničko sučelje, prikazano slici 38, čine sljedeći dijelovi:

1. Karta – element sučelja zadužen za prikaz karte, te prikaz učitanih podataka na istoj, sastoji se od same karte, alata za dodavanje elemenata na istu, te elemenata koji predstavljaju učitane podatke,
2. Funkcijska traka – traka na vrhu sučelja sa funkcionalnošću za učitavanje podataka, definiranje filtera za učitane podatke, brisanje učitanih podataka sa mape, te postavljanje posebnih prostornih upita, objašnjenih poglavljem 4.2.4, definiranih dodanim elementima na karti,
3. Traka pogleda – dno sučelja sa mogućnostima izbora željenog prikaza – tabličnog prikaza rezultata ili prikaz predefiniranih grafikona.



Slika 38: Grafičko korisničko sučelje

Grafičko korisničko sučelje izrađeno je korištenjem HTML i JavaScript jezika, te je za razvoj odabran *Visual Studio Code*[14].

[14] Visual Studio Code je softver za uređivanje teksta razvijen od strane Microsofta, podržava razne programske jezike i operativne sustave, te sadrži veliki broj dostupnih nadogradnji i alata.

4.3.1 Prikaz podataka na karti

Karta je implementirana pomoću *Leaflet* JavaScript biblioteke (engl. „*library*“) – *Leaflet* je vodeća biblioteka za implementaciju interaktivnih karata, usmjerena na jednostavnost, brzinu i uporabljivost. Za prikaz karte moguće je odabrati jedan od raznih podržanih pružatelja karata, kao što su *OpenStreetMap*, *MapBox*, *Bing Maps*, *Esri ArcGIS*, *MapQuest* te *Here Maps*. Sadrži mnoštvo dodataka, iako i u osnovnoj izvedbi sadrži sve potrebne funkcionalnosti za rad i prikaz podataka na karti, [50].

Leaflet biblioteka odabrana je zbog detaljno napisane dokumentacije i uputa, jednostavnosti implementacije te performansi. Za izvor karte odabran je davatelj usluge *OpenStreetMap,* jer sadrži javne podatke, prikupljene i objavljene od strane zajednice, te jer je u potpunosti besplatno rješenje.

Implementacija karte na mrežnu stranicu koristeći JavaScript jezik, prikazana je na slici 39:

```
1 var map = L.map('map').setView([45.815, 15.9819], 13);
2 L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
3     maxZoom: 19,
4     attribution: '© OpenStreetMap',
5 }).addTo(map);
```

Slika 39: Implementacija Leaflet karte koristeći JavaScript

Za potrebe prikazivanja podataka, dohvaćeni podaci na kartu se dodaju kao slojevi. U ovom primjeru, kako je ranije navedeno, podaci se dohvaćaju kao kolekcija GeoJSON objekata, odnosno jedan GeoJSON objekt predstavlja jedan sloj. Dohvaćeni GeoJSON objekti mogu biti tipa Point ili LineString, gdje Point-ovi predstavljaju početnu i krajnju točku putovanja, a LineString vezu između istih.

Kroz Leaflet biblioteku moguće je definirati grupu slojeva nastalih iz GeoJSON kolekcije, te definirati stil i funkcije koje će se izvršavati na svakom od dodanih slojeva unutar iste.

Na slici 40 prikazana je takva definicija grupe slojeva za prikaz dohvaćenih podataka o putevima. Kroz *pointToLayer* funkciju definira se ponašanje za dodavanje Point-ova. U navedenoj funkciji mijenja se početno zadano ponašanje, da se Point na kartu dodaje kao Marker – koji je u Leafletu predefinirana ikona koju nije moguće uređivati, već se isto dodaje kao kružnica, kojoj je moguće mijenjati boju i druga svojstva. *OnEachFeature* svojstvo grupe slojeva omogućuje postavljanje funkcije ili svojstva koje će se izvršiti nad svakim dohvaćenim GeoJSON objektom u kolekciji. U ovom slučaju, definira se funkcija koja će nad svakim

kreiranim slojem kreirati skočni izbornik, popunjen sa informacijama o tom specifičnom putovanju, te unutar istog dodati gumb, pritiskom na koji će se izvršiti neka druga funkcija – odnosno učitavanje i prikaz segmenata tog putovanja na karti. *Style* svojstvo omogućuje definiranje svojstava prikaza sloja za svaki pojedini dohvaćeni objekt, te se isto koristi za prikaz točke, koja se kao što je ranije navedeno ucrtava kao kružnica, zelenom bojom ukoliko se radi o početku putovanja, odnosno crvenom ukoliko se radi o kraju. Osim navedenog za tip Point, moguće je postaviti i drugačiji format prikaza za LineString, npr. da boja same linije ovisi o duljini putovanja na koje se odnosi, i dr.

```
let layerGroupTrips = L.geoJSON(null, {
   pointToLayer: function(feature, latlng){
      return L.circleMarker(latlng, geojsonMarkerOptionsTrips);
      },
  onEachFeature: function (feature, layer) {
   let pop = document.createElement('div');
   let btnLoad = document.createElement('button');
   btnLoad.innerHTML = "Load segments";
   btnLoad.onclick = function () {
      loadSegForTrip(feature.properties.trip);
   };
   let startTime = new Date(feature.properties.trip.Start.Time);
   let endTime = new Date(feature.properties.trip.End.Time);
   let ihtml = '<h1>ID: '+feature.properties.trip.VehicleID+'</h1><h3>Length: '+ feature.properties.trip.Length +' km </h3>'+'<h3>Start:
'+startTime.toLocaleString("hr-HR")+'</h3>'+'<h3>End: '+endTime.toLocaleString("hr-HR")+'</h3>'
   pop.innerHTML = ihtml;
   pop.append(btnLoad);
   layer.bindPopup(pop);
},
  style: function(feature){
   if(feature.geometry.type === 'Point'){
      switch(feature.properties.type){
         case 'start': return {fillColor: "#118314", color: "#118314"};
         case 'end' : return {fillColor: "#b1060c", color: "#b1060c"};
      }
   }
}
}).addTo(map);
```

Slika 40: Definicija grupe slojeva za ucrtavanje puteva na karti

Navedeno znatno olakšava razvoj, jer je moguće u potpunosti odvojiti logiku dohvata podataka od same logike prikazivanja dohvaćenih podataka na karti. Navedena grupa slojeva definirana je na razini cijelog sučelja, što znači da će se ranije navedene funkcije primjenjivati u svakom trenutku dodavanja, te na svaki sloj koji se dodaje u grupu.

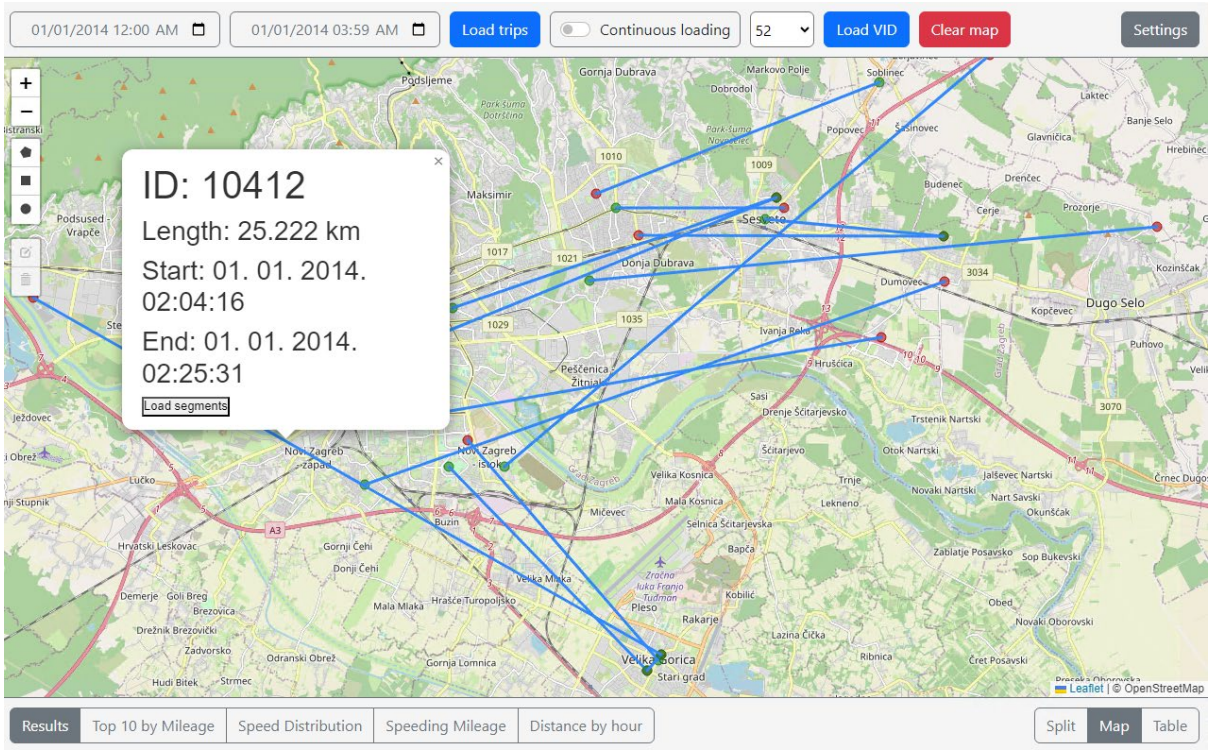
Za prikaz segmenata koji čine putovanje, kreirana je zasebna grupa slojeva – prikazana na slici 41, sa sličnim opcijama kao i grupa slojeva za putovanja. Za Point objekte primjenjuju se iste izmjene kao i u primjeru za putovanja, dok se u ovom slučaju za LineString objekte ucrtavaju linije obojane ovisno o prekoračenju ograničenja brzine na specifičnom segmentu – ukoliko je ispod ograničenja, isti se ucrtava plavom bojom, a ukoliko je prekoračeno ograničenje, ucrtava se crvenom bojom.

```
let layerGroupSegs = L.geoJSON(null, {
   pointToLayer: function(feature, latlng){
   return L.circleMarker(latlng, geojsonMarkerOptionsSegs);
   },
   style: function(feature){
   if(feature.geometry.type === 'Point'){
      switch(feature.properties.type){
         case 'start': return {fillColor: "#118314", color: "#118314", radius: 10};
         case 'end' : return {fillColor: "#b1060c", color: "#b1060c", radius: 10};
      }
   }
   else
   {
      if(feature.properties.sg.SpeedLimit > 0){
         if(feature.properties.sg.Start.Velocity > feature.properties.sg.SpeedLimit || feature.properties.sg.End.Velocity > feature.properties.sg.SpeedLimit){
            return {color: "#ff0000", weight: 3};
         }
         else{
            return {color: "#0088ff", weight: 3};
         }
      }
      else
      {
         return {color: "#0088ff", weight: 3};
      }
   }

 },
  onEachFeature: function (feature, layer) {
   let pop = document.createElement('div');

   let startTime = new Date(feature.properties.sg.Start.Time);
   let endTime = new Date(feature.properties.sg.End.Time);
   let ihtml = '<h1>ID: '+feature.properties.sg.VehicleID+'</h1>'+'<h3>Length: '+feature.properties.sg.Length+' m </h3>'+'<h3>Speed Limit:
'+feature.properties.sg.SpeedLimit+' km/h </h3>'+'<h2>Start</h2>'+'<h3>Time: '+startTime.toLocaleString("hr-HR")+'</h3>'+'<h3>Speed:
'+feature.properties.sg.Start.Velocity+' km/h</h3>'+'<h2>End</h2>'+'<h3>Time: '+endTime.toLocaleString("hr-HR")+'</h3>'+'<h3>Speed:
'+feature.properties.sg.End.Velocity+' km/h</h3>'
   pop.innerHTML = ihtml;
   layer.bindPopup(pop);
   },
   pane: 'paneSeg'
}).addTo(map);
```

