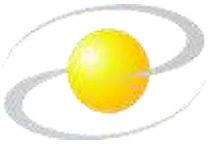
UNIVERZITET CRNE GORE
ELEKTROTEHNIČKI FAKULTET



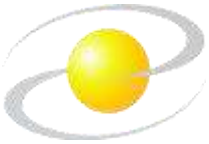
Ognjen Filipović

Razvoj alata za monitoring saobraćaja primjenom Apache Spark i Apache Kafka alata

MASTER RAD

Podgorica, 2023. godine

UNIVERZITET CRNE GORE
ELEKTROTEHNIČKI FAKULTET



Ognjen Filipović

Razvoj alata za monitoring saobraćaja primjenom Apache Spark i Apache Kafka alata

MASTER RAD

Podgorica, 2023. godine

PODACI I INFORMACIJE O STUDENTU

Ime i prezime:	Ognjen Filipović
Datum i mjesto rođenja:	25.04.1999. godine, Mojkovac, Crna Gora
Naziv završenog osnovnog studijskog programa i godina završetka studija:	Studijski program Primijenjeno računarstvo, Elektrotehnički fakultet, Univerzitet Crne Gore, 240 ECTS kredita, 2021. godine.

INFORMACIJE O MASTER RADU

Naziv master studija:	Master studije primijenjenog računarstva
Naslov rada:	Razvoj alata za monitoring saobraćaja primjenom Apache Spark i Apache Kafka alata
Fakultet na kojem je rad odbranjen:	Elektrotehnički fakultet

UDK, OCJENA I ODBRANA MASTER RADA

Datum prijave master rada:	02.01.2023. godine
Datum sjednice Vijeća na kojoj je prihvaćena tema:	08.05.2023. godine
Komisija za ocjenu/odbranu rada:	Prof. dr Milutin Radonjić Prof. dr Nikola Žarić Prof. dr Zoran Veljović
Mentor:	Prof. dr Nikola Žarić
Komisija za ocjenu rada:	Prof. dr Milutin Radonjić Prof. dr Nikola Žarić Prof. dr Zoran Veljović
Komisija za odbranu rada:	Prof. dr Milutin Radonjić Prof. dr Nikola Žarić Prof. dr Zoran Veljović
Datum odbrane:	19.09.2023. godine

PREDGOVOR

Porodica je temelj svakog društva, sigurna luka i neiscrpan izvor ljubavi, pažnje, podrške i inspiracije zato ovaj rad posvećujem njima… Svojim roditeljima, majci Stanki i ocu Živku. Vi ste moji prvi i najbolji učitelji. Mojoj braći Ivici i Stefanu. Hvala im što su vjerovali u mene i što su me uvijek motivisali da idem dalje u ostvarenju svojih snova.

Posebnu zahvalnost dugujem svome mentoru, prof. dr Nikoli Žariću. Njegovi stručni savjeti, vrijeme koje je odvojio da mi pomogne, vođstvo i podrška bili su neizmjerni tokom cijelog procesa istraživanja i pisanja. Nesebičnost, stručnost i predanost značajno su doprinijeli kvalitetu ovog rada.

Na kraju, želim se zahvaliti svim mojim prijateljima, kolegama i meni dragim osobama koji su me podržavali u svakom koraku na ovom životnom putu koji prolazim. Vaše riječi ohrabrenja, razgovori i smijeh doprinijeli su mojoj inspiraciji i motivaciji. Neizmjerno vam hvala svima od srca.

Ognjen Filipović, BApp

IZVOD RADA

Razvijeno rješenje ima za cilj pružiti korisnicima alat za nadgledanje saobraćaja koji kombinuje tehnologije Kafka i Spark Streaming. Sistem omogućava efikasno prikupljanje podataka, obradu u realnom vremenu, predviđanje saobraćajnih tokova i detekciju promjena, što pomaže vozačima, upravljačima saobraćajnih sistema i drugim relevantnim akterima da donose bolje odluke i unaprijede efikasnost i sigurnost saobraćaja.

Razvija se rješenje za nadgledanje saobraćaja koje koristi kombinaciju Kafka i Spark Streaming tehnologija. Ovo rješenje ima sljedeće funkcionalnosti:

1. Prikupljanje podataka o saobraćaju: Rješenje koristi senzore postavljene duž puteva kako bi prikupilo podatke o brzini vozila, gustini saobraćaja, broju vozila na putu i drugim relevantnim informacijama o saobraćaju.

2. Streaming obrada podataka: Podaci sa senzora se šalju na Kafka topic koji služi kao ulazna tačka sistema. Spark Streaming tehnologija se koristi za obradu podataka u realnom vremenu. Ovo omogućava brzu i efikasnu analizu velikog broja podataka kako bi se izvukle korisne informacije o saobraćaju.

3. Predviđanje saobraćajnih tokova: Rješenje ima sposobnost predviđanja saobraćajnih tokova na osnovu prethodnih podataka. Na primjer, može predvidjeti moguće gužve na određenim putevima u određeno doba dana ili tokom posebnih događaja.

4. Detekcija promjena: Rješenje takođe omogućava detekciju promjena u saobraćaju. Analiza podataka i identifikacija nepravilnosti kao što su nagle promjene, povećanje broja vozila ili neočekivane promjene u saobraćaju. Ova funkcionalnost omogućava brzo reagovanje na potencijalne probleme ili nesreće na putu.

5. Vizualizacija podataka: Rješenje pruža mogućnost vizualizacije prikupljenih i obradjenih podataka o saobraćaju putem grafova, tabela ili interaktivnih mapa. Ovo omogućava korisnicima da jasno vide stanje saobraćaja i donose odluke na bazi relevantnih informacija.

.

ABSTRACT

The developed solution aims to provide users with a comprehensive traffic monitoring tool that combines Kafka and Spark Streaming technologies. The system enables efficient data collection, real-time processing, traffic flow prediction, and change detection, benefiting drivers, traffic management authorities, and other relevant stakeholders in making better decisions and enhancing traffic efficiency and safety.

A traffic monitoring solution is being developed that utilizes a combination of Kafka and Spark Streaming technologies. This solution offers the following functionalities:

1. Traffic data collection: The solution utilizes sensors placed along the roads to gather data on vehicle speed, traffic density, the number of vehicles on the road, and other relevant traffic information.

2. Streaming data processing: Sensor data is sent to a Kafka topic, which serves as the entry point of the system. Spark Streaming technology is employed to process the data in real-time, enabling fast and efficient analysis of large volumes of traffic data to extract valuable insights.

3. Traffic flow prediction: The solution is capable of predicting traffic flows based on historical data. For example, it can anticipate potential congestion on specific roads during certain times of the day or special events.

4. Change detection: The solution also facilitates the detection of traffic changes. By analyzing data and identifying anomalies such as sudden fluctuations, increased vehicle counts, or unexpected traffic patterns, it enables swift responses to potential issues or accidents on the road.

5. Data visualization: The solution offers data visualization capabilities, presenting the collected and processed traffic data through graphs, tables, or interactive maps. This allows users to have a clear overview of the traffic conditions and make informed decisions.

Ime i prezime autora: Ognjen Filipović, BApp

ETIČKA IZJAVA

U skladu sa članom 22 Zakona o akademskom integritetu i članom 18 Pravila studiranja na master studijama, pod krivičnom i materijalnom odgovornošću, izjavljujem da je magistarski rad pod naslovom

" Razvoj alata za monitoring saobraćaja primjenom Apache Spark i Apache Kafka alata"

moje originalno djelo.

Podnosilac izjave,

Ognjen Filipović, Bapp

Ognjen Filipović

U Podgorici, dana 30.06.2023. godine

Sadržaj

1. Uvod

U današnjem digitalnom dobu, sve veći broj organizacija i gradova se oslanja na tehnologiju kako bi prikupili, obradili i vizualizovali podatke o saobraćaju radi donošenja odluka na bazi relevantnih informacija. Integracija Apache Spark i Apache Kafka alata otvara mogućnosti za efikasno prikupljanje, obradu i vizualizaciju ovih podataka kroz jedinstveni korisnički interfejs. U cilju istraživanja ove teme, postavljena su četiri ključna istraživačka pitanja.

Prvo pitanje istražuje mogućnost integrisanja Apache Spark i Apache Kafka alata kako bi se omogućilo prikupljanje i obrada podataka o saobraćaju na način koji olakšava vizualizaciju kroz korisnički interfejs. Ova integracija ima potencijal da pruži kontinuirani tok podataka iz različitih izvora. Za postizanje ovog cilja, neophodno je integrisati ove alate sa drugim alatima za vizualizaciju i prikaz podataka.

Drugo istraživačko pitanje se fokusira na ograničenja i izazove povezane sa integracijom podataka o saobraćaju u realnom vremenu u web platformu za praćenje saobraćaja i podršku odlučivanju. Tehnički, organizacioni i sigurnosni izazovi mogu uticati na efikasnost prikupljanja, obrade i analize podataka. Razumijevanje ovih izazova je ključno za uspješno planiranje i implementaciju projekta.

Treće istraživačko pitanje se bavi najboljim načinom smiještanja prikupljenih podataka o saobraćaju u jedinstvenu bazu podataka radi olakšanja njihove obrade. Ovakav pristup omogućava bržu obradu, poboljšava kvalitet i tačnost podataka, te olakšava njihovo korišćenje. Kroz odgovarajući dizajn baze podataka, postiže se optimalno skladištenje i upravljanje podacima.

Četvrto istraživačko pitanje se fokusira na razvoj korisničkog interfejsa koji omogućava donosiocima odluka jednostavan i intuitivan pristup vizualizaciji podataka o saobraćaju kroz različite parametre kao što su vrijeme, mjesto, gustina saobraćaja i druge relevantne metrike. Takav interfejs olakšava pristup relevantnim informacijama i omogućava bolje planiranje i upravljanje održivom mobilnošću.

Kroz detaljno istraživanje ova četiri istraživačka pitanja, cilj je stvoriti sveobuhvatan uvid u mogućnosti integracije Apache Spark i Apache Kafka alata za prikupljanje, obradu i vizualizaciju podataka o saobraćaju. Ovaj istraživački rad ima za cilj razumijevanje tehničkih

aspekata, identifikaciju izazova i ograničenja, kao i pronalaženje najboljih praksi za uspješnu implementaciju integrisanog sistema.

1.1. Razlozi, motivi i glavni ciljevi u procesu istraživanja

Postoje ključni razlozi zbog kojih je istraživanje razvoja alata za monitoring saobraćaja primjenom Apache Spark i Apache Kafka alata od značaja:

1. Efikasnije planiranje urbane mobilnosti: Uz pomoć Apache Spark i Apache Kafka alata, moguće je prikupljati veliku količinu podataka o saobraćaju u realnom vremenu i analizirati ih kako bi se dobili korisni uvidi o stanju saobraćaja. Ovi podaci mogu se koristiti za poboljšanje planiranja i organizacije održive mobilnosti, što bi moglo dovesti do efikasnijeg i održivijeg saobraćaja u urbanim sredinama.
2. Poboljšanje sigurnosti u saobraćaju: Analiza podataka o saobraćaju omogućava identifikaciju mjesta s visokim rizikom od nesreća i gužvi, a planiranje i organizacija saobraćaja na ovim mjestima može poboljšati sigurnost u saobraćaju. Pored toga, optimizacija rute javnog prevoza i upotreba pametnih semafora mogla bi smanjiti gužve i smanjiti rizik od nesreća.
3. Povećanje kvaliteta života u gradovima: Poboljšanje saobraćajne efikasnosti i sigurnosti u urbanim sredinama, kao i optimizacija urbanog prevoza, mogli bi značajno poboljšati kvalitet života stanovnika u gradovima.
4. Kombinacija alata: Ovaj rad ima za cilj pokazati kako se ovi alati mogu zajedno kombinovati, te kako se rezultati te kombinacije mogu koristiti za podršku odlučivanju u planiranju i poboljšanju održive mobilnosti.

Svi ovi razlozi čine istraživanje razvoja alata za monitoring saobraćaja primjenom Apache Spark i Apache Kafka alata relevantnim i značajnim za istraživanje.

Motiv ovog istraživanja je razviti alat za monitoring saobraćaja primjenom velikog broja različitih alata.