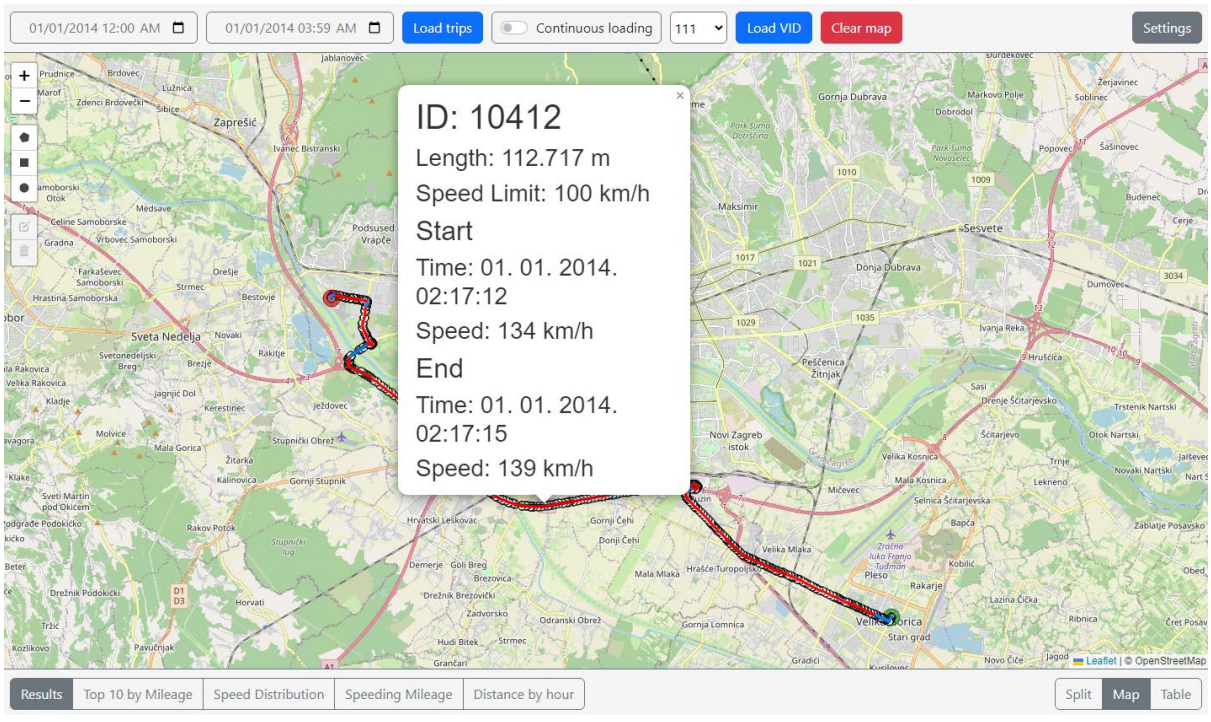
Slika 41: Definicija grupe slojeva za ucrtavanje segmenata na karti

Prilikom ucrtavanja segmenata na kartu, osim samih linija i točaka, ucrtava se i animirana putanja od početka do kraja putovanja na kojeg se segmenti odnose, implementirana korištenjem *Leaflet Ant Path* dodatka.

Rezultat navedenih funkcija za prikaz na karti prikazan je slikom 42 za prikaz putovanja i slikom 43 za prikaz segmenata određenog putovanja na karti – odnosno rezultat pritiska gumba „*Load segments*“ unutar skočnog prozora na slici 42.



Slika 42: Prikaz putovanja na karti



Slika 43: Prikaz segmenata određenog putovanja na karti

Osim samog prikaza podataka, element karte sadrži i mogućnost crtanja, odnosno dodavanja elemenata na istu u svrhu učitavanja podataka unutar specifičnih područja. Dodavanje elemenata na kartu implementirano je pomoću dodatka *Leaflet Draw*, na način

prikazan slikom 44. Podržani elementi za dodavanje u ovom rješenju su kružnice i mnogokuti. Prilikom kreiranja alatne trake potrebno je specificirati grupu značajki, odnosno grupu slojeva kojima će pripadati nacrtani elementi, te je u tu svrhu kreirana grupa *drawnItems*.

```
var drawnItems = new L.FeatureGroup(null, {
   pane: 'paneEdit'
});

map.addLayer(drawnItems);

var drawControl = new L.Control.Draw({
   draw: {
         polyline: false,
         marker: false,
         rectangle : false
   },
    edit: {
        featureGroup: drawnItems
    },
    pane: 'paneEdit'
});

map.addControl(drawControl);
```

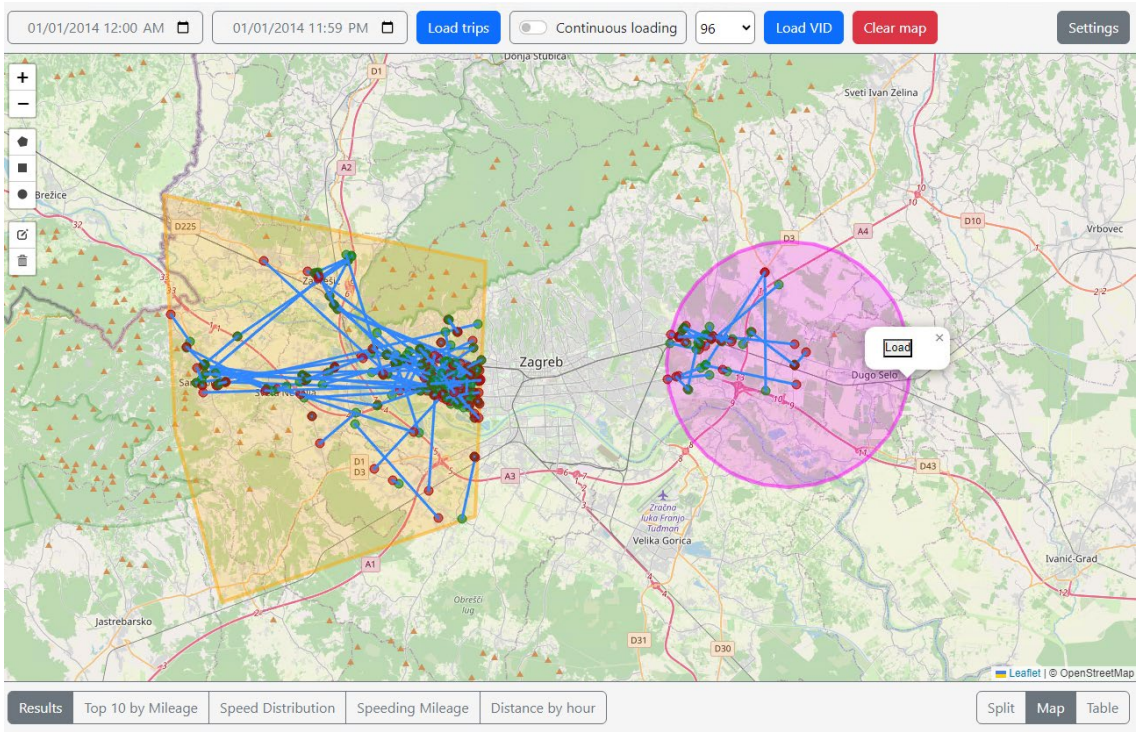
Slika 44: Dodavanje alatne trake za crtanje na *Leaflet* kartu

Svaki nacrtani element prilikom dodavanja boja se nasumičnom bojom iz niza predefiniranih boja, te mu se dodaje skočni prozor s gumbom *Load* kojim je omogućen dohvat putovanja koja se nalaze unutar istog, te prikaz istih na karti. Pritiskom gumba *Load* izvršava se učitavanje puteva čija je početna i krajnja točka unutar ucrtanog objekta, a isto je moguće promijeniti kroz postavke, objašnjene u idućem poglavlju. Takav zahtjev prikazan je ranije na slici 35 i slici 37, iz kojih je vidljivo da se u zahtjevu sa sučelja šalje GeoJSON objekt. Dopušteni elementi za dodavanje na kartu su mnogokuti i kružnice, od kojih je samo mnogokut dostupan kao GeoJSON objekt, stoga se kružnica mora pretvoriti u približan prikaz mnogokutom, korištenjem ugrađene metode iz *Leaflet Draw* dodatka – *circleToPolygon*. Navedena metoda pretvara kružnicu u mnogokut sastavljen od 60 točaka. Navedene operacije vrše se prilikom svakog dodavanja elementa na kartu, korištenjem događaja *L.Draw.Event.CREATED*, prikazano slici 45.

Na slici 46 prikazana su dva dodana elementa, lijevo mnogokut, te desno kružnica pretvorena u mnogokut, te učitani putevi unutar oba elementa.

```
map.on(L.Draw.Event.CREATED, function (e) {
   var randomColor;
   do {
      if(addedColors.length === colors.length){
         addedColors.length = 0;
      }
      randomColor = colors[Math.floor(Math.random()*colors.length)];
    } while (addedColors.includes(randomColor));


   e.layer.options.color = randomColor;
   addedColors.push(randomColor);

   var type = e.layerType,
      layer = e.layer;

   if(type === 'circle') {
      layer = circleToPolygon(layer);
   }

   let pop = document.createElement('div');

   let btnLoad = document.createElement('button');
   btnLoad.innerHTML = "Load";
   btnLoad.onclick = function(){
      var geoJSONstr;

      geoJSONstr = layer.toGeoJSON();

      loadTripsInGeoJSON(geoJSONstr);
   }

   pop.appendChild(btnLoad);
   layer.bindPopup(pop);

   drawnItems.addLayer(layer);
   needsRefresh = true;
});
```

Slika 45: Obrada slojeva prilikom dodavanja na kartu



Slika 46: Prikaz ucrtanih elemenata na karti te putovanja koja se nalaze unutar istih

4.3.2 Dohvat podataka, postavke dohvata i postavke prikaza

Za komunikaciju između grafičkog korisničkog sučelja i API-ja koristi se *fetch* funkcija JavaScript-a, te je kreirana funkcija *getData* za GET zahtjeve, te *postData* za POST zahtjeve, prikazano na slici 47. Za sve dohvate podataka koristi se POST zahtjev, u čije tijelo se postavljaju parametri zahtjeva, dok se za resetiranje stanja DBCache pohrane, objašnjene u poglavlju 4.2.2, koristi GET zahtjev.

```
// WEB REQUESTS - POST
async function postData(url = '', data = '') {
  const response = await fetch(url, {
    method: 'POST',
    mode: 'cors',
    cache: 'no-cache',
    credentials: 'same-origin',
    headers: {
      'Content-Type': 'application/json'
    },
    redirect: 'follow',
    referrerPolicy: 'origin',
    body: JSON.stringify(data)
  });
  return response.json();
}

// WEB REQUESTS - GET
async function getData(url = '') {
  const response = await fetch(url, {
    method: 'GET',
    mode: 'cors',
    cache: 'no-cache',
    credentials: 'same-origin',
    headers: {
      'Content-Type': 'application/json'
    },
    redirect: 'follow',
    referrerPolicy: 'origin'
  });
  return response.json();
}
```

Slika 47: POST i GET funkcije za dohvat podataka iz API-ja

Funkcijska traka sastoji se od HTML elemenata za unos vrijednosti, gumbi za pokretanje određenih funkcija, te potvrdnog okvira za uključivanje odnosno isključivanje mogućnosti kontinuiranog učitavanja podataka.

Koristeći dva elementa za odabir datuma, moguće je odabrati početni i krajnji datum između kojih će se dohvaćati željeni podaci o putevima. Za implementaciju istih korišteni su elementi za unos tipa *datetime-local*, kojima su postavljena ograničenja na unos vrijednosti – najraniji datum koji je moguće odabrati je ujedno i prvi datum za koji postoje podaci u bazi – 1.1.2014., a najkasniji datum je posljednji datum za koji postoje podaci u bazi – 1.7.2020.

Za provjeru postojanja podataka po datumima u bazi, korišten je program napisan u Pythonu, čiji programski kôd je prikazan na slici 48, koji dohvaća podatke iz baze dan po dan, od 1.1.2010., do 1.1.2023. te pohranjuje podatke o datumu, broju puteva te ukupnoj duljini puteva na taj datum u datoteku u CSV formatu[15].

```
import requests
import json
import time
import datetime

Date = datetime.date(2010,1,1)
endDate = datetime.date(2023,1,1)
startTime = datetime.time(0,0)
endTime = datetime.time(23,59)
day = datetime.timedelta(days = 1)

today = datetime.date.today()

result = '''date,count,length'''

while Date != endDate:
    startDateTime = datetime.datetime.combine(Date,startTime)
    endDateTime = datetime.datetime.combine(Date,endTime)
    epochBegin = int(startDateTime.timestamp())
    epochEnd = int(endDateTime.timestamp())

    queryStr = "SELECT count(len), sum(len) from st.trips where t[0] >= "+str(epochBegin)+"  AND t[1] <= "+str(epochEnd)
    URL = "http://192.168.202.222:1248/query"
    body = queryStr
    r = requests.post(url=URL, data=body)
    ex = r.text
    loaded_json = json.loads(json.loads(ex))
    dataDict = loaded_json['tables'][0]["data"]

    line = str(Date) + "," + str(dataDict[0][0]) + "," + str(dataDict[0][1])
    result = result + '\n' + line
    Date = Date + day

    print(line)

path = "Z:\Diplomski_rad\\"
fileName = "dates"
filePath= path + fileName + ".csv"
outputFile = open(filePath,'w')
outputFile.write(result)
outputFile.flush()
outputFile.close()
```

Slika 48: Programski kôd za provjeru postojanja podataka po datumima

Pritiskom na gumb *Load Trips* na mapi se ucrtavaju svi putevi unutar zadanog vremenskog perioda, koji presijecaju rubove trenutno vidljivog područja karte, te se popunjava padajući izbornik sa identifikatorima vozila. Kako bi se dohvatili samo oni na trenutno vidljivom području karte, kreira se JSON objekt koji se sastoji od Polygon objekta, koji predstavlja

[15] CSV – engl. „*Character Separated Values*“ – tablični format vrijednosti odvojenih određenim znakom, najčešće zarezom.

koordinate rubova trenutno vidljivog djela karte, koje se dohvaćaju preko Leaflet funkcije *map.getBounds*, te Trip objekta, unutar kojeg su pohranjeni parametri vezani za putovanje – vremena između kojih je potrebno dohvatiti putovanja, prikazano slici 49.

Pritiskom na gumb Load VID, na kartu se učitavaju samo podaci za vozilo odabrano u padajućem izborniku. Prilikom svakog učitavanja podataka, generira se i tablica dohvaćenih rezultata, koju je moguće prikazati na donjem dijelu sučelja.

Osim dugmadi za učitavanje podataka, na funkcijskoj traci nalazi se i opcija „Continuous Loading“ – koja iskorištava Leaflet događaj *moveend* te prilikom svake izmjene vidljivog područja karte ponovno učitava podatke koji presijecaju to, novo područje.

Gumb *Clear map* služi za resetiranje stanja mape, te pritiskom na isto se brišu svi prikazani putevi, te svi učitani ID-jevi vozila, uz što se poziva i API poziv za brisanje svih podataka iz ranije objašnjenog DBCache.

```
function loadTripsForBounds(){
      var coordinates = map.getBounds();
      var coordjson =
        [
          [
            coordinates._northEast.lat,
            coordinates._northEast.lng
          ],
          [
            coordinates._southWest.lat,
            coordinates._southWest.lng
          ]
        ];

      var result = makepolytripJSON(coordjson);

      getTripsInPolygon(result);
}

function getTripsInPolygon(polyTripJSON){
   postData('https://localhost:7051/trips/gettripsinpolygon', polyTripJSON)
    .then((data) => {
      showHideTrips(true);
      layerGroupTrips.addData(data);
      data.features.flatMap(item => item.geometry.type.includes("LineString") ? makeTripsTable(item.properties.trip) : null);
      sortTable();
      vehicleIDs.sort(function(a,b){return a-b});
      vehicleIDs.forEach(makeCBOptions)
    });
}
```

Slika 49: Dohvat putovanja koja sijeku trenutno vidljiv okvir karte

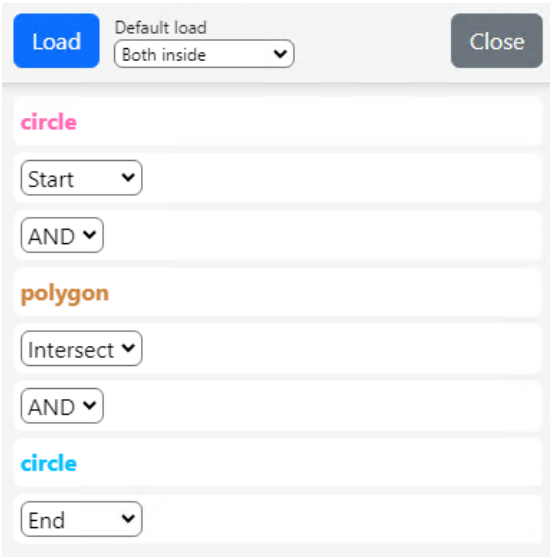
Na desnoj strani funkcijske trake nalazi se gumb *Settings*, koja sadrži opcije koje se odnose na elemente dodane na kartu, objašnjene u prijašnjem poglavlju. Pritiskom na isto, otvara se skočni prozor, unutar kojeg je moguće definirati zadanu opciju za učitavanje podataka za jedan

dodani (nacrtani) element na karti – „Default Load“, a moguće opcije su *Start*, *End*, *Both* i *Intersect*, već objašnjene ranije.

Osim postavljanja zadanog učitavanja za pojedine objekte, kroz isti je moguće kreirati i povezane prostorne upite, objašnjene u poglavlju 4.2.4.