Postoje nekoliko motiva koji su razlog razvijanja web platforme za monitoring saobraćaja primjenom Apache Spark i Apache Kafka alata:

1. Povećanje efikasnosti i održivosti saobraćaja: Ubrzan i održiv transport ključan je za razvoj urbanih sredina i poboljšanje kvaliteta života stanovnika. Uz pomoć web platforme moguće je izvući korisne informacije o gužvama i kašnjenjima, optimizovati semafore i rute javnog prevoza, smanjiti zagađenje i troškove

transporta, te poboljšati efikasnost i održivost saobraćaja u urbanim sredinama.

2. Potreba za naprednim alatima za obradu podataka: U današnjem svijetu velikih podataka, korišćenjem Apache Spark i Apache Kafka alata, moguće je efikasno prikupljati, analizirati i obraditi velike količine podataka u realnom vremenu, što je ključno za podršku odlučivanju u planiranju održive mobilnosti.
3. Potencijalne primjene u gradovima: Razvijeni alat može biti primijenjen u gradovima kako bi se poboljšala efikasnost i održivost saobraćaja, smanjili troškovi, te poboljšao kvalitet života stanovnika. Ovaj alat ima potencijal da donese značajne koristi gradovima, a istraživanje se fokusira na simulaciji primjene u stvarnim urbanim sredinama.

Ciljevi istraživanja su sljedeći:

1. Prikupljanje podataka o saobraćaju iz različitih izvora, uključujući senzore, GPS uređaje i druge dostupne izvore podataka.
2. Obrada, kombinovanje i analiza prikupljenih podataka korišćenjem odgovarajućih alata i integracija velikih podataka, kako bi se dobili relevantni rezultati i cjelovita slika o stanju saobraćaja.
3. Pružanje bolje vizualizacije podataka o saobraćaju kroz karte, grafikone, tabele i druge vizualne prikaze koji omogućavaju korisnicima da lakše procesuiraju informacije.

Svi ovi ciljevi su usmjereni na razvoj i primjenu alata za monitoring saobraćaja koji koristi Apache Spark i Apache Kafka alate.

1.2. Predmet istraživanja

Glavni predmet istraživanja je pokazati kako se ove tehnologije mogu primijeniti za prikupljanje, analizu i obradu velike količine podataka o saobraćaju u urbanim sredinama, te kako se ove informacije mogu koristiti za podršku odlučivanju u planiranju i poboljšanju održive mobilnosti. U cilju realizacije istraživanja predmet ovog rada je razvoj web platforme za monitoring saobraćaja primjenom različitih alata poput Apache Spark, Apache Kafka, ZooKeeper, Cassandra i alata za prikaz vizualizacije pomoću karte, grafikona i tabela.

Konkretno, istraživanje će se usresrediti na sljedeće teme:

- Analiza i obrada podataka o saobraćaju pomoću Apache Spark i Apache Kafka alata;
- Korišćenje ZooKeepera za upravljanje koonfiguracijom i kordinacijom servisa;
- Upotreba Cassandre za skladištenje velikih količina podataka;
- Generisanje korisnih informacija o stanju saobraćaja, poput gužvi i njihova vizualizacija, koje se mogu koristiti za podršku odlučivanju u planiranju i poboljšanju održive mobilnosti;
- Korisnički interfejs na jednostavan način bi omogućavao korišćenje i pružao jasne informacije korisnicima o saobraćaju.

Kroz ovo istraživanje, treba da se pokaže kako primjena naprednih tehnologija poput Apache Spark, Apache Kafka, ZooKeeper i Cassandra može pomoći u poboljšanju efikasnosti i održivosti saobraćaja u urbanim sredinama.

Kako bi se adekvatno sagledala opravdanost predmetnog istraživanja, neophodno je sagledati postigunuća prethodnih istraživanja. Istraživanja [1-10] proučavaju različite teme vezane za praćenje, upravljanje i analizu saobraćaja koristeći moderne tehnologije kao što su Internet inteligentnih uređaja (Internet of Things), računarstvo u oblaku (Cloud computing) i velike podatke. Konkretno, članci pokrivaju:

Istraživanje [1] predstavlja sistem za praćenje saobraćaja u realnom vremenu koji koristi

Apache Spark, koji obrađuje podatke o saobraćaju prikupljene od senzora i kamera i pruža uvid u tok saobraćaja, zagušenja i nesreće. Sistem koristi algoritme mašinskog učenja za predviđanje saobraćaja i otkrivanje anomalija. Cilj rada je da se pomogne u rješavanju problema preopterećenog saobraćaja u velikim gradovima i da se podrži planiranje urbane mobilnosti. U radu se takođe opisuju eksperimenti koji su izvedeni da bi se evaluirala performansa sistema. Rezultati pokazuju da je sistem u stanju da obrađuje veliku količinu podataka u realnom vremenu i da može generisati korisne informacije o saobraćaju koje se mogu koristiti za podršku odlučivanju.

Rad [2] govori o analizi gradskog saobraćajnog toka obradom podataka dobijenih od senzora. Koristili su tehnike mašinskog učenja za modeliranje saobraćajnih obrazaca i predložili okvir za predviđanje saobraćajnih zagušenja. Do 2050. skoro 70% stanovništva živjeće u gradovima. Kako stanovništvo raste, potražnja za putovanjima se povećava i to može uticati na kvalitet vazduha u urbanim područjima. Ovaj rad opisuje implementaciju modela gradskog saobraćajnog toka u gradu Modena na osnovu stvarnih podataka senzora saobraćaja. Ovo je dio evropskog projekta koji ima za cilj proučavanje korelacije između saobraćaja i zagađenja vazduha, dakle kombinovanje simulacija saobraćaja i zagađenja vazduha za testiranje različitih urbanih scenarija i podizanje svijesti građana o kvalitetu vazduha tamo gdje je to potrebno.

U istraživanju [3] se predlaže sistem za analizu saobraćaja vozila u realnom vremenu koji koristi mreže kratkoročne memorije (LSTM) u Apache Spark-u. Sistem analizira podatke o saobraćaju prikupljene od senzora i kamera i predviđa protok saobraćaja, zagušenja i nesreće. U pogledu ove situacije, praćenje i analitika saobraćaja postaje potreba svakog sata. Analiza saobraćaja u realnom vremenu zahtijeva obradu tokova podataka koji se kontinuirano generišu kako bi stekli brzi uvidi. Da biste brže obradili strim podatke, potrebne su nam tehnologije sa visokim računarskim kapacitetom. Okviri velikih podataka kao što su Apache Hadoop, Spark i Kafka, sa svojom sposobnošću obrade ogromne količine podataka, omogućili su razvoj naprednih i efikasnih sistema za obradu tokova podataka. Izazov analize tokova podataka za predviđanje u realnom vremenu može se prevazići korišćenjem tehnika dubokog učenja.

Kul et al. i ostali [4] predlažu sistem za detekciju vozila u realnom vremenu koji koristi mikroservise zasnovane na događajima sa Apache Kafka streamova. Sistem analizira podatke o saobraćaju na osnovu tipa, boje i atributa brzine i detektuje i klasifikuje vozila u realnom vremenu.

U radu [5] je predstavljena web platforma za upravljanje informacijama o premjeru i održavanju puteva, koja može pomoći u razvoju pametnih sistema upravljanja putevima. Platforma prikuplja i analizira podatke o putevima i pruža uvid u stanje na putevima i zahteve održavanja.

Anusha et al. i ostali [6] predlažu sistem za praćenje saobraćaja zasnovan na IoT-u koji daje prioritet vozilima za hitne slučajeve. Sistem koristi senzore i kamere za otkrivanje i praćenje vozila hitne pomoći i pruža im jasan put do odredišta.

U radu [7] je predložena platforma za upravljanje urbanim podacima, integraciju i obradu. Platforma prikuplja podatke iz različitih izvora i pruža jedinstven pogled na urbane podatke, koji se mogu koristiti za poboljšanje urbanog planiranja i upravljanja.

U članku [8] se govori o donošenju odluka za predloženo rešenje za upravljanje saobraćajem korišćenjem IoT tehnologije. Autori predlažu sistem koji koristi senzore i kamere za prikupljanje podataka o saobraćaju i algoritme mašinskog učenja za analizu podataka i pružanje uvida u obrasce saobraćaja i zagušenja.

U istraživanju [9] se predlaže pametna kontrolna tabla zasnovana na IoT-u za pametne gradove. Kontrolna tabla prikuplja podatke iz različitih izvora, uključujući senzore i kamere, i pruža uvid u realnom vremenu u protok saobraćaja, kvalitet vazduha i druge urbane parametre.

U radu [10] je dizajnirana platforma da poboljša bezbjednost na putevima i smanji negativan uticaj transporta na životnu sredinu. Studija je uključivala grupu od 100 učesnika koji su bili izloženi različitim scenarijima u virtuelnom okruženju. Rezultati su pokazali da je korišćenje platforme povećalo znanje učesnika o održivim putevima i pozitivno uticalo na njihov stav prema održivom transportu. Autori sugerišu da ova vrsta platforme ima potencijal da bude dragocjeno sredstvo za promovisanje planiranja urbane mobilnosti i edukacije javnosti o ekološkim praksama.

Sve u svemu, ove studije pokazuju važnost praćenja i upravljanja saobraćajem u urbanim sredinama i ističu potencijal IoT i tehnika mašinskog učenja za poboljšanje uslova saobraćaja i poboljšanje urbane mobilnosti [11] [12] [14].

Studije pokazuju širok spektar tehnologija i pristupa dostupnih za razvoj sistema upravljanja saobraćajem.

1.3. Metodologija istraživanja

Naučne metode koje su primijenjene u ovom istraživanju baziraju se na: kombinaciji različitih metoda, uključujući prikupljanje, obradu, integraciju, vizualizaciju, testiranje i simulaciju podataka. Svi ovi koraci su važni kako bi se osigurala kvalitetna i pouzdana analiza podataka o saobraćaju.

1. Prikupljanje podataka: Podaci o saobraćaju mogu se prikupljati iz različitih izvora, uključujući senzore, GPS uređaje i druge dostupne izvore podataka. U cilju prikupljanja podataka, mogu se koristiti i unaprijed prikupljeni podaci ili generisani podaci, ako nema dovoljno dostupnih izvora.

2. Obrada podataka: Nakon prikupljanja podataka, podaci se obrađuju pomoću različitih alata za obradu podataka, uključujući Apache Spark i Apache Kafka. Ova obrada podataka omogućava sistematizaciju i organizaciju velike količine podataka, što olakšava njihovu obradu i analizu.

3. Integracija podataka: Integracija podataka o saobraćaju može se postići korišćenjem različitih alata i tehnologija za skladištenje i upravljanje podacima. U ovom slučaju, baza podataka koja će se koristiti za skladištenje podataka o saobraćaju treba biti dobro organizovana i prilagođena potrebama analize i vizualizacije.

4. Vizualizacija podataka: Podaci o saobraćaju mogu se vizualizovati kroz jedinstveni korisnički interfejs koji omogućava donosiocima odluka da lako pristupe relevantnim informacijama.

5. Testiranje i simulacija: Potrebno je testirati funkcionalnost platforme i provesti samu simulaciju. Ovo testiranje može uključivati provjeru performansi platforme, provjeru njene skalabilnosti, kao i testiranje kako bi se provjerilo da li platforma na kvalitetan način pruža informacije o saobraćaju.

1.4. Očekivani rezultati, primjena i doprinosi

Očekivani rezultati istraživanja su:

- Uspješna integracija podataka o saobraćaju iz različitih izvora (senzori, GPS uređaji itd.) korišćenjem različitih alata i tehnologija.
- Funkcionalna platforma koja će biti lako dostupna donosiocima odluka, uz intuitivan i jednostavan korisnički interfejs.
- Mogućnost vizualizacije podataka o saobraćaju kroz interaktivni korisnički interfejs, što će omogućiti korisnicima da lako pristupe i analiziraju relevantne informacije.
- Prikaz informacija o saobraćaju koje se nude na platformi, što će omogućiti donosiocima odluka da brzo i efikasno donose odluke o saobraćaju.
- Učinkovitost i skalabilnost platforme u odnosu na veliku količinu podataka koje će se generisati i obrađivati.

Eventualna praktična primjena rezultata istraživanja integracije podataka o saobraćaju kroz web platformu sa vizualizacijom može biti širokog opsega. Neki od primjera mogu uključivati:

1. Unapređenje saobraćajne sigurnosti: Integracija podataka o saobraćaju i vizualizacija istih kroz jedinstveni korisnički interfejs može doprinijeti unapređenju saobraćajne sigurnosti. Donosioci odluka u saobraćaju mogu koristiti ove informacije za donošenje brzih odluka i akcija u cilju smanjenja saobraćajnih nesreća.
2. Planiranje i upravljanje saobraćajem: Integracija podataka o saobraćaju i vizualizacija istih kroz web platformu može pomoći u planiranju i upravljanju saobraćajem. Ove informacije mogu se koristiti za identifikaciju područja saobraćajnih gužvi i preusmjeravanje saobraćaja na alternativne rute, što može smanjiti gužve i ubrzati kretanje saobraćaja.
3. Unapređenje postojećih tehnologija: Rezultati ovog istraživanja imaju potencijal doprinijeti unapređenju postojećih tehnologija za prikupljanje, obradu i vizualizaciju podataka, što može dovesti do razvoja novih inovativnih proizvoda i usluga. Unapređenje tih tehnologija ima potencijal da dovede do razvoja novih inovativnih proizvoda i usluga u saobraćajnom sektoru. Na primjer, mogu se razviti napredni sistemi upravljanja saobraćajem koji koriste podatke prikupljene

slično kao ova web platforma kako bi se predvidjela gužva, optimizovala ruta ili upozorila na potencijalne probleme u saobraćaju.

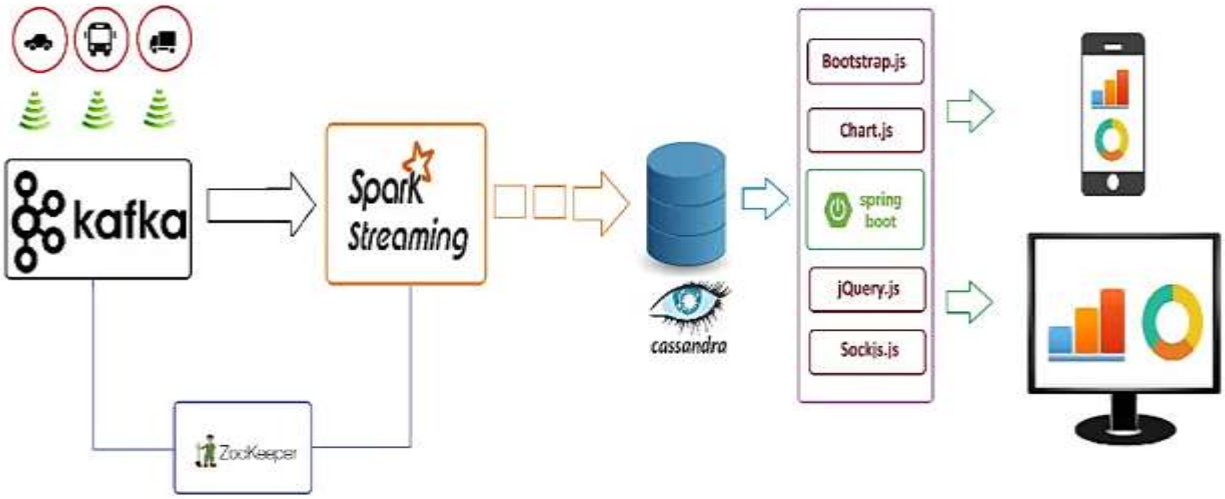
Ukupno, praktična primjena rezultata istraživanja integracije podataka o saobraćaju kroz web platformu može biti vrlo široka i korisna u mnogim sektorima, što bi moglo doprinijeti poboljšanju kvaliteta života ljudi i unapređenju efikasnosti poslovnih procesa.

Očekivani doprinos ovog istraživanja u odnosu na postojeća istraživanja uključuje integraciju različitih alata za prikupljanje, obradu, integraciju i vizualizaciju podataka o saobraćaju u jedinstvenu platformu, uz testiranje funkcionalnosti platforme. Ovo istraživanje takođe ima za cilj identifikaciju najboljih praksi za integraciju podataka o saobraćaju, što bi moglo biti korisno za druge projekte koji se bave sličnim izazovima. Konačno, očekuje se da će ovaj rad imati primjenu u rješavanju izazova u praćenju i upravljanju saobraćajem u urbanim područjima, što bi moglo doprinijeti poboljšanju efikasnosti, sigurnosti i održivosti transporta u gradovima.

2. Analiza tehnologija/alata

U nastavku će biti prikazana cjelokupna arhitektura razvoja web platforme koja posjeduje jedinstveni dizajn za praćenje saobraćaja, uz detaljno objašnjenje svih alata koji se koriste.

Cjelokupna arhitektura je:



Slika 1: Arhitektura alata za monitoring saobraćaja

Slika 2: Prikaz izgleda platforme za praćenje saobraćaja.

2.1. Apache Spark

Apache Spark predstavlja jedno od najbržih rješenja za operacije koje sadrže veliki broj podataka. Nastao je kao rješenje otvorenog koda (open source) 2009. godine na UC Berkeley univerzitetu [1]. Ovo rješenje je zbog svoje sveobuhvatnosti i dobre podrške zasijenio Hadoop. Apache Spark ima mnogo veće mogućnosti u pogledu raspodjele memorije, brzine i resursa.

Od njegovog prvog izdanja Apache Spark zbog open source osobine je izabran od mnogih kompanija koje vrše obradu podataka u industrijske svrhe. Internet giganti Netflix, Yahoo i eBay koriste Apache spark u pozadini svojih procesa da bi obrađivali petabajtove podataka (2^50 bajtova) na klasterima od preko 8000 čvorova (nodova). Veoma je brzo postao najprihvatljivije i najveće okruženje za obradu velike količine podataka sa više od 1000 saradnika iz 250 organizacija.

Slika 3: Svestranost Apache Spark rješenja
https://jelvix.com/blog/hadoop-vs-spark-what-to-choose-to-process-big-data

Hadoop MapReduce je model programiranja koji se koristi za efikasno procesuiranje velikih skupova podataka kroz paralelne i distribuirane algoritme. Međutim, ovaj model ima određene izazove kada je riječ o sekvencijalnom izvršavanju koraka i performansama zbog ulazno-izlaznih (input/output) operacija na disku.

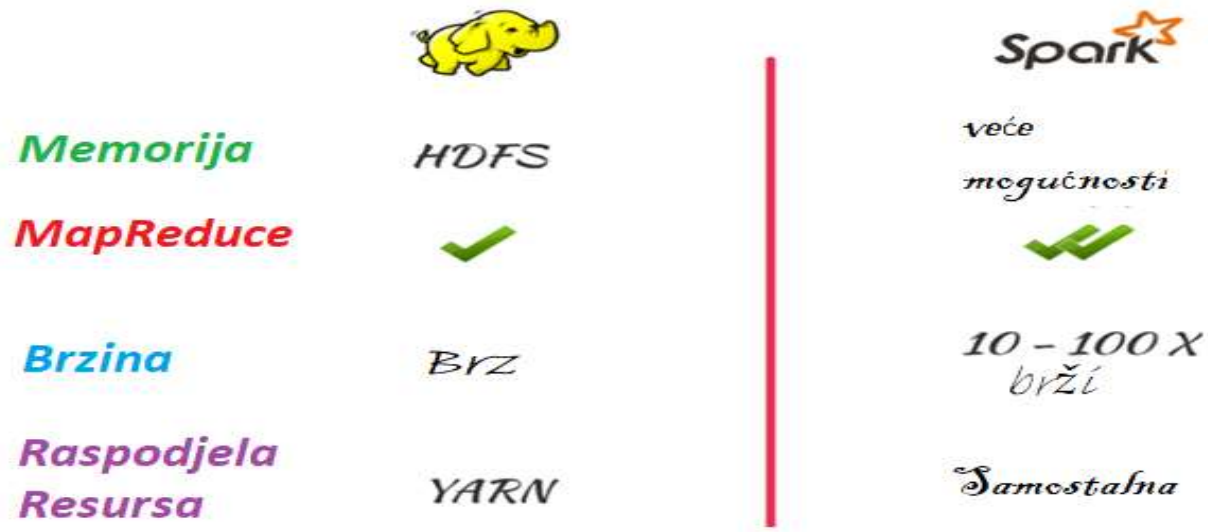
Kada se koristi Hadoop MapReduce, podaci se čitaju sa distribuiranog klastera i određene operacije se izvršavaju nad tim podacima. Nakon toga, rezultat se vraća u Hadoop Distributed File

System (HDFS). Međutim, svaki korak u procesu zahtijeva upis i čitanje podataka, što može dovesti do zastoja zbog vremena koje je potrebno za ulazno-izlazne operacije na hard disku.

Spark je kreiran da odgovori na ograničenja za MapReduce, smanjuje broj koraka potrebnih za izvršenje i ponovo koristi podatke u više paralelnih operacija. Sa Spark-om, potreban je samo jedan korak gdje se podaci čitaju iz memorije, izvršavaju operacije i vraćaju rezultati – što rezultira mnogo bržim izvršavanjem. Spark takođe ponovo koristi podatke koristeći keš memoriju da bi u velikoj mjeri ubrzao algoritme mašinskog učenja koji više puta pozivaju funkciju. Ponovna upotreba podataka se postiže kreiranjem DataFrames-a, kao apstrakcije preko otpornog distribuiranog skupa podataka (RDD), koji je kolekcija objekata koji se keširaju u memoriji i ponovo koriste u više Spark operacija. Ovo dramatično smanjuje kašnjenje čineći Spark višestruko bržim od MapReduce-a, posebno kada se radi o mašinskom učenju i interaktivnoj analitici.

2.1.1. Apache Spark vs Hadoop

Iako se Hadoop MapReduce i Spark razlikuju po dizajnu mnoge organizacije su ih prihvatile kao komplementarna rješenja koriteći ih za rješavanje šireg spektra problema. Hadoop predstavlja open source rješenje koje koristi HDFS (Hadoop Distributed File System) za skladištenje, YARN (Yet Another Resource Negotiator) za raspolaganje resursima i MapReduce kao model izvršnog engine-a. U Hadoop-u mogu da se koriste i drugi izvršni engine-i kao što su Spark, Tez i Pesto.



Slika 4: Razlike između Hadoop i Spark

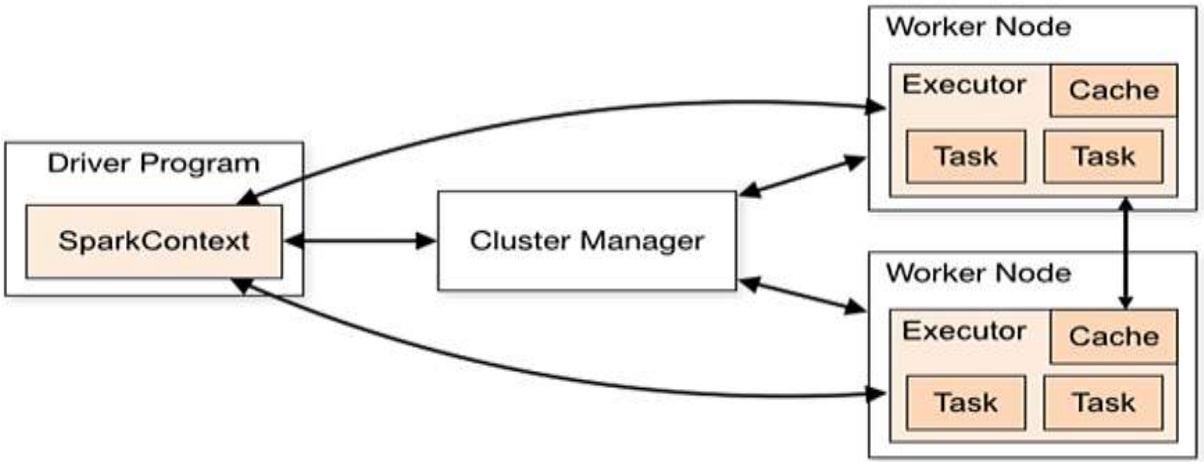
Spark je open source frejmvork fokusiran na interaktivnim upitima, mašinskom učenju i radnjama u realnom vremenu. Nema sopstveni sistem za čuvanje podataka kao Hadoop (HDFS) ali ima mogućnost da vrši različite analitčke zadatke na sistemima za čuvanje kao što je HDFS ili drugim kao što su Casandra, Amazon Redshift, Couchbase i drugi.

Što se tiče osnovnih karakteristika Spark nikada nije imao cilj da potpuno zamijeni Hadoop već da ga upotpuni i učini boljim. Osnova je brzina računanja gdje se Spark može pohvaliti brzinama 100 puta većim u odnosu na Hadoop što ga stvarno čini boljim rješenjem. Neki koncepti koji koriste MapReduce takođe su primjenjeni u Sparku samo sada na drugačiji bolji način. Posebno je dobro rješenje kada se problemi susrijeću sa podacima na kojima se vrše operacije koje se stalno čitaju i pišu na disk. Zašto? Većina operacija koje izvršava Spark biće izvršavane iz memorije (najčešće keš memorije) pa su operacije izuzetno brze i efikasne.