Kako je objašnjeno u prijašnjem poglavlju, svaki dodani element dobiva svoju boju, te je potrebno definirati željeni dohvat za svaki od elemenata, te poveznicu između međusobnih elemenata. Poveznice mogu biti *AND* – mora biti zadovoljen uvjet postavljen na oba elementa, te *OR* – mora biti zadovoljen bilo koji od postavljenih uvjeta. Cijeli *Settings* prozor se rekreira svaki put kada se otvara, koristeći dostupne slojeve iz grupe *drawnItems*, gdje se za svaki sloj dodaju izbornici za opcije dohvata i vezu s drugim dohvatima. Pritiskom na dugme Load, poziva se API poziv *trips/GetTripsMultipleGEOJSON* te se u tijelo zahtjeva postavlja JSON objekt, koji se sastoji od Trips objekta, u kojem je specificirano vrijeme unutar kojeg je potrebno dohvatiti putovanja, *geojson* reprezentacija slojeva iz grupe *drawnItems* – odnosno elementi dodani na karti, te *settings* objekt u kojem su pohranjene odabrane opcije iz izbornika za opcije dohvata i veze s drugim dohvatima.

Primjer *Settings* izbornika za kreiranje povezanih dohvata prikazan je na slici 50, a primjer JSON objekt koji se predaje API-ju za zahtjev kreiran kroz navedene opcije izbornika prikazan je na slici 51.

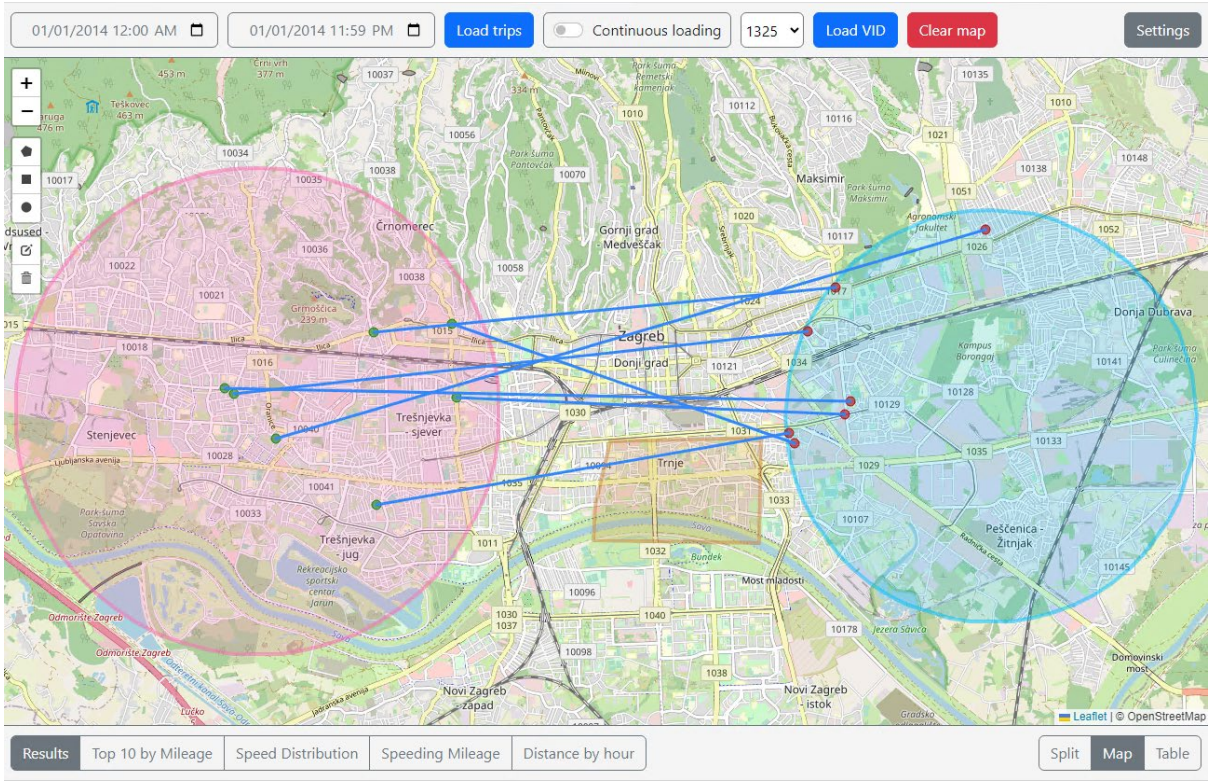


Slika 50: *Settings* izbornik za dohvat unutar višestrukih geometrijskih elemenata

```
{{
  "trip": {
    "start": {
      "time": "2014-01-01T00:00"
    },
    "end": {
      "time": "2014-01-01T23:59"
    }
  },
  "geojson": {
    "type": "FeatureCollection",
    "features": [
      {
        "type": "Feature",
        "properties": {},
        "geometry": {
          "type": "Polygon",
          "coordinates": [
           // koordinate točaka prve (ružičaste) kružnice
          ]
        }
      },
      {
        "type": "Feature",
        "properties": {},
        "geometry": {
          "type": "Polygon",
          "coordinates": [
            // koordinate točaka narančastog mnogokuta
          ]
        }
      },
      {
        "type": "Feature",
        "properties": {},
        "geometry": {
          "type": "Polygon",
          "coordinates": [
              // koordinate točaka druge (plave) kružnice
          ]
        }
      }
    ]
  },
  "settings": [
    {
      "id": 0,
      "operation": "Start",
      "connector": "None"
    },
    {
      "id": 1,
      "operation": "Intersect",
      "connector": "AND"
    },
    {
      "id": 2,
      "operation": "End",
      "connector": "AND"
    }
  ]
}}
```

Slika 51: JSON objekt kreiran za dohvat povezanog upita

Rezultat ovog dohvata, prikazan na slici 52, su svi putevi čija se početna točka nalazi unutar ružičastog kruga, čiji segmenti presijecaju narančasti mnogokut, te čija je završna točka unutar plavog kruga. S obzirom upit uzima u obzir sve puteve čiji segmenti presijecaju navedeni mnogokut, moguće je da vizualna reprezentacija samog puta ne presijeca isti, ali ukoliko se učitaju segmenti za takvo putovanje, biti će vidljivo da isti presijecaju mnogokut.



Slika 52: Rezultat povezanog prostornog upita

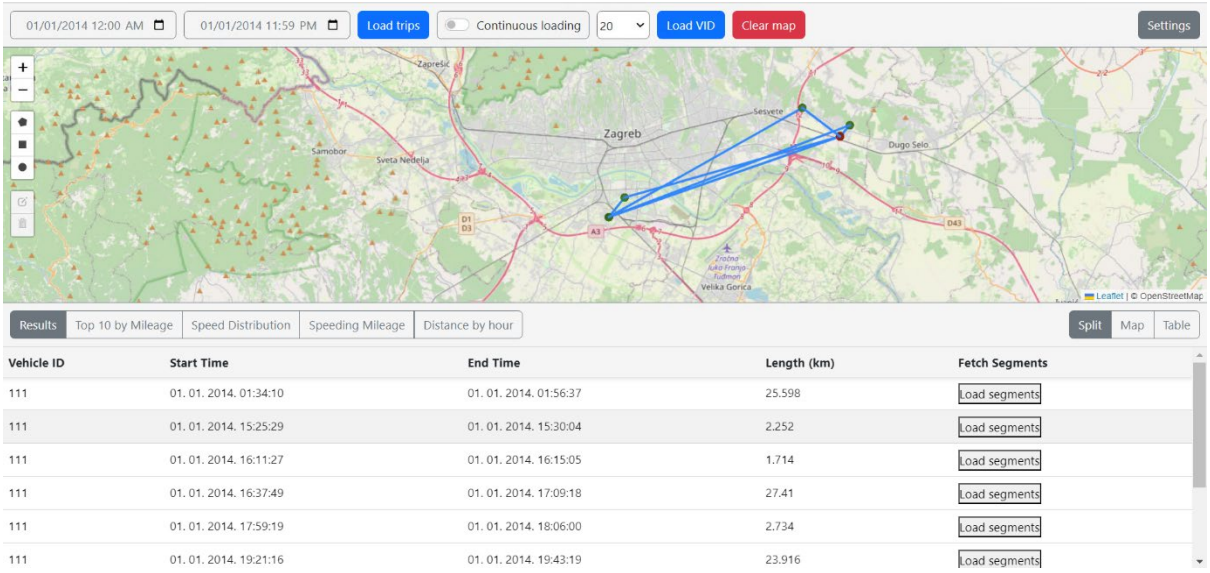
Traka pogleda sastoji se od dva skupa gumbi za odabir vrste željenog pogleda ispod same karte. Sa desne strane nalaze se opcije za postavljanje prostora kojeg zauzima sama karta:

- „Map“ – 90% vertikalnog prostora koristi se za prikaz karte,
- „Split“ – 50% vertikalnog prostora koristi se za prikaz karte, a preostali donji dio koristi se za tablični prikaz rezultata, ili prikaz grafova,
- „Table“ – primarno se prikazuju rezultati ili grafovi, karta zauzima 20% vertikalnog prostora.

Dok se s lijeve strane nalaze opcije za prikaz unutar prostora za rezultate, tablično ili kroz jedan od pred-definiranih grafova.

Nakon što su podaci učitani i prikazani, podaci o dohvaćenim putevima prikazani su u tablici na dnu sučelja, te je unutar iste također moguće napraviti dohvat segmenata za puteve.

Prikaz sučelja sa učitanim podacima, te odabranom opcijom „Split“ i odabranim tabličnim prikazom vidljiv je na slici 53.



Slika 53: Sučelje sa učitanim podacima

4.3.3 Grafikoni

Osim prikaza tablice sa rezultatima, moguće je prikazati rezultate u obliku grafikona, i to sljedećih tipova:

1. „Top 10 by Mileage“ – prikaz razdiobe ukupno prijeđene udaljenosti za deset vozila sa najvećom prijeđenom udaljenošću,
2. „Speed Distribution“ – Distribucija brzina dohvaćenih puteva za vozila,
3. „Speeding Mileage“ – Prikaz ukupno prijeđene udaljenosti po vozilu te udaljenosti prijeđene brzinom veće od ograničenja,
4. „Distance by Hour“ – Prikaz prijeđene udaljenosti u ovisnosti o vremenu u danu.