Dakle, snaga Spark-a leži u njegovoj brzini i njegov mehanizam za izvršavanje je veoma efikasan u poređenju sa Hadoop-ovom MapReduce implementacijom, a Spark je savršeno dizajniran da radi na vrhu postojećeg Hadoop klastera koji koristi YARN za upravljanje resursima i HDFS za skladištenje.

2.1.2. Apache Spark arhitektura

Apache Spark je klasični primjer master-slave arhitekture. Master će se zvati glavni čvor (Driver program) a slave dio će biti radni čvorovi (Worker node). U daljem dijelu rada gdje god bude pominjana arhitektura biće korišćena engleska nomenklatura zarad lakšeg objašnjena. Kada bude kreirana Spark platforma i pokrenuta Driver će kreirati ulaznu tačku unutar cijelog sistema, sve operacije će biti rađene na Worker čvoru od strane izvršioca (Executor) a menadžmentom resursa će se baviti Cluster (klaster) menadžment.



Slika 5: Arhitektura Apache Spark okruženja

https://0x0fff.com/spark-architecture/

Spark Driver će biti osnovni dio Spark koji je tu da održava sva stanja platforme koje rade na Spark klasteru. Da bi kasnije dobili resurse i pokrenuli same Worker čvorove Spark Driver mora biti povezan sa menadžerom klastera.

Svi zadaci dobijeni od Spark Drivera će biti izvršeni preko Worker node i izvršioca kao Executer. Njihov zadatak je da izvrše dobijene zadatke i na neki način pošalju informaciju o tome dali je rješavanje zadataka bilo uspješno ili ne. U suštini svaki program u Sparku će morati da ima Executor-e.

Klaster menadžer predstavlja više mašina koje će da izvršavaju Spark platforme. Njegov rad je baziran na apstrakciji master i slave nodova. U pozadini se vrše Deamon procesi (procesi koji ne zahtijevaju rad programera već rade sami u pozadini) koji vrše organizaciju ostalih djelova kao što je Driver program i Worker node. Postoje mnogi načini po kojima će Cluster Manager raditi. Claster Manager Spark može da ima :

1. Standalone - samostalni Spark klaster menadžer koji je veoma lako postaviti i koristiti;
2. Apache Mesos –Mesos će imati određene funkcionalnosti koje će dopustiti da se izvršava Hadoop Map reduce kao i ostale Spark platforme;
3. Hadoop Yarn osnovni i najčešće korišćeni klaster manadzer;
4. Kubernetes – open source sistem za automatizaciju primjene, skaliranja i upravljanja kontejnerskim platformama.

Ovdje je primjetno da Spark ima veoma velike moći adaptacije i totalno različitih primjena za istu arhitekturu.

Apache Kafka je distribuirana platforma za striming koja se koristi za izgradnju protoka podataka u realnom vremenu i platformi za striming. Dizajniran je za rukovanje velikim tokovima podataka, velike propusnosti i tolerantnih na greške. Kafka obezbjeđuje model objavljivanja i pretplate gdje proizvođači pišu podatke u teme, a potrošači čitaju podatke iz tema.

2.1.3. Apache Spark RDD (Resilient Distributed Dataset)

RDD predstavlja osnovnu strukturu podataka i takođe i primarnu apstrakciju podataka vezanu za Apache Spark. RDD će predstavljati nepromjenljive distribuirane kolekcije podataka koje su otporne na greške. To znači da jednom kreiran RDD neće moći da se izmijeni. Skupovi podataka u RDD-u će dalje biti podijeljeni u logičke djelove na kojim se može izršavati izračunavanje na različitim nodovima klastera.

Predstavljat će skup podataka sličan kolekcijama u programskom jeziku Scala stim da će RDD vršiti određena izračunavanja na više Java virtuelnih mašina drugačije objašnjenih kao JVM (Java Virtual Machine) na više fizičkih servera odnosno čvorova u klasteru, dok će sama Scala programski jezik raditi na jednom JVM.

RDD-ovi obezbjeđuju apstrakciju podataka particionisanja i distribucije podataka koji su dizajnirani da paralelno izvode izračunavanja na nekoliko čvorova, dok obavljajući transformacije na RDD-u nije potrebno misliti o paralelizmu jer ga Spark obezbeđuje.

Prednosti otpornog distribuiranog dataseta (RDD) su:

1. In-Memory Processing - procesuiranje unutar memorije;
2. Immutability – nepromjenljivost;
3. Fault Tolerance - tolerancija greške;
4. Lazy Evolution - odgođena evaluacija;
5. Partitioning - dijeljenje na particije;
6. Parallelize - paralelizovanje.

2.1.4. Princip upotrebe RDD-a

RDD predstavlja osnovnu strukturu podataka i takođe i primarnu apstrakciju podataka vezanu za Apache Spark.

Prvo će biti definisana sesija. Sastoji se od metoda za kreiranje šablona *builder()* i poziva *GetOrCreate()* metod. Ako već postoji sesija ovaj poziv će samo da vrati postojeću, a ako ne postoji kreiraće novu Spark sesiju. U power shell-u će biti ispisano:

```
scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala> object SparkSessionTest extends App {
     |   val spark = SparkSession.builder()
     |       .master("local[1]")
     |       .appName("SparkByExamples.com")
     |       .getOrCreate();
     |   println(spark)
     |   println("Spark Version : "+spark.version)
     | }
defined object SparkSessionTest
```

Slika 6: Prikaz definisanja i izvršavanja Spark sesije

Pošto se sesija zvala spark sada kada bude ukucano spark na komandnoj liniji rješenje će biti:

```
scala> spark
res2: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@40a38a44

scala> print(spark)
org.apache.spark.sql.SparkSession@40a38a44
```

Slika 7: Prikaz izvršavanja Spark print komande

Sada je lako izvršavati druge zadatke unutar Spark shell-a. Moguće je dobiti unutar Scala programskog jezika Spark sesiju programerski, kreirati novu sesiju, konfigurisati je.

```
scala> val spark2 = SparkSession.builder().getOrCreate()
spark2: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@40a38a44

scala> print(spark2)
org.apache.spark.sql.SparkSession@40a38a44
scala>

scala> val spark3 = spark.newSession()
spark3: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@6bfe7897

scala> print(spark3)
org.apache.spark.sql.SparkSession@6bfe7897
scala> val spark = SparkSession.builder()
spark: org.apache.spark.sql.SparkSession.Builder = org.apache.spark.sql.SparkSession$Builder@28c7ae7f

scala>       .master("local[1]")
res6: org.apache.spark.sql.SparkSession.Builder = org.apache.spark.sql.SparkSession$Builder@28c7ae7f

scala>       .appName("SparkByExamples.com")
res7: org.apache.spark.sql.SparkSession.Builder = org.apache.spark.sql.SparkSession$Builder@28c7ae7f

scala>       .config("spark.some.config.option", "config-value")
res8: org.apache.spark.sql.SparkSession.Builder = org.apache.spark.sql.SparkSession$Builder@28c7ae7f

scala>       .getOrCreate();
23/01/17 12:43:40 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.
res9: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@40a38a44
```

Slika 8: Kreiranje, editovanje i konfigurisanje Spark sesija

Dalje će biti kreirani Data frame-ovi da bi se mogao kreirati i DataSet podataka. Prikaz će biti u vidu tabele.

```
scala> val df = spark.createDataFrame(
     |     List(("Scala", 25000), ("Spark", 35000), ("PHP", 21000)))
df: org.apache.spark.sql.DataFrame = [_1: string, _2: int]

scala> df.show()
+-----+-----+
|   _1|   _2|
+-----+-----+
|Scala|25000|
|Spark|35000|
|  PHP|21000|
+-----+-----+
```

Slika 9: Spark Data Frame funkcija

Sada je već moguć prikaz preko RDD-a. Prvo će biti kreirana sekvenca podataka pa će ta sekvenca biti paralelizovana.

```
scala> val dataSeq = Seq(("Java", 20000), ("Python", 100000), ("Scala", 3000))
dataSeq: Seq[(String, Int)] = List((Java,20000), (Python,100000), (Scala,3000))

scala> val rdd=spark.sparkContext.parallelize(dataSeq)
rdd: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[0] at parallelize at <console>:23
```

Slika 10: RDD paralelizacija

Već su moguće RDD transformacije. One se definišu kao lijenje operacije jer neće biti izvršavane dok ne bude pozvana akcija na RDD. Pošto RDD karakteristiše nepromjenljivost transformacije će vratiti novi RDD, tj. neće mijenjati stari. Funkcije za transformaciju su *count(), collect(), first(), max(), reduce()* i druge.

Moguće je transformisati RDD u Data frame. Ovo je potrebno jer Data frame i Data set imaju niz pogodnosti u odnosu na RDD. Jedna od pogodnosti je što Data frame-ovi predstavljaju kolekciju podataka koji su distribuirani u kolone slično kao kod baza podataka što ubrzava optimizacije i povećava performanse.

```
scala> import spark.implicits._
import spark.implicits._

scala> val columns = Seq("language","users_count")
columns: Seq[String] = List(language, users_count)

scala> val data = Seq(("Java", "20000"), ("Python", "100000"), ("Scala", "3000"))
data: Seq[(String, String)] = List((Java,20000), (Python,100000), (Scala,3000))

scala> val rdd = spark.sparkContext.parallelize(data)
rdd: org.apache.spark.rdd.RDD[(String, String)] = ParallelCollectionRDD[1] at parallelize at <console>:26
```

Slika 11: Kreiranje RDD sekvence koju treba prebaciti u Data frame

Konvertovanje Spark RDD u DataFrame može se obaviti korišćenjem *toDF(), createDataFrame()* i transformisanjem RDD[Red] u Data frame. Osnovna i najakša funkcija je *doDF()* koja se pored RDD može koristiti za konvertovanje sekvence (Seq()) liste List(). Da bi bilo izvršeno prebacivanje preko *toDF()* može biti pozvan paket koji omogućava implicitno prebacivanje (import spark.implicits).

```
scala> val dfFromRDD1 = rdd.toDF()
dfFromRDD1: org.apache.spark.sql.DataFrame = [_1: string, _2: string]

scala> dfFromRDD1.printSchema()
root
 |-- _1: string (nullable = true)
 |-- _2: string (nullable = true)
```

Slika 12: *toDF()* funkcija

Kao rješenje funkcija će ispisati dvije kolone i daće im imena 1 i 2. Sada je moguće funkciji *toDf()* proslijediti argumente da bi bila definisana imena novo-nastalih kolona.

```
scala> val dfFromRDD1 = rdd.toDF("language","users_count")
dfFromRDD1: org.apache.spark.sql.DataFrame = [language: string, users_count: string]

scala> dfFromRDD1.printSchema()
root
 |-- language: string (nullable = true)
 |-- users_count: string (nullable = true)
```

Slika 13: *toDF()* sa dva argumenta za ime kolona

Ostale funkcije su slične sintakse i neće im biti posvećena dalja pažnja. Sledeća funkcija koja će biti pokazana je *reduceByKey()* sa RDD primjerom. Ova funkcija predstavlja transformaciju gdje se iste vrijednosti spajaju preko asocijativne operacije smanjenja. To je šira transformacija jer miješa podatke preko više particija (RDD djelova) i radi na parovima RDD (par ključ/vrijednost).

ReduceByKey() funkcija je dostupna u *org.apache.spark.rdd.PairRDDFunctions*.