Grafovi su implementirani koristeći *Chart.js* biblioteku. *Chart.js* je jednostavna JavaScript biblioteka otvorenog kôda za kreiranje dijagrama i grafikona na mrežnim stranicama. Podržava kreiranje različite tipove grafikona, te ju je proširiva raznim dodacima. Ovo rješenje odabrano je zbog temeljito napisane dokumentacije, te brzine implementacije.

Grafikon se dodaje na način da se definira HTML *canvas* element, unutar kojeg će se prikazivati grafikon, te JSON objekt unutar kojega su definirane postavke prikaza grafikona i željeni podaci za prikaz.

S obzirom da API pohranjuje dohvaćene i trenutno prikazane podatke o putovanju vozila, dovoljno je u grafičkom korisničkom sučelju pozvati metodu za dohvat željenog grafikona –

koja na sučelje dostavlja odgovor unutar kojeg je sadržan ranije naveden JSON objekt. Primjer dohvata podataka za prikaz grafikona distribucije brzina prikazan je na slici 54, a primjer JSON objekta kreiranog u API-ju je prikazan slikom 55.

```
function getSpeedDistribution(){
  const ctx = document.getElementById('SpeedDistributionChart');
  getData('https://localhost:7051/Graph/SpeedDistribution').then((data) => {
      new Chart(ctx, data);
   });
}
```

Slika 54: Primjer dohvata podataka, kreiranja i prikaza grafikona distribucije brzina

```
{
    "type": "line",
    "data": {
        "labels": [
            "< 10",
            "10 - 20",
            "20 - 30",
            "30 - 40",
            "40 - 50",
            "50 - 60",
            "60 - 70",
            "70 - 80",
            "80 - 90",
            "90 - 100",
            "> 100"
        ],
        "datasets": [
            {
                "label": "All data",
                "data": [
                    4651, // podaci za brzine < 10
                    4912, // podaci za brzine 10 - 20
                    5085, // podaci za brzine 20 - 30
                    4022, // podaci za brzine 30 - 40
                    3147, // podaci za brzine 40 - 50
                    2836, // podaci za brzine 50 - 60
                    1732, // podaci za brzine 60 - 70
                    1327, // podaci za brzine 70 - 80
                    589,  // podaci za brzine 80 - 90
                    57,   // podaci za brzine 90 - 100
                    64    // podaci za brzine > 100
                ],
                "borderWidth": 1,
                "type": "line",
                "stack": "combined",
                "backgroundColor": "#ff0000"
            },
            {
                "label": 1347,
                "data": [
                    ...
                ],
                "order": 1,
                "type": "bar",
                "startsAtZero": true,
                "stack": 0,
                "backgroundColor": "#8925E2"
            },
            ...
        ]
    },
    "options": {
    }
}
```

Slika 55: Primjer JSON objekta za konfiguriranje chart.js grafikona

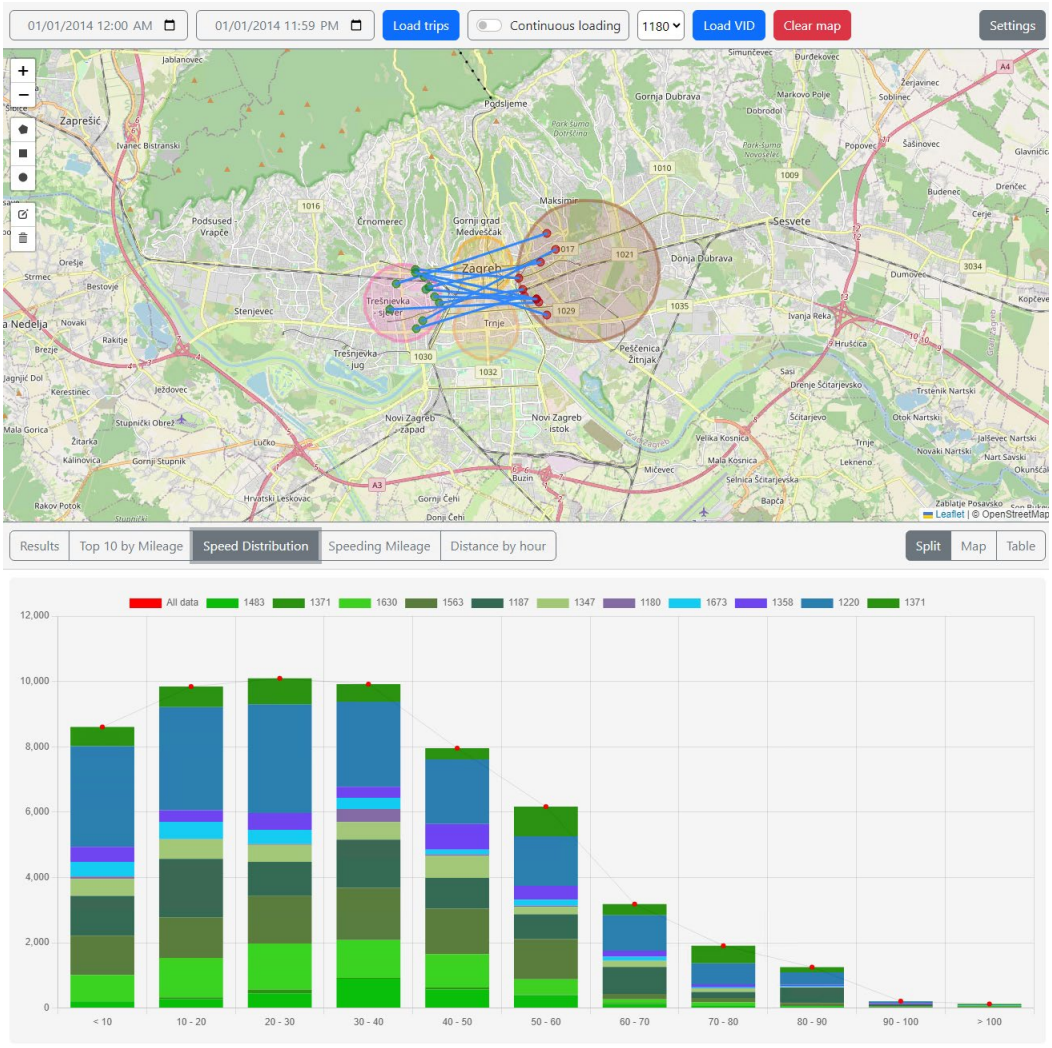
JSON objekt za konfiguriranje *Chart.js* grafikona, prikazan slici 55, sastoji se od sljedećih objekata:

- „*type*“ – predstavlja željeni tip grafikona, može biti linijski, stupčasti, prstenasti, i dr.,
- „*data*“ – objekt koji sadrži podatke za prikaz (engl. „*datasets*“), i opisne natpise (engl. „*labels*“) za iste,
- „*options*“ – dodatne opcije za grafikon kao što su naslov, opcije za osi, i dr.

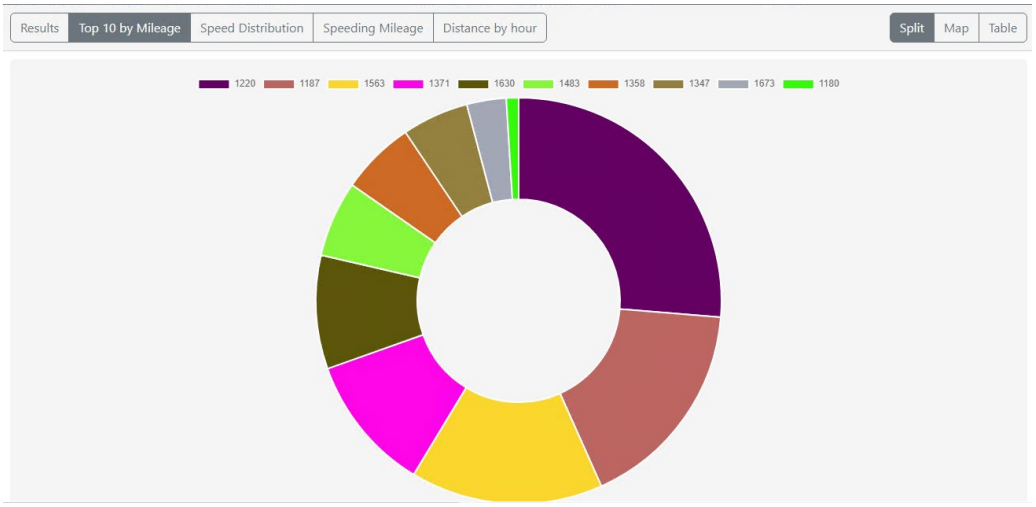
„*Data*“ dio JSON objekta sastoji se od „*labels*“ koje predstavljaju osnovnu podjelu grafikona po X osi, u ovom primjeru to su grupe brzina. „*Datasets*“ objekt predstavlja grupu zasebnih podataka koji se prikazuju. „*Label*“ unutar *datasets* objekta predstavlja naziv te grupe podataka, u primjeru grafikona distribucije brzine, prva grupa će imati naziv „*All Data*“, te ta grupa predstavlja sumu podataka iz svih ostalih grupa, dok će ostale grupe imati naziv ovisan o ID-ju vozila na kojega se podaci iz te grupe odnose. „*Data*“ unutar *datasets* objekta predstavlja skup podataka koji se prikazuje u grafikonu, za tu grupu, te su podaci poredani tako da odgovaraju količini i poretku skupa naziva *label* – definiranog u *data* objektu u korijenu JSON objekta. Odnosno, za svaki zapis unutar *label* skupa mora postojati odgovarajuća vrijednost unutar *data* skupa nekog *dataseta*.

„*All Data*“ *dataset* prikazan je linijskim grafom, te predstavlja ukupnu distribuciju svih puteva za sva vozila unutar dohvaćenih podataka, dok će ostali *dataset*-i biti prikazani stupčastim grafikonom, te predstavljaju distribuciju specifičnog vozila, te se kao takvi označavaju ID-jem vozila na kojeg se odnose

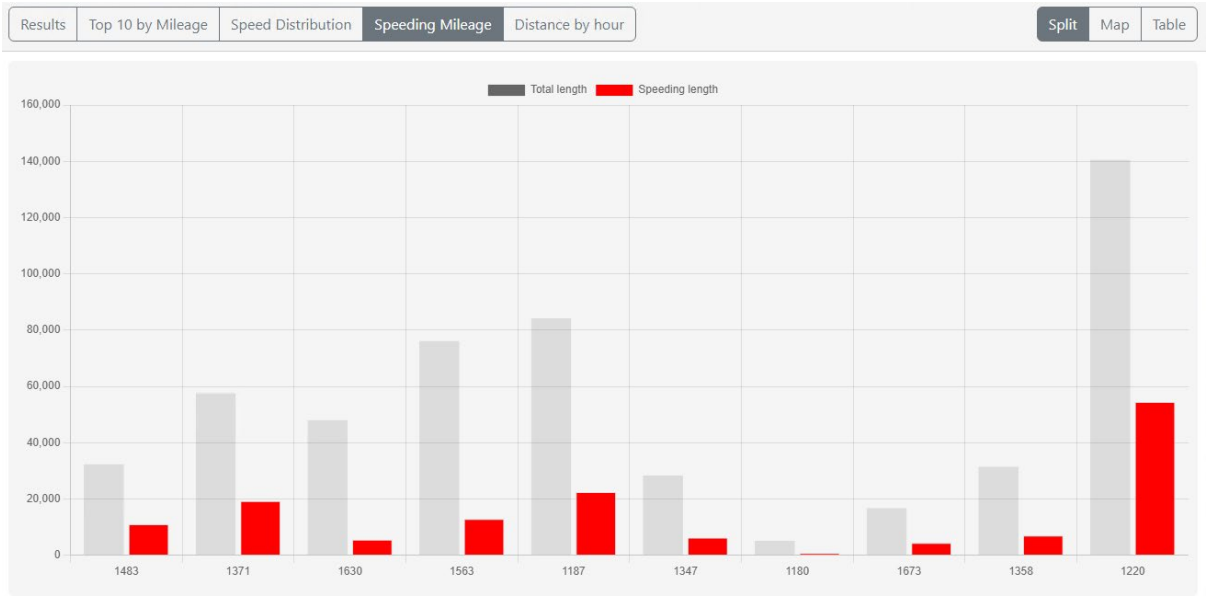
Primjer grafikona generiranog iz JSON objekta na slici 55, prikazan je slikom 56, te primjeri ostalih dostupnih grafikona prikazani su na slici 57, slici 58, te slici 59.



Slika 56: Grafikon distribucije brzina vozila u dohvaćenim putevima



Slika 57: Razdioba pređene udaljenosti za 10 vozila sa najvećom pređenom udaljenosti



Slika 58: Prikaz ukupno prijeđene udaljenosti po vozilu te udaljenosti prijeđene brzinom veće od ograničenja

Slika 59: Prikaz prijeđene udaljenosti vozila po satima u danu

5. Analiza rada implementiranih prostorno-vremenskih baza podataka

Analiza rada vršiti će se nad podatcima iz o putevima iz 2020. godine dohvaćenim iz Mireo Space-Time baze podataka. S obzirom da se radi o povijesnim podacima o kretanju vozila, odnosno o početnoj i krajnjoj točki putovanja, uspoređivati će se rad Space-Time baze podataka u usporedbi sa relacijskim bazama podataka SQL Server i PostgreSQL s implementiranim dodatkom za prostorne tipove podataka, PostGIS.

Svrha analize je uvid u razlike u brzini prostornih dohvata podataka navedenih baza podataka, te uvid u razlike zbog korištenih tehnologija indeksiranja.

5.1. Postavljanje okruženja za analizu

5.1.1 Dohvat podataka

Prvi korak postavljanja okruženja je preuzimanje podataka iz Mireo Space-Time baze podataka. S obzirom na opseg od cijele 2020. godine, količina i veličina podataka je prevelika za jedan dohvat. Stoga, napisan je Python program, prikazan koji preuzima podatke po dnevnoj granulaciji, od 1.1.2020. – 31.12.2020., te ih pohranjuje u zasebnu datoteku, u formatu u kojem su i preuzeti sa REST API sučelja Space-Time baze, kao što je ranije prikazano slikom 23. Rezultat navedenog je direktorij unutar kojega su datoteke koje sadrže JSON objekt odgovora koji sadrži podatke za jedan dan. Dohvaćeni podaci sadrže ukupno 5.049.298 zapisa.

```
import requests
import json
import time
import datetime
import os

Date = datetime.date(2020,1,1)
endDate = datetime.date(2020,12,31)
startTime = datetime.time(0,0)
endTime = datetime.time(23,59)
delta = datetime.timedelta(days=1)

while Date != endDate:
    startDateTime = datetime.datetime.combine(Date,startTime)
    endDateTime = datetime.datetime.combine(Date,endTime)
    epochBegin = int(startDateTime.timestamp())
    epochEnd = int(endDateTime.timestamp())

    queryStr = "SELECT * from st.trips where t[0] >= "+str(epochBegin)+"  AND t[1] <= "+str(epochEnd)

    URL = "http://192.168.202.222:1248/query"
    r = requests.post(url=URL, data=queryStr)
    ex=r.text

    output_dir = "data"
    if not os.path.exists(output_dir):
        os.mkdir(output_dir)

    output_filename = startDateTime.strftime("%Y-%m-%d") + "_output.txt"
    output_path = os.path.join(output_dir, output_filename)

    with open(output_path, "w") as output_file:
        output_file.write(ex)

    Date = Date + delta
```

Slika 60: Programski kôd za dohvat i spremanje podataka iz Mireo Space-Time baze

5.1.2 SQL Server

SQL Server sadrži funkcionalnosti za rad sa prostornim podacima, odnosno geometrijskim objektima. Kreiranje tablice koja odgovara strukturi tablice *st.trips*, prikazane tablicom 10, te odgovarajućih prostornih indeksa, prikazano je na slici 61. Tablica sadrži dva geometrijska tipa, točke, koji predstavljaju početak i kraj putovanja. Prostorni indeksi se kreiraju na geometrijskim tipovima, uz definiranje okvira (engl. „*bounding box*“), koji se definira sa minimalnim i maksimalnim vrijednostima geografske širine i dužine geometrijskih tipova – točaka. Prostorni indeksi u SQL Server implementaciji koriste B-stablo. Vrijeme kreiranja indeksa za popunjenu tablicu iznosi 27 sekundi.

```
CREATE TABLE trips (
    id INT IDENTITY(1,1) PRIMARY KEY,
    start_point geometry NOT NULL,
    end_point geometry NOT NULL,
    start_time DATETIME2(0) NOT NULL,
    end_time DATETIME2(0) NOT NULL,
    vid INT NOT NULL
);

CREATE SPATIAL INDEX idx_start_point ON trips (start_point) USING GEOMETRY_AUTO_GRID
WITH (BOUNDING_BOX = (-167.3155023157597, -52.042636724285735, 157.2107988595963, 70.05415778316986));

CREATE SPATIAL INDEX idx_end_point ON trips (end_point) USING GEOMETRY_AUTO_GRID
WITH (BOUNDING_BOX = (-71.00312724709511, -37.88937285653938, 113.2501897215843, 70.053757472639));
```

Slika 61: Kreiranje tablice *trips* u SQL Server bazi podataka

Unos podataka izvršen je kroz Python program, koji prolazi kroz sve ranije navedene generirane datoteke, pretvara podatke u format podoban za unos, te za svaki podatak kreira SQL INSERT naredbu koja se izvršava u bazi podataka. Programski kôd navedenog rješenja prikazan je slikom 62:

```python
import datetime
import json
import os
import pyodbc
from decimal import Decimal, getcontext
import math

def calcLongitude(x):
    getcontext().prec = 16
    return Decimal(Decimal(x) / Decimal(int("08000000", 16)) * Decimal(180.0))

def calcLatitude(y):
    return (2 * math.atan(math.exp((y / (int("08000000", 16)) * math.pi))) - math.pi / 2) * 180.0 / math.pi

server = 'localhost'
database = 'Geo'
username = 'username'
password = 'password'
cnxn = pyodbc.connect('DRIVER={SQL Server};SERVER='+server+';DATABASE='+database+';UID='+username+';PWD='+ password)

sql = "INSERT INTO trips (vid, start_point, end_point, start_time, end_time) VALUES (?, geometry::STPointFromText('POINT(' + CAST( ? AS nvarchar) + ' ' + CAST( ? AS nvarchar) + ')', 4326), geometry::STPointFromText('POINT(' + CAST( ? AS nvarchar) + ' ' + CAST( ? AS nvarchar) + ')', 4326), ?, ?)"

directory = '/data/'

for i, filename in enumerate(os.listdir(directory)):
    if filename.endswith(".txt"):
        with open(os.path.join(directory, filename), 'r') as f:
            json_str = f.read().strip('"')
            json_str = json_str.replace("\\", "")
            data = json.loads(json_str)
            for trip in data["tables"][0]["data"]:
                vid = trip[0]
                x0 = calcLongitude(trip[2])
                y0 = calcLatitude(trip[3])
                x1 = calcLongitude(trip[4])
                y1 = calcLatitude(trip[5])
                t0 = datetime.datetime.fromtimestamp(trip[1][0])
                t1 = datetime.datetime.fromtimestamp(trip[1][1])
                values = (vid, x0, y0, x1, y1, t0, t1)
                cursor = cnxn.cursor()
                cursor.execute(sql, values)
                cnxn.commit()

```

Slika 62: Programski kôd za unos podataka u SQL Server

5.1.3 PostgreSQL

PostgreSQL podršku za prostorne podatke ostvaruje kroz dodatak PostGIS, kojeg je potrebno zasebno instalirati. Implementacija PostGIS dodatka na bazu, te kreiranje tablice *trips* i odgovarajućih indeksa prikazano je slikom 63:

```sql
CREATE EXTENSION postgis

CREATE TABLE trips (
    id SERIAL PRIMARY KEY,
    start_point geometry(Point, 4326) NOT NULL,
    end_point geometry(Point, 4326) NOT NULL,
    start_time TIMESTAMP NOT NULL,
    end_time TIMESTAMP NOT NULL,
    vid INT NOT NULL
);

CREATE INDEX idx_trips_start_point ON trips USING GIST (start_point);

CREATE INDEX idx_trips_end_point ON trips USING GIST (end_point);
```

Slika 63: Kreiranje tablice *trips* u PostgreSQL bazi podataka

Napomena: PostGIS sadrži različite vrste indeksiranja za prostorne podatke, te su za analizu razmatrani GIST, i SP GIST, čije su razlike u performansama bile zanemarive, stoga je odabran GIST indeks, koji koristi R-stablo. Vrijeme kreiranja indeksa za popunjenu tablicu je 41 sekundu.