Izlaz će biti podijeljen na numPartitions ili na podrazumijevani nivo paralelizma. Pojavljuje se i hash particija. Prvo će biti predstavljeni podaci koji su paralelizovani i nad njima će biti izvršena *reduceByKey()* operacija.

```
scala>  val data = Seq(("Project", 1),
     |   ("Gutenberg?s", 1),
     |   ("Alice?s", 1),
     |   ("Adventures", 1),
     |   ("in", 1),
     |   ("Wonderland", 1),
     |   ("Project", 1),
     |   ("Gutenberg?s", 1),
     |   ("Adventures", 1),
     |   ("in", 1),
     |   ("Wonderland", 1),
     |   ("Project", 1),
     |   ("Gutenberg?s", 1))
data: Seq[(String, Int)] = List((Project,1), (Gutenberg&s,1), (Alice&s,1), (Adventures,1), (in,1), (Wonderland,1), (Pro
ect,1), (Gutenberg&s,1), (Adventures,1), (in,1), (Wonderland,1), (Project,1), (Gutenberg&s,1))

scala>

scala>   val rdd=spark.sparkContext.parallelize(data)
rdd: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[2] at parallelize at <console>:26
```

Slika 14: Paralelizacija podataka za funkciju *ReduceByKey()*

Da se primijetiti da unutar ovog seta podataka postoje podaci koji se ponavljaju. Na primjer Adventures podatak ima 2 ponavljanja, Project 3 ponavljanja itd. Sada na ovim

podacima treba vršiti operacije smanjenja.

```
scala> val rdd2=rdd.reduceByKey(_ + _)
rdd2: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[3] at reduceByKey at <console>:26

scala>   rdd2.foreach(println)
(Alice?s,1)                                                     (0 + 16) / 16]
(Adventures,2)
(Wonderland,2)
(in,2)
(Project,3)
(Gutenberg?s,3)
```

Slika 15: Nakon izvršenja operacije *ReduceByKey()*

Pošto je data operacija i dati primjeri su sabrani. Cijeli ispisan fajl će biti:

```
scala> object ReduceByKeyExample extends App{
     |
     |   val spark: SparkSession = SparkSession.builder()
     |     .master("local[1]")
     |     .appName("SparkByExamples.com")
     |     .getOrCreate()
     |
     |   val data = Seq(("Project", 1),
     |   ("Gutenberg?s", 1),
     |   ("Alice?s", 1),
     |   ("Adventures", 1),
     |   ("in", 1),
     |   ("Wonderland", 1),
     |   ("Project", 1),
     |   ("Gutenberg?s", 1),
     |   ("Adventures", 1),
     |   ("in", 1),
     |   ("Wonderland", 1),
     |   ("Project", 1),
     |   ("Gutenberg?s", 1))
     |
     |   val rdd=spark.sparkContext.parallelize(data)
     |
     |   val rdd2=rdd.reduceByKey(_ + _)
     |
     |   rdd2.foreach(println)
     |
     | }
defined object ReduceByKeyExample
```

Slika 16: *ReduceByKey()* kod

Na istom principu samo ne za par ključeva će raditi *reduce()*. Preko ove funkcije je moguće izračunati minimum, maksimum takođe i broj elemenata u datasetu i još mnogo toga. Kao i u prošlom primjeru isto će biti definisane određene RDD vrijednosti nad kojima će biti izvršene određene transformacije.

```
scala>  val listRdd = spark.sparkContext.parallelize(List(1,2,3,4,5,3,2))
listRdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:25

scala>   println("output min using binary : "+listRdd.reduce(_ min _))
output min using binary : 1

scala>   println("output max using binary : "+listRdd.reduce(_ max _))
output max using binary : 5

scala>   println("output sum using binary : "+listRdd.reduce(_ + _))
output sum using binary : 20
```

Slika 17: Obični reduce sa različitim operacijama

Sada se mogu dalje definisati slične operacije bazirane najčešće na tautologiji gdje se od cijelog dataseta mogu uzeti pojedini članovi i nad njima izvršiti *reduce()*.

```
scala>  val listRdd = spark.sparkContext.parallelize(List(1,2,3,4,5,3,2))
listRdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[1] at parallelize at <console>:25

scala>   println("output min : "+listRdd.reduce( (a,b) => a min b))
output min : 1

scala>   println("output max : "+listRdd.reduce( (a,b) => a max b))
output max : 5

scala>   println("output sum : "+listRdd.reduce( (a,b) => a + b))
output sum : 20
```

Slika 18: *reduce()* operacije za specijalne slučajeve

Dalje je moguće vršiti kompleksnije operacije kao što su stepenovanje, povezivanje intigera i stringa (po istom principu kao i kod *ReduceByKey()*) često korišćene za kompleksnije operacije u platformama proizvodnje.

```
scala> val inputRDD = spark.sparkContext.parallelize(List(("Z", 1),("A", 20),("B", 30),("C", 40),("B", 30),("B", 60)))
inputRDD: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[2] at parallelize at <console>:25

scala>

scala>   println("output min : "+inputRDD.reduce( (a,b)=> ("max",a._2 min b._2))._2)
output min : 1

scala>   println("output max : "+inputRDD.reduce( (a,b)=> ("max",a._2 max b._2))._2)
output max : 60

scala>   println("output sum : "+inputRDD.reduce( (a,b)=> ("Sum",a._2 + b._2))._2)
output sum : 181
```

Slika 19: Kompleksnije operacije sa *reduce()*

Sledeća funkcija koja će biti prikazana je *fold()*. Isto se koristi za pronalaženje minimuma, maksimuma i totalnog broja elemenata. Ona će vršiti agregaciju elemenata unutar svake particije (jer RDD vrši dijeljenje na paralelne particije) i nakon toga vrši krajnju agregaciju

sa svim ostalim participima.

```
scala>  val listRdd = spark.sparkContext.parallelize(List(1,2,3,4,5,3,2))
listRdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:22

scala>   println("Partitions : "+listRdd.getNumPartitions)
Partitions : 16

scala>   println("Total : "+listRdd.fold(0)((acc,ele) => {acc + ele}))
Total : 20

scala>   println("Total with init value 2 : "+listRdd.fold(2)((acc,ele) => {acc + ele}))
Total with init value 2 : 54

scala>   println("Min : "+listRdd.fold(0)((acc,ele) => {acc min ele}))
Min : 0

scala>   println("Max : "+listRdd.fold(0)((acc,ele) => {acc max ele})) 23/01/17 14:30:08 WARN ProcfsMetricsGetter: Excep
tion when trying to compute pagesize, as a result reporting of ProcessTree metrics is stopped

Max : 5
```

Slika 20: Funkcija *fold()*

Kao i *reduce()* ona će imati i upotrebu sa uparenim vrijednostima.

```
scala>  val listRdd = spark.sparkContext.parallelize(List(1,2,3,4,5,3,2))
listRdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:22

scala>   println("Partitions : "+listRdd.getNumPartitions)
Partitions : 16

scala>   println("Total : "+listRdd.fold(0)((acc,ele) => {acc + ele}))
Total : 20

scala>   println("Total with init value 2 : "+listRdd.fold(2)((acc,ele) => {acc + ele}))
Total with init value 2 : 54

scala>   println("Min : "+listRdd.fold(0)((acc,ele) => {acc min ele}))
Min : 0

scala>   println("Max : "+listRdd.fold(0)((acc,ele) => {acc max ele})) 23/01/17 14:30:08 WARN ProcfsMetricsGetter: Excep
tion when trying to compute pagesize, as a result reporting of ProcessTree metrics is stopped

Max : 5
```

Slika 21: *fold()* za kompleksnije slučajeve

Posljednja funkcija koja će biti objašnjena je *map()*. Ona je veoma moćan alat koji se koristi za primjenu transformacije na svaki element RDD-a, Data Set-a i Data frame-a i konačno vraća novi RDD/skup podataka. Kao i do sada prvo će biti kreiran skup podataka paralelizovan u RDD i nad njime će biti izvršene transformacije.

```
scala> val data = Seq("Project",
     |   "Gutenberg?s",
     |   "Alice?s",
     |   "Adventures",
     |   "in",
     |   "Wonderland",
     |   "Project",
     |   "Gutenberg?s",
     |   "Adventures",
     |   "in",
     |   "Wonderland",
     |   "Project",
     |   "Gutenberg?s")
data: Seq[String] = List(Project, Gutenberg&s, Alice&s, Adventures, in, Wonderland, Project, Gutenberg&s, Adventures, i
, Wonderland, Project, Gutenberg&s)

scala>

scala>   val rdd=spark.sparkContext.parallelize(data)
rdd: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[2] at parallelize at <console>:23
```

Slika 22: Podaci koji će biti korišćeni za *Map()*

U sledećem primjeru će biti dodat novi elelment sa vrijednošću 1 za svaki element. Kao rezultat će biti dobijeni parovi sa ključevima gdje će svaki ključ biti uparen sa jedinicom.

```
scala> val rdd2=rdd.map(f=> (f,1))
rdd2: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[3] at map at <console>:23

scala> rdd2.foreach(println)
(Gutenberg?s,1)
(Project,1)
(Wonderland,1)
(in,1)
(Adventures,1)
(Wonderland,1)
(Gutenberg?s,1)
(Alice?s,1)
(in,1)
(Project,1)
(Gutenberg?s,1)
(Adventures,1)
(Project,1)
```

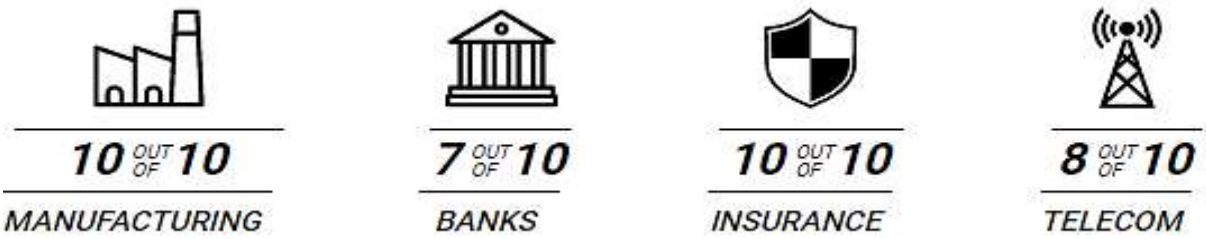
Slika 23 : Primjer korišćenja *Map() funkcije*

Dalje je moguće preko *toDF()* prebacati podatke u Data frame i dalje transformisati sve particije.Spark je moćno rješenje koje ima mogućnost kreiranja virtuelnih mašina (JVM), ima brzine izvršenja do 100 puta veće od samog Hadoopa i takođe se razuđeno (na više odvojenih pravaca razvija). Sve to dovodi do sve većeg korišćenja Sparka, a ovdje su samo postavljene osnovne stavke rada sa RDD-om.

2.2. Apache Kafka

Platforma otvorenog koda koja služi za distribuirani striming događaja koju koriste hiljade kompanija za prenos podataka visokih performansi, striming analitiku, integraciju

podataka i kritične aplikacije. Kafka je vrlo popularna platforma za obradu strimova podataka u realnom vremenu, a primenjuje se u različitim oblastima, kao što su finansijske usluge, telekomunikacije, internet stvari, obrada logova i drugo.

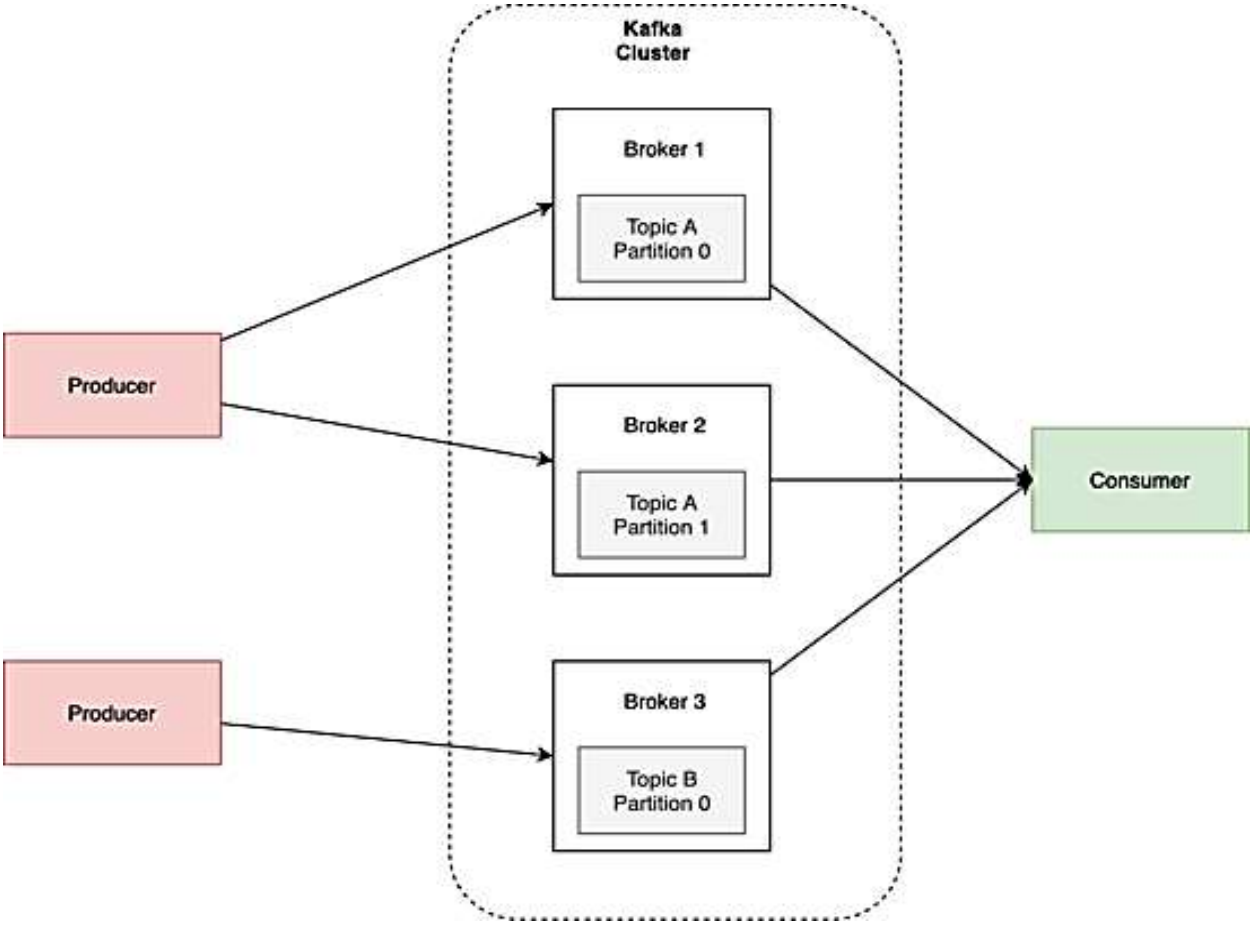


Slika 24: Primjena Apache Kafke
https://kafka.apache.org/

Kafka se sastoji od nekoliko ključnih komponenti:

1. Broker: Kafka broker je server koji prima, skladišti i šalje poruke. Brokerski klaster se sastoji od jednog ili više servera koji se nazivaju Kafka brokers.
2. Producers: Producers su klijenti koji šalju poruke na Kafka broker. Producers mogu biti dio bilo koje aplikacije koja želi slati poruke u Kafka klaster.
3. Consumers: Consumers su klijenti koji primaju poruke sa Kafka brokera. Konzumiranje podataka izvršava se čitanjem poruka sa određenih Kafka tema.
4. Topics: Kafka topics su kanali za koje proizvođači šalju poruke i sa kojih potrošači konzumiraju poruke. Svaki Kafka topic se sastoji od jednog ili više particija.
5. Partitions: Particije su djelovi Kafka topica koji su fizički raspoređeni po Kafka brokerima. Poruke se zapisuju u particije na brokerima i potrošači ih konzumiraju.

Kafka omogućava obradu velikih količina podataka u realnom vremenu, visoku dostupnost i horizontalno skaliranje. To ga čini popularnim rješenjem za procesuiranje streaming podataka, logova i drugih aplikacija koje zahtijevaju visok protok podataka.



Slika 25: Apache Kafka Klaster

https://mindmajix.com/kafka-tutorial

Karakteristike Kafke su:

1. Može da isporuči poruke sa ograničenom propusnošću koristeći grupu mašina. Kafka ima visoku propusnost.
2. Obradu vrši hronološki, što znači da obrađuje podatke događaj po dogadjaj i to sve sa malim kašnjenjem.
3. Obrađuje podatke tako što čuva stanje svakog događaja. Podržava obradu stanja uključujući distribuirano spajanje i agregaciju.
4. Može se implementirati koristeći popularne jezike kao sto su Java/Scala, a ima svoj jednostavan jezik specifičan za domen (DSL).
5. Pruža mogućnost obrade događaja podataka pomoću Windowinga.
6. Nema zastoja u stalnim implementacijama i ima distribuiranu obradu i toleranciju grešaka sa brzim prelaskom na grešku.

2.3. ZooKeeper

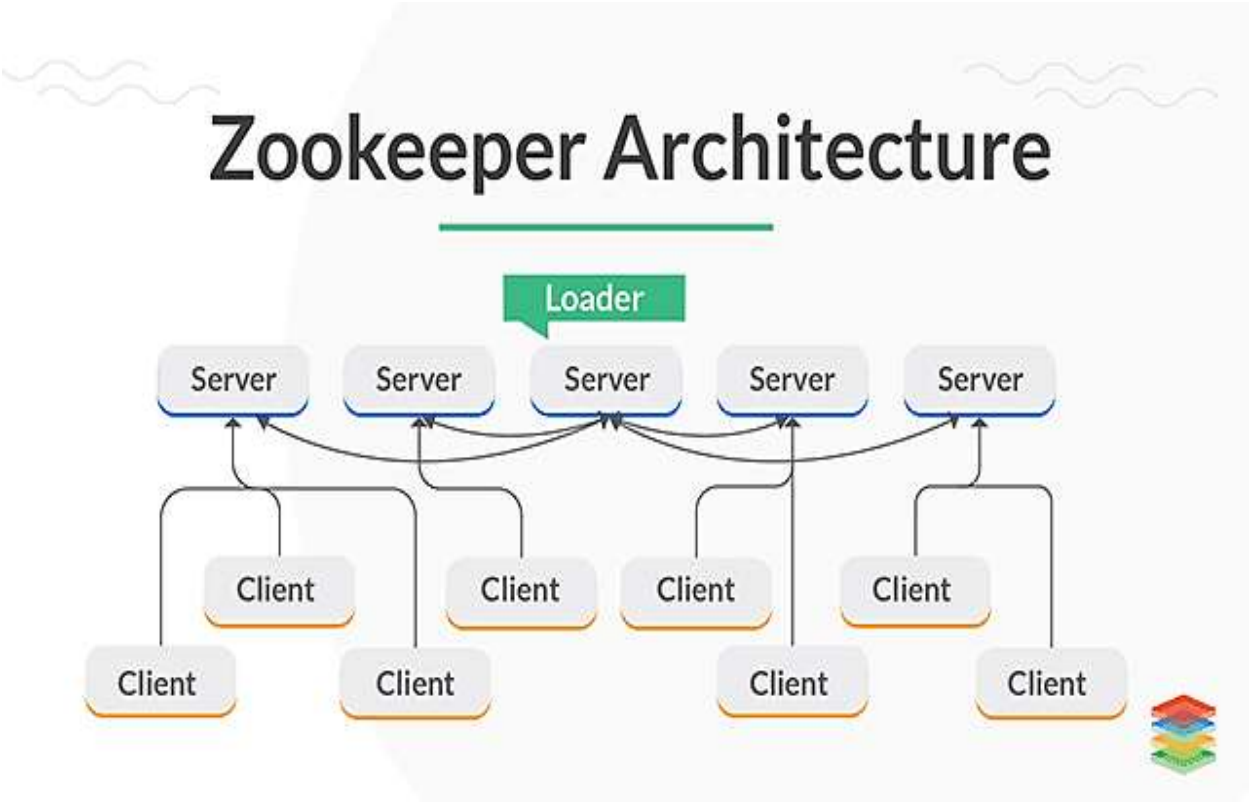
ZooKeeper je distribuirana koordinaciona usluga otvorenog koda koja se koristi za upravljanje i koordinaciju distribuiranih sistema. Često se koristi zajedno sa Apache Kafkom i drugim distribuiranim sistemima da obezbijedi funkcije kao što su upravljanje distribuiranom konfiguracijom, izbor lidera i distribuirana sinhronizacija.

ZooKeeper obezbeđuje hijerarhijsko skladište ključ-vrijednost, slično sistemu datoteka, gdje se podaci čuvaju i pristup vrši pomoću putanja. Takođe, podržava atomske operacije, kao što su operacije čitanja-izmjene-pisanja i satove (eng. watches), koji omogućavaju klijentima da primaju obavještenja kada se podaci promijene.

ZooKeeper koristi repliciranu arhitekturu, gdje više ZooKeeper čvorova formira klaster, a podaci se repliciraju preko ovih čvorova. Ovo obezbjeđuje otpornost na greške, jer jedan kvar na jednom čvoru ne dovodi do gubitka podataka. ZooKeeper koristi konsenzus algoritam nazvan ZAB (ZooKeeper Atomic Broadcast) da bi osigurao dosljednost i koordinaciju između čvorova.

Neke od karakteristika ZooKeeper-a uključuju:

- Visoka dostupnost: ZooKeeper je dizajniran da pruži visoku dostupnost. Podaci se replikuju na više čvorova (tzv. ensemble), čime se osigurava redundancija i tolerancija grešaka. U slučaju da jedan ili više čvorova ne uspije, ZooKeeper automatski preuzima ulogu drugih ispravnih čvorova, omogućavajući nastavak rada i nedostatak prekida usluge.
- Koordinacija: ZooKeeper se može koristiti za koordinaciju distribuiranih sistema, kao što su Apache Kafka klasteri, pružanjem centralizovanog i dosljednog pogleda na sistem.
- Atomske operacije: ZooKeeper podržava atomske operacije čitanja, mijenjanja i pisanja, što obezbjeđuje da ažuriranja podataka budu dosljedna i uzastopna.
- Satovi: ZooKeeper podržava satove, koji omogućavaju klijentima da primaju obavještenja kada se podaci promijene. Ovo se može koristiti za implementaciju arhitekture vođene događajima.
- Skalabilnost: ZooKeeper može horizontalno skalirati dodavanjem više čvorova u klaster.



Slika 26: ZooKeeper arhitektura

https://hopetutors.com/blog/big-data/what-is-apache-zookeeper/

2.4. Cassandra

Cassandra je distribuirani NoSQL sistem za upravljanje bazom podataka otvorenog koda koji je prvobitno razvio Facebook. Dizajniran je da rukuje velikim količinama podataka na višestrukim robnim serverima istovremeno pružajući visoku dostupnost i toleranciju grešaka.

Cassandra je baza podataka orijentisana na kolone, što znači da su podaci organizovani u kolone, a ne u redove. Koristi distribuiranu arhitekturu, gdje se podaci repliciraju na više čvorova u klasteru, obezbeđujući visoku dostupnost i toleranciju grešaka. Ova arhitektura omogućava Cassandri horizontalno skaliranje dodavanjem više čvorova u klaster kako podaci rastu.

Cassandra pruža fleksibilan model podataka, sa podrškom za struktuirane, polustruktuirane i nestruktuirane podatke. Takođe podržava fleksibilno skladištenje i preuzimanje podataka, sa funkcijama kao što su automatsko particionisanje, podesiva konzistentnost i ugrađena podrška za replikaciju i rezervnu kopiju.

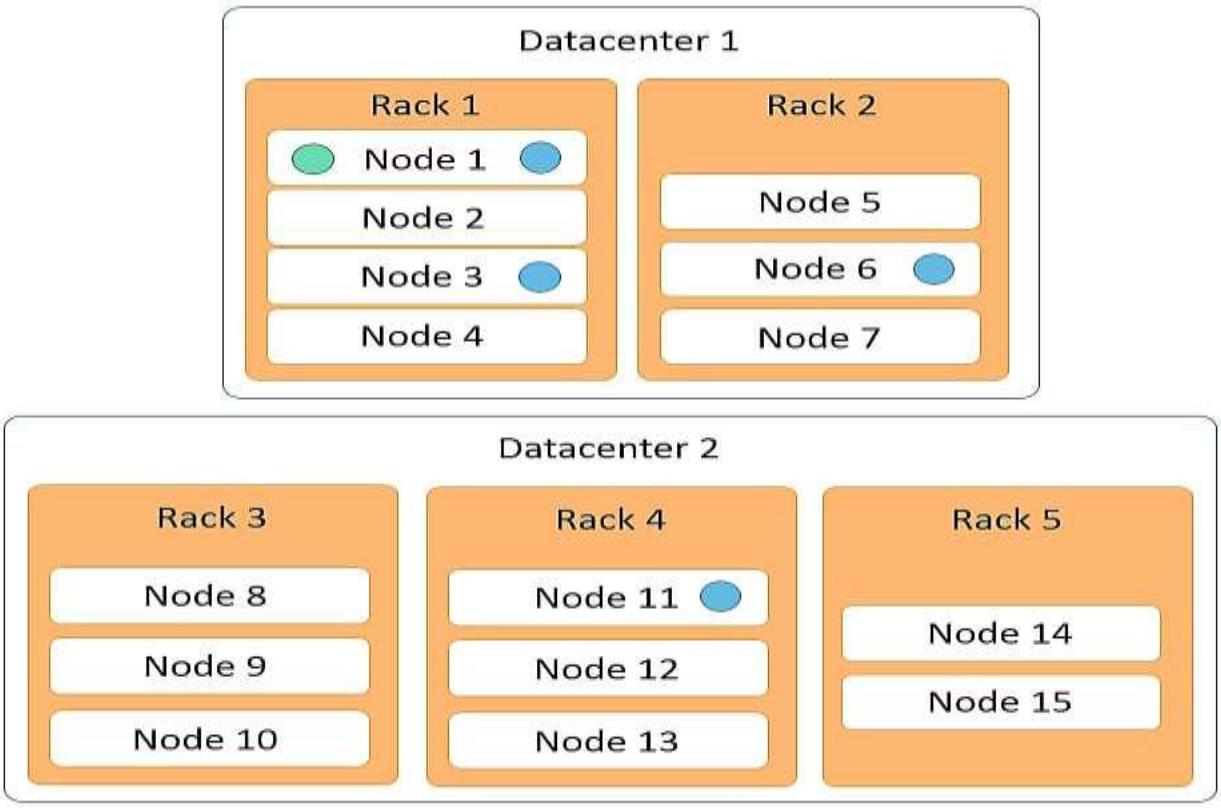
Jedna od ključnih karakteristika Cassandre je njena sposobnost da rukuje velikim količinama podataka sa malim kašnjenjem. Ovo se postiže kroz arhitekturu, koja omogućava

distribuciju podataka na više čvorova, smanjujući količinu podataka koji treba da se prenesu preko mreže.