Unos podataka izvršen je kroz Python program, koji prolazi kroz sve ranije navedene generirane datoteke, pretvara podatke u format podoban za unos, te za svaki podatak kreira SQL INSERT naredbu koja se pohranjuje u novu datoteku. Nakon što je kreirana, datoteku je potrebno prebaciti na poslužitelj baze podataka, te ju je moguće unijeti. Programski kôd navedenog rješenja prikazan je slikom 64. Nakon što je datoteka kreirana i na poslužitelju, unos podataka iz iste moguće je pokrenuti naredbom

```
psql -d geo -f inserts.sql --username=postgres,
```

gdje „geo“ predstavlja ime baze podataka unutar koje se nalazi tablica za unos podataka.

```python
import datetime
import json
import os
from decimal import Decimal, getcontext
import math

def calcLongitude(x):
    getcontext().prec = 16
    return Decimal(Decimal(x) / Decimal(int("08000000", 16)) * Decimal(180.0))

def calcLatitude(y):
    return (2 * math.atan(math.exp((y / (int("08000000", 16)) * math.pi))) - math.pi / 2) * 180.0 / math.pi

output_file = 'inserts.sql'

table_name = 'trips'

with open(output_file, 'w') as f:
    directory = '/data/'
    for i, filename in enumerate(os.listdir(directory)):
        if filename.endswith(".txt"):
            with open(os.path.join(directory, filename), 'r') as fjson:
                f.write("BEGIN;\n")
                json_str = fjson.read().strip('"')
                json_str = json_str.replace("\\", "")
                data = json.loads(json_str)
                start_time = datetime.datetime.now()
                for trip in data["tables"][0]["data"]:
                    vid = trip[0]
                    x0 = calcLongitude(trip[2])
                    y0 = calcLatitude(trip[3])
                    x1 = calcLongitude(trip[4])
                    y1 = calcLatitude(trip[5])
                    t0 = datetime.datetime.fromtimestamp(trip[1][0])
                    t1 = datetime.datetime.fromtimestamp(trip[1][1])
                    start_point = f"ST_SetSRID(ST_MakePoint({x0}, {y0}), 4326)"
                    end_point = f"ST_SetSRID(ST_MakePoint({x1}, {y1}), 4326)"
                    values = (vid, start_point, end_point, t0, t1)
                    insert_statement = f"INSERT INTO {table_name} (vid, start_point, end_point, start_time, end_time) VALUES ({vid}, {start_point}, {end_point},
'{t0}', '{t1}');\n"
                    f.write(insert_statement)
                f.write("COMMIT;\n")
```

Slika 64: Programski kôd za kreiranje naredbi za unos podataka u PostgreSQL

5.2. Analiza brzine izvršavanja upita

Odabrani upiti biti će izvršeni na implementiranim bazama podataka, sa i bez korištenog prostornog indeksa, redom:

1. Ukupan broj zapisa,
2. Ukupan broj zapisa čija početna točka se nalazi na prostoru Zagreba,
3. Ukupan broj zapisa čija početna i krajnja točka se nalaze na prostoru Zagreba,
4. Dohvat ukupne sume duljina između početne i krajnje točke na prostoru Zagreba,
5. Dohvat 10 zapisa čija početna točka se nalazi najbliže Znanstveno-učilišnom kampusu Borongaj,
6. Dohvat površine poligona koji pokriva sve početne točke svih zapisa.

Rezultati analize prikazani su tablicom 11, te je iz istih vidljivo kako Space-Time baza podataka pruža najkonzistentnija vremena odgovora na upit, neovisno o kompleksnosti istog, dok vrijeme odgovora SQL Server baze podataka izrazito ovisi o korištenju indeksa, te ni u najboljem slučaju nije prihvatljive brzine. PostgreSQL vrijeme odaziva je u nekim slučajevima približno ili čak brže od Space-Time baze podataka, ali treba imati na umu da SQL Server i PostgreSQL sadrže samo podatke iz 2020. godine, dok Space-Time baza prolazi kroz znatno veći broj zapisa.

Tablica 11: Rezultati analize implementiranih baza podataka

Upit / Baza podataka	1	2	3	4	5	6
Space-Time	0,40 s	0,80 s	0,80 s	0,80 s	/	/
SQL (bez indeksa)	0,40 s	15,26 s	14,74 s	35,28 s	6,21 s	88,04 s
SQL (sa indeksom)	0,37 s	0,64 s	1,11 s	15,21 s	6,10 s	77,14 s
PostgreSQL (bez indeksa)	1,00 s	1,57 s	2,03 s	2,17 s	2,00 s	12,95 s
PostgreSQL (sa indeksom)	0,33 s	0,35 s	2,00 s	0,17 s	0,20 s	12,37 s

6. Zaključak

Prikupljanje podataka je samo po sebi uzaludan zadatak, ukoliko ne postoje mehanizmi i mogućnosti dugoročne isplative pohrane istih, funkcionalne i učinkovite mogućnosti obrade i analitike, te naposlijetku pretvorbe u korisne informacije.

Tradicionalni sustavi baza podataka, zbog visokih cijena pohrane, sadržavali su samo podatke koji predstavljaju trenutno stanje sustava – za što su relacijske baze podataka bile idealno primjenjive. Smanjenje cijene pohrane podataka dovelo je do znatnog skoka u količini prikupljanja i pohrane podataka. Takav trend, uz trend povećanja količine pohrane koju podaci zauzimaju, te broja različitih podataka koje je moguće prikupiti, rezultirao je potrebom za novim rješenjima, koja omogućavaju skaliranje sustava ovisno o potrebama za procesorskim i prostornim kapacitetom, te čija je osnovna namjena usmjerena na velike skupove podataka.

Prikupljanje prostornih podataka s vremenom postaje sve popularnije – slučajevi uporabe istih se šire iz dana u dan, te uključuju primjene kao što su optimizacije prometnica, logističkih procesa, automatske naplate cestarina, osiguranja temeljena na korištenju, marketinške svrhe, društvene mreže, navigacija ovisna o stvarno-vremenskim uvjetima, i mnogi drugi. Za sve navedene primjene potrebno je prikupljati, pohranjivati te obrađivati izrazito veliku količinu podataka s različitih izvora, te uz to podržavati korištenje prostornih i vremenskih koncepata i funkcija. Iako je područje sustava baza podataka izrazito raznoliko, te postoji jako veliki broj sustava što generičkih što specifičnih namjena, područje prostorno-vremenskih baza podataka, specifično baza podataka pokretnih objekata, je i dalje područje sa relativno malim brojem komercijalnih rješenja.

Kroz izradu sučelja za dohvat i prikaz prostornih podataka na karti, prikazan je cijeli proces izrade i implementacije rješenja otvorenog kôda, u svrhu filtriranja velike količine prostornih podataka, te prezentacije i prikaza istih. Glavna svojstva kreiranog sučelja su brzina rada i brzina razvoja, te odvojenost slojeva korisničkog sučelja od programske logike i baze podataka – što znatno povećava održivost i nadogradivost ovog rješenja. Kreirano sučelje predstavlja odličnu početnu točku prilikom kreiranja kompletnog sustava za analizu kretanja flote vozila, te pregled povijesti kretanja istih. Kroz proces kreiranja rješenja iskazala se važnost projekata otvorenog kôda, te važnost kvalitetno napisane i opsežne dokumentacije – što je bilo jedan od odlučujućih faktora prilikom odabira biblioteka i alata.

Rezultat analize prostornih dohvata za različite inačice sustava baza podataka naglašava važnost dobre procjene i definiranja slučajeva upotrebe, te opsega i veličine korištenih

podataka. Za određene slučajeve uporabe, npr. pohranu i dohvat povijesnih podataka o kretanju vozila, ukoliko nisu potrebni kompleksni upiti za veće količine podataka, relacijske baze podataka, uz korištenje dostupnih dodataka te odabir optimalne vrste indeksa, pružaju zadovoljavajuću brzinu prilikom dohvata podataka. Za slučaj uporabe koji zahtijeva veliku razinu dostupnosti, te mogućnosti pohrane nekoliko tisuća nestrukturiranih podataka u sekundi, stabilnije i konzistentnije se pokazuje baza podataka NoSQL ili NewSQL tipa. Mogućnosti modernih sustava baza podataka, kao što su mogućnost dodavanja više neovisnih čvorova, te mogućnost dislokacije sustava na Cloud infrastrukture, prenose sustave baza podataka u novu eru – eru povećane dostupnosti, performansi, jednostavnosti i brzine razvoja i održavanja, te cijene ovisne o razini korištenja sustava, a ne o maksimalnoj projektiranoj utilizaciji prilikom planiranja sustava.

Bibliografija

[1] S. E. Dr. S. Sumathi, Fundamentals of Relational Database Management Systems, Berlin; London : Springer-Verlag Berlin Heidelberg , 2007.

[2] S. S. L. T. N. H. J. Toby J. Teorey, Database Modeling and Design: Logical Design, Morgan Kaufmann , 2005.

[3] S. B. N. Ramez Elmasri, Fundamentals of Database Systems, Pearson Education, 2003.

[4] »Computer Notes - A Complete Guide,« [Mrežno]. https://ecomputernotes.com/fundamental/what-is-a-database/type-of-data-models#Record_Based_Logical_Models. [Pokušaj pristupa 25 10 2022].

[5] M. J. Hernandez, »Database Design for Mere Mortals,« Addison-Wesley, 2008.

[6] E. F. Codd, »The relational model for database management: version 2,« Addison-Wesley, 1990.