Cassandra takođe pruža ugrađenu podršku za replikaciju, pri čemu se podaci repliciraju na više čvorova u klasteru. Ovo obezbjeđuje visoku dostupnost i toleranciju grešaka, jer jedan kvar na jednom čvoru ne dovodi do gubitka podataka.

Cassandra podržava podesivu konzistentnost, što omogućava korisnicima da odaberu nivo doslednosti koji im je potreban za svoje podatke. Ovo može biti korisno za aplikacije u kojima je konzistentnost prihvatljiva, jer može smanjiti količinu podataka koji treba da se prenesu preko mreže.

Sve u svemu, Cassandra je moćan i fleksibilan NoSQL sistem za upravljanje bazom podataka koji je dizajniran za visoku dostupnost i toleranciju grešaka. Pogodan je za rukovanje velikim količinama podataka sa malim kašnjenjem, što ga čini idealnim izborom za moderne web platforme koje zahtjevaju veliki broj podataka.



Slika 27: Cassandra arhitektura

https://www.simplilearn.com/tutorials/big-data-tutorial/cassandra-architecture

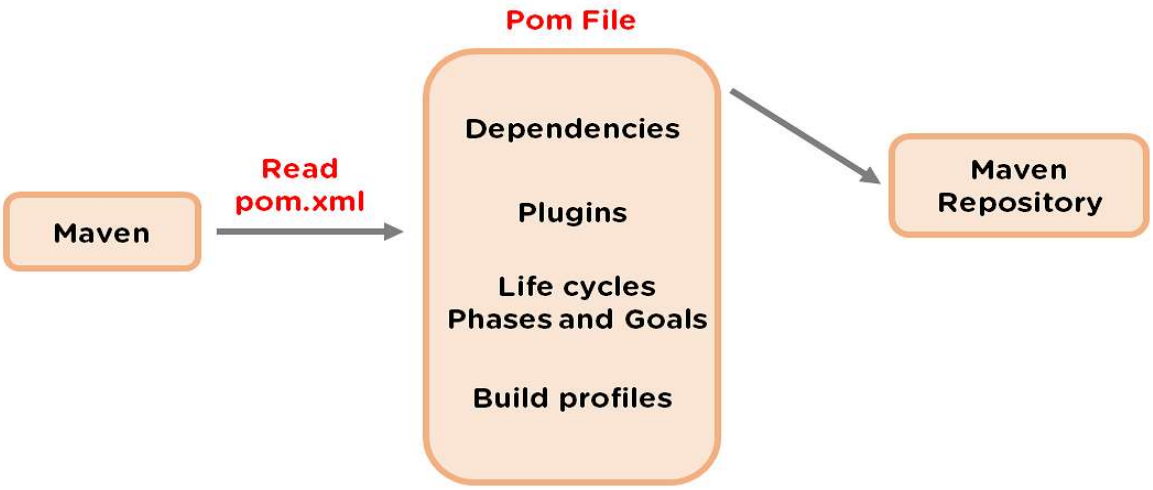
2.5. Apache Maven

Apache Maven je alatka za automatizaciju izgradnje koja se prvenstveno koristi za Java projekte. Pomaže u upravljanju procesom izgradnje projekta, zavisnostima i dokumentacijom. Maven koristi deklarativni konfiguracioni fajl zasnovan na KSML-u (Keep Simplifying Markup Language), nazvan POM (Project Object Model), koji opisuje strukturu projekta i zavisnosti.

Maven pruža standardizovan način izgradnje projekata, olakšavajući programerima da upravljaju životnim ciklusom projekta. Automatizuje mnoge zadatke koji se ponavljaju u izradi projekta, kao što su preuzimanje zavisnosti, kompajliranje koda, pokretanje testova, pakovanje koda u format za distribuciju i generisanje dokumentacije.

Neke ključne karakteristike Maven-a uključuju:

- Upravljanje zavisnostima: Maven pomaže u upravljanju zavisnostima projekta tako što ih preuzima i automatski uključuje.
- Profili izgradnje: omogućava da definišete više profila izgradnje, svaki sa sopstvenim skupom konfiguracionih opcija i ciljeva.
- Sistem dodataka: Maven ima arhitekturu dodataka koja omogućava programerima da lako prošire njegovu funkcionalnost.
- Maven je moćan alat koji pomaže da se pojednostavi proces izgradnje za Java projekte, olakšavajući programerima da se fokusiraju na pisanje koda ili na upravljanje zadacima izgradnje.



Slika 28: Maven funkcionisanje

https://www.simplilearn.com/tutorials/maven-tutorial/introduction-to-maven

3. Arhitektura web platforme

Platforma koja će biti predstavljena je IoT data monitoring platforma koja se koristi Spark Streaming-om. Ovdje će se slati podaci u realnom vremenu stim da se misli na podatke konektovanih vozila da bi se na neki način pratio saobraćaj. Sam alat će biti podijeljen u tri modula i dodatni modul za prikaz pomoću Google Maps API. Ovi moduli predstavljaju samostalne Maven module napisane u programskom jeziku Java u frameworku Spring Boot.

IoT Data Producer predstavlja prvi modul pomoću koga su simulirana povezana vozila. Povezana vozila će generisati IoT poruke koje će biti prikupljene od strane message brokera i poslate platformi za monitoring saobraćaja. Sam IoT Data Producer će biti za simulaciju i koristiti Apache Kafka da bi generisao IoT događaje.

IoT Data Processor će podatke dobijene od više vozila prihvatiti i procesuirati da bi dobili korisne podatke o saobraćaju. U suštini IoT Data Processor će prvo prikupiti sve informacije o broju različitih vozila na različitim putevima i proslijediti u Cassandra bazu podataka. To će biti izvršeno u vremenskom intervalu od poslednjih 5 minuta pri čemu se sve informacije smještaju ponovo u Cassandra bazu podataka. Na osnovu ovih podataka se mogu preuzeti podaci o vozilima od interesa za određeno rastojanje.

IoT Data DashBoard predstavlja Spring Boot platformu koja će preuzimati podatke iz baze podataka napravljene u Cassandri i slati ih na web stranicu. Platforma će koristiti web sockete (za dobro uspostavljanje dvosmjerne TCP konekcije) i jQuery da bi podaci bili poslati web stranici u određenim intervalima. DashBoard će podatke predstavljati preko tabela i grafika. Takođe korišćenjem Bootstrapa dobjamo stranicu koja će se interaktivno mijenjati zavisno od upita i odabira odeđenih opcija (grafika i tabela).

Posljednji modul uključivaće prikaz Google Map API sa označenim putanjama i mogućnošću odabira novih putanja.

Kao što se da primijetiti iako postoje četiri funkcionalna dijela koja predstavljaju platformu, platforma kao cjelina će biti usko povezana preko svojih djelova. Data producer će se baviti generisanjem podataka, Data processor njihovim procesuiranjem, a DashBoard više samim prikazom podataka preko Bootstrap-a, jQuery-ja i Spring Boot-a.

3.1. Instalacioni fajlovi svih potrebnih tehnologija i alata

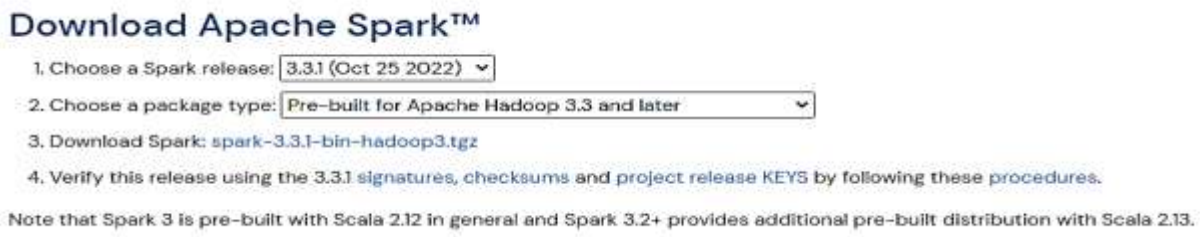
U sljedećoj tabeli će biti predstavljeni linkovi za pojedine sajtove gdje će biti instalacija.

Tabela 1: Linkovi za download tehnologija i alata

Tehnologije i alati	Verzije	Linkovi za download
JDK	1.8.0_251	hhttps://www.oracle.com/java/technologies/javase/javase8u211-later-archive-downloads.html
Maven	3.8.7	https://maven.apache.org/download.cgi
ZooKeeper	3.4.8	https://zookeeper.apache.org
Kafka	2.10-0.10.0.	http://kafka.apache.org/downloads.html
Cassandra	2.6	http://cassandra.apache.org/download/
Spring Boot	1.35.1	https://mvnrepository.com/artifact/org.springframework.boot/spring-boot/1.3.5.RELEASE
Spark	3.2.3	https://spark.apache.org/downloads.html
Python(za Cassandru)	2.7.1	https://www.python.org/downloads/release/python-2718/

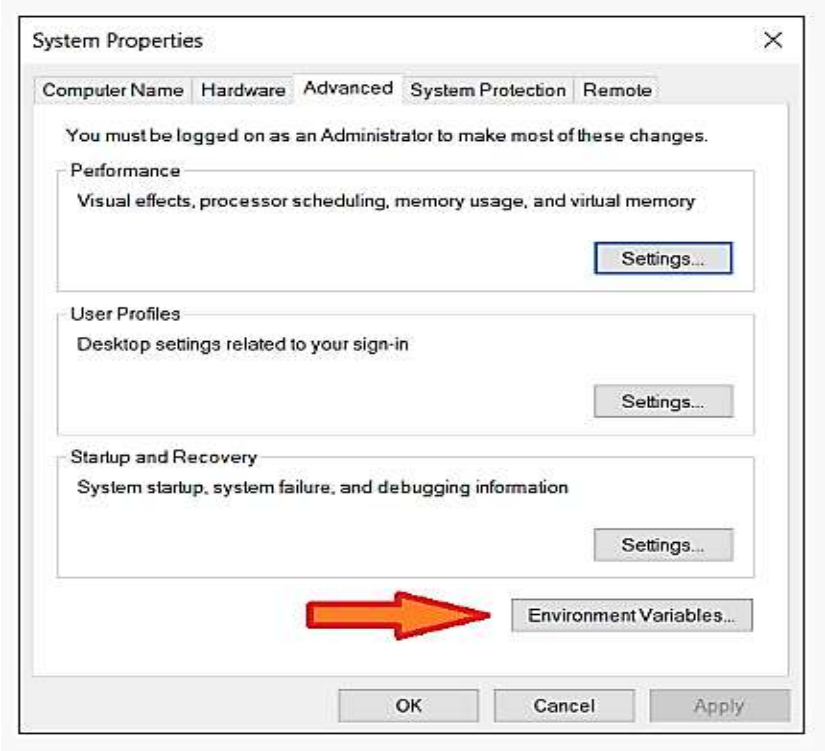
3.1.1. Apache Spark instalacija

Na Windows 10 operativnom sistemu prvo treba preuzeti sa linka (https://spark.apache.org/downloads.html). U ponuđenim rješenjima može se izabrati verzija Sparka, vrstu paketa za koji je Spark napravljen.