[7] C. Date, Database in Depth - Relational Theory for Practitioners, Sebastopol: O'Reilly Media, Inc., 2005.

[8] J. G. Raghu Ramakrishnan, Database Management Systems, 2003.

[9] H. D. N. L. C.J. Date, Temporal Data & the Relational Model, Morgan Kaufmann, 2002.

[10] J. D. U. J. W. Hector Garcia-Molina, Database Systems: The Complete Book, Pearson Education, 2013.

[11] M. Chand, »C# Corner,« 20 3 2023. [Mrežno]. https://www.c-sharpcorner.com/article/what-are-the-most-popular-relational-databases/. [Pokušaj pristupa 19 04 2023].

[12] H. D. C. J. Date, A Guide to SQL Standard, Addison-Wesley Professional, 1996.

[13] B. Kelechava, »American National Standards Institute,« 5 10 2018. [Mrežno]. https://blog.ansi.org/2018/10/sql-standard-iso-iec-9075-2016-ansi-x3-135/. [Pokušaj pristupa 26 2 2023].

[14] The Open Group, Data Management: Structured Query Language (SQL), Version 2, Open Company Ltd., 1996.

[15] ISO/IEC 9075-1 - Information technology - Database languages - SQL, International Organization for Standardization, 2016.

[16] »What is OLTP,« Oracle, [Mrežno]. https://www.oracle.com/database/what-is-oltp/. [Pokušaj pristupa 14 06 2023].

[17] »Databricks ACID Transactions,« [Mrežno]. https://www.databricks.com/glossary/acid-transactions. [Pokušaj pristupa 17 05 2023].

[18] »AWS - What Is OLAP,« Amazon, [Mrežno]. https://aws.amazon.com/what-is/olap/. [Pokušaj pristupa 18 06 2023].

[19] B. Marr, »Forbes - What Is A Data Lakehouse?,« Forbes, 18 01 2022. [Mrežno]. https://www.forbes.com/sites/bernardmarr/2022/01/18/what-is-a-data-lakehouse-a-super-simple-explanation-for-anyone/. [Pokušaj pristupa 18 06 2023].

[20] »MongoDB - What is NoSQL,« MongoDB, [Mrežno]. https://www.mongodb.com/nosql-explained. [Pokušaj pristupa 18 06 20203].

[21] »What is NoSQL?,« Amazon, [Mrežno]. https://aws.amazon.com/nosql/. [Pokušaj pristupa 20 06 2023].

[22] »How to Scale MongoDB,« MongoDB, [Mrežno]. https://www.mongodb.com/basics/scaling. [Pokušaj pristupa 20 06 2023].

[23] S. S. Nazrul, »Medium - CAP Theorem and Distributed Database Management Systems,« 24 04 2018. [Mrežno]. https://towardsdatascience.com/cap-theorem-and-distributed-database-management-systems-5c2be977950e. [Pokušaj pristupa 18 06 2023].

[24] S. Kumar, »SQL vs NoSQL vs NewSQL: An In-depth Literature Review,« Sumit's Diary, 09 06 2022. [Mrežno]. https://blog.reachsumit.com/posts/2022/06/sql-nosql-newsql/. [Pokušaj pristupa 20 06 2023].

[25] »ACID vs. BASE: Comparison of Database Transaction Models,« pheonixNAP, 25 11 2020. [Mrežno]. https://phoenixnap.com/kb/acid-vs-base. [Pokušaj pristupa 19 06 2023].

[26] L. Hu, »NewSQL, Lakehouse, HTAP, and the Future of Data,« Medium, 3 8 2022. [Mrežno]. https://towardsdatascience.com/newsql-lakehouse-htap-and-the-future-of-data-69d427c533e0. [Pokušaj pristupa 19 06 2023].

[27] R. T. Snodgrass, Developing Time-Oriented Database Applications In Sql, Morgan-Kaufmann, 2000.

[28] M. S. Ralf Hartmut Güting, Moving Objects Databases, Morgan Kaufmann , 2005.

[29] R. W. Tom Johnston, Managing Time in Relational Databases: How to Design, Update and Query Temporal Data, Morgan Kaufmann, 2010.

[30] H. K. Richard T. Snodgrass, The TSQL2 Temporal Query Language, Springer US, 1995.

[31] »Open Geospatial Consortium Inc.,« OSGeo, [Mrežno]. Available: https://www.osgeo.org/partners/ogc/. [Pokušaj pristupa 20 4 2023].

[32] G. B. H. Albert K.W. Yeung, Spatial Database Systems: Design, Implementation and Project Management, Springer, 2007.

[33] Open Geospatial Consortium Inc., »OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 1:Common architecture,« Open Geospatial Consortium Inc., 2005.

[34] C. S. J. M. O. S. Michael H. Bohlen, Spatio-Temporal Database Management, Edinburgh: Springer, 1999.

[35] »SECONDO - An Extensible Database System,« SECONDO, 06 08 2009. [Mrežno]. https://secondo-database.github.io/. [Pokušaj pristupa 20 06 2023].

[36] Mireo d.d., »Space-Time,« Mireo d.d., [Mrežno]. https://www.mireo.com/spacetime.html. [Pokušaj pristupa 20 06 2023].

[37] Mireo d.d., »The definitive guide through an unusual and fundamentally new approach of using spatiotemporal data in our everyday routine,« Mireo d.d., [Mrežno]. https://www.mireo.com/spacetime-the-guide. [Pokušaj pristupa 20 06 2023].

[38] PostgreSQL, »PostgreSQL: The World's Most Advanced Open Source Relational Database,« [Mrežno]. https://www.postgresql.org/. [Pokušaj pristupa 20 06 2023].

[39] PostGIS, »PostGIS Manual - Chapter 4. Data Management,« [Mrežno]. http://postgis.net/docs/manual-3.3/using_postgis_dbmanagement.html#build-indexes. [Pokušaj pristupa 20 06 2023].

[40] MobilityDB, »GitHub - MobilityDB,« [Mrežno]. https://github.com/MobilityDB/MobilityDB. [Pokušaj pristupa 20 06 2023].

[41] GeoMesa, »Store, index, query, and transform spatio-temporal data at scale in HBase, Accumulo, Cassandra, Redis, Kafka and Spark,« GeoMesa, [Mrežno]. https://www.geomesa.org/. [Pokušaj pristupa 20 06 2023].

[42] GeoMesa, »GeoMesa Documentation,« GeoMesa, [Mrežno]. https://www.geomesa.org/documentation/stable/user/introduction.html#what-is-geomesa. [Pokušaj pristupa 20 06 2023].

[43] Mireo d.d., »SpaceTime - Developer's guide«.

[44] »Red Hat,« 8 5 2020. [Mrežno]. https://www.redhat.com/en/topics/api/what-is-a-rest-api. [Pokušaj pristupa 22 3 2023].

[45] Oracle, »REST API for Oracle Enterprise Performance Management Cloud,« Oracle, [Mrežno]. https://docs.oracle.com/en/cloud/saas/enterprise-performance-management-common/prest/rest_api_methods.html. [Pokušaj pristupa 3 22 2023].

[46] K. L. Rick Anderson, »Tutorial: Create a web API with ASP.NET Core,« Microsoft, 27 03 2023. [Mrežno]. https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-7.0&tabs=visual-studio. [Pokušaj pristupa 02 04 2023].

[47] A. Moussawi, »SQLKata the dotnet SQL query builder,« [Mrežno]. https://sqlkata.com/. [Pokušaj pristupa 02 04 2023].

[48] Internet Engineering Task Force, The GeoJSON Format - RFC7946, 2016.

[49] »GeoJSON,« [Mrežno]. https://geojson.org/. [Pokušaj pristupa 02 04 2023].

[50] V. Agafonkin, »Leaflet,« [Mrežno]. https://leafletjs.com/. [Pokušaj pristupa 10 04 2023].

[51] »Hrvatski jezični portal,« [Mrežno]. https://hjp.znanje.hr/index.php?show=search_by_id&id=d1pjWRU%3D&keyword=subverzija. [Pokušaj pristupa 8 November 2022].

[52] A. I. Din, Structured Query Language (SQL), NCC Blackwell, 1994.

[53] W3C, »W3C Recommendation,« [Mrežno]. https://www.w3.org/TR/REC-xml/. [Pokušaj pristupa 13 Veljača 2023].

[54] »Epoch101,« [Mrežno]. https://www.epoch101.com/What-is-Unix-Time. [Pokušaj pristupa 22 05 2023].

[55] »GISGeography,« 28 05 2022. [Mrežno]. https://gisgeography.com/wgs84-world-geodetic-system/. [Pokušaj pristupa 28 05 2023].

[56] »MDN Web Docs,« Mozilla, 21 02 2023. [Mrežno]. https://developer.mozilla.org/en-US/docs/Glossary/MVC. [Pokušaj pristupa 25 05 2023].

[57] »OCI - What is Big Data?,« Oracle, [Mrežno]. https://www.oracle.com/big-data/what-is-big-data/. [Pokušaj pristupa 18 06 2023].

Popis slika

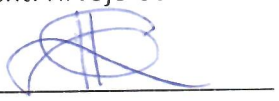
Popis tablica

Sveučilište u Zagrebu
Fakultet prometnih znanosti
Vukelićeva 4, 10000 Zagreb

IZJAVA O AKADEMSKOJ ČESTITOSTI I SUGLASNOSTI

Izjavljujem i svojim potpisom potvrđujem da je diplomski rad isključivo rezultat mojega vlastitog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu, a što pokazuju upotrijebljene bilješke i bibliografija. Izjavljujem da nijedan dio rada nije napisan na nedopušten način, odnosno da je prepisan iz necitiranog rada te da nijedan dio rada ne krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.
Svojim potpisom potvrđujem i dajem suglasnost za javnu objavu završnog/diplomskog rada pod naslovom RAZVOJ USLUGE INTERAKTIVNE KARTE ZA PRIKAZ PODATAKA O KRETANJU VOZILA PROMETNOM MREŽOM, u Nacionalni repozitorij završnih i diplomskih radova ZIR.

Student: Hrvoje Goldner



U Zagrebu, 26.6.2023.