Slika 29: Download Apache Spark

Nakon preuzetog Sparka potrebno je ovaj fajl raspakovati i to je najlakše uraditi preko Winrara-a ili 7zip-a. Nakon što je raspakovan sada treba otvoriti System Properties.

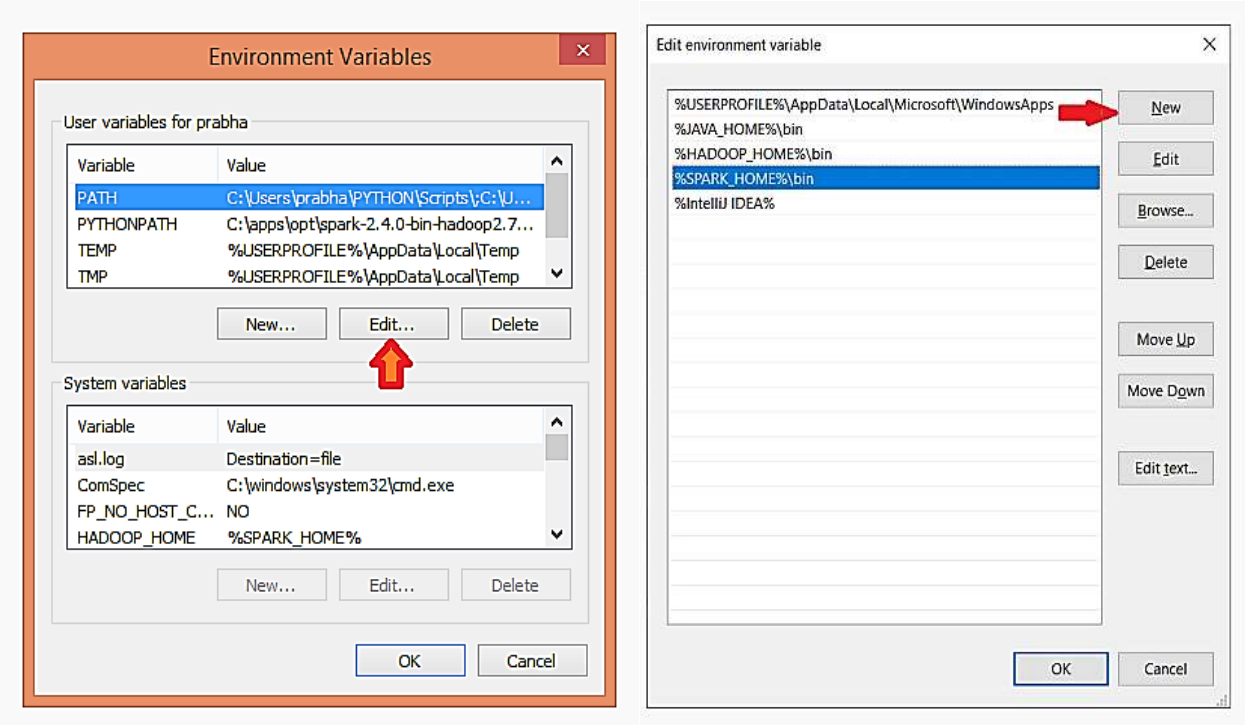


Slika 30: System Properties

Potrebno je kliknuti na strelicom pokazano polje i dodati nove vrijednosti. Ovdje su prikazane putanje do instalacionog foldera na računaru. To su:

```
JAVA_HOME = C:\Program Files\Java\jdk1.8.0_201
SPARK_HOME  = C:\Users\Korisnik\Desktop\spark
HADOOP_HOME = C:\Users\Korisnik\Desktop\spark
```

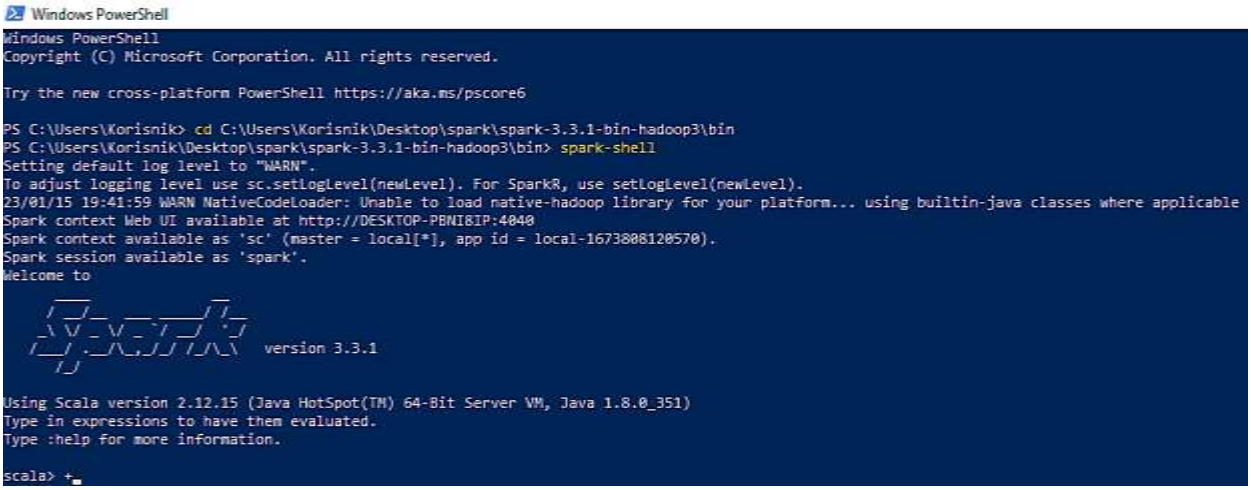
direktorijum će biti drugačiji zavisno od toga gdje su raspakovani fajlovi). Sledeće mora biti izmijenjen Path user varijabla.



Slika 31: Dodavanje na Path (editovanje)

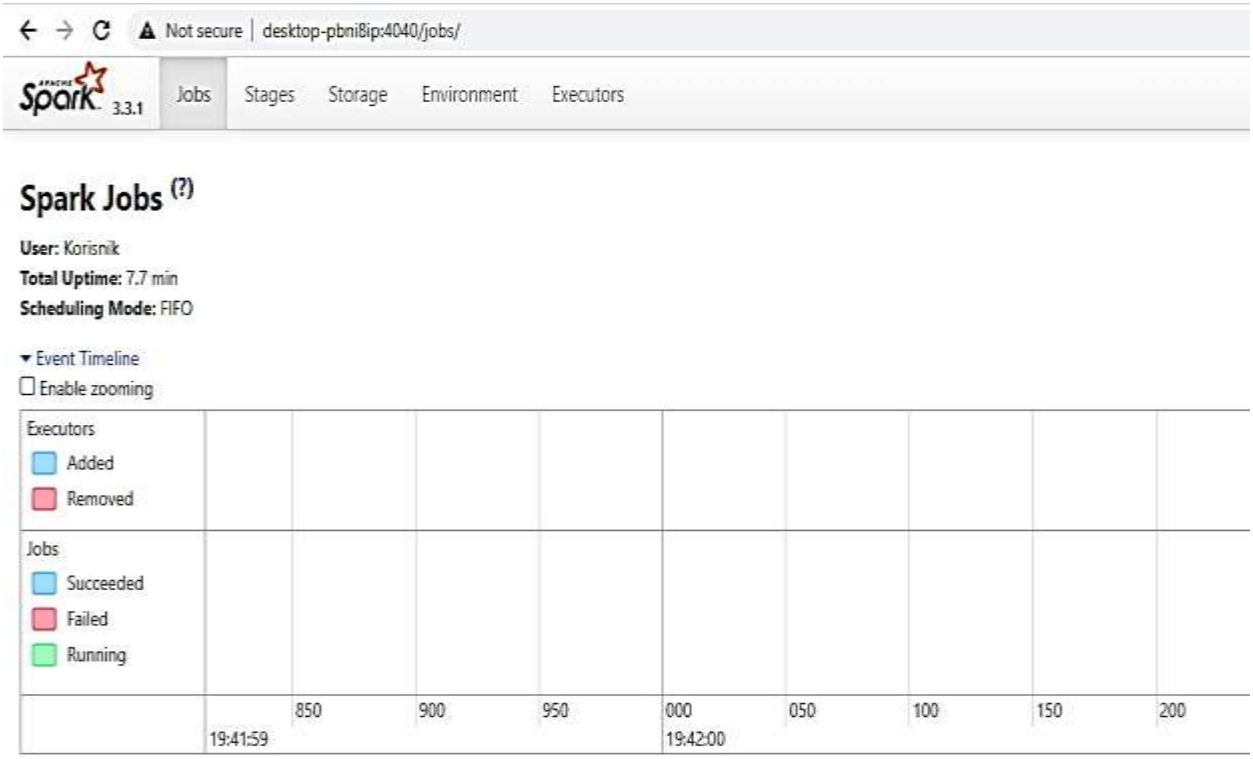
Putanje moraju da budu do raspakovanog fajla %SPARK_HOME%\bin. Sljedeći fajl koji je potrebno preuzeti jeste winutils.exe. Ovaj fajl omogućava korišćenje servisa koji su specifični za Windows i pomaže prilikom izvršavanja Shell komandi u Windows okruženju. Karakterističan je za svaki Hadoop i mora se preuzeti sa web stranice (https://github.com/steveloughran/winutils).

Takođe za instalaciju Sparka potrebna je odgovarajuća verzija Jave i Hadoopa. Sada se treba pozicionirati unutar home foldera preko Command Prompt-a CD %SPARK_HOME%\bin. Komandom spark-shell otvara se Spark.



Slika 32: Power shell nakon spark-shell komande

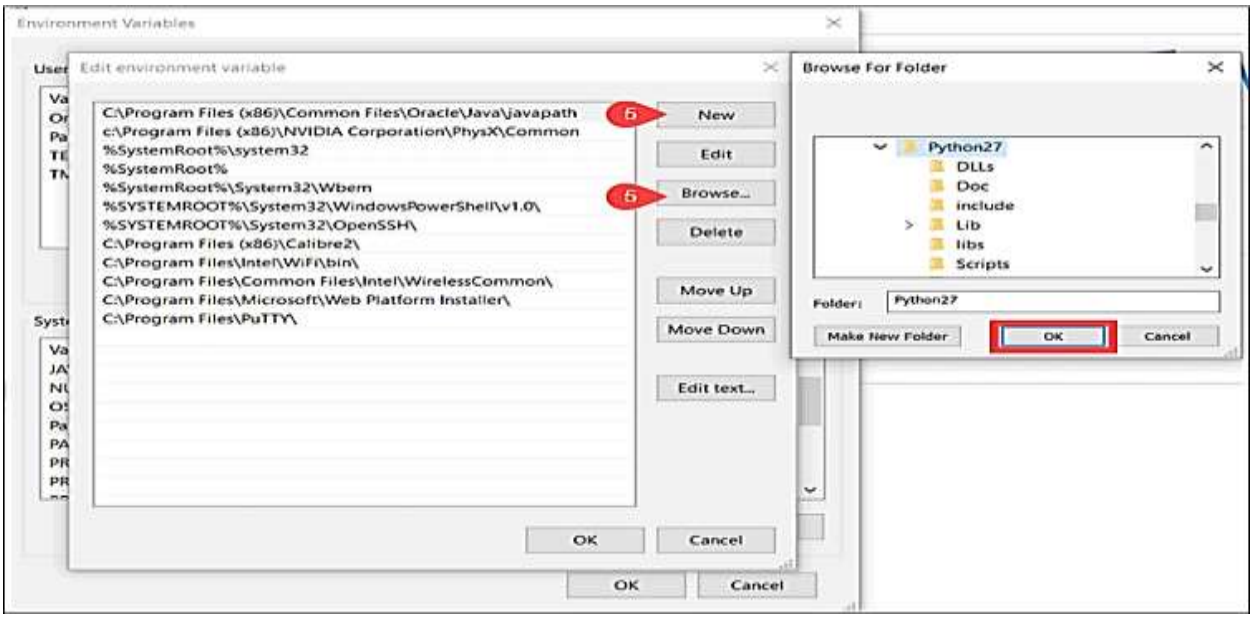
Sada postoji mogućnost za izvršenje nekih zahtjevnijih zadataka i da se koristi moćan prozor alat koji se nalazi na http://localhost:4040.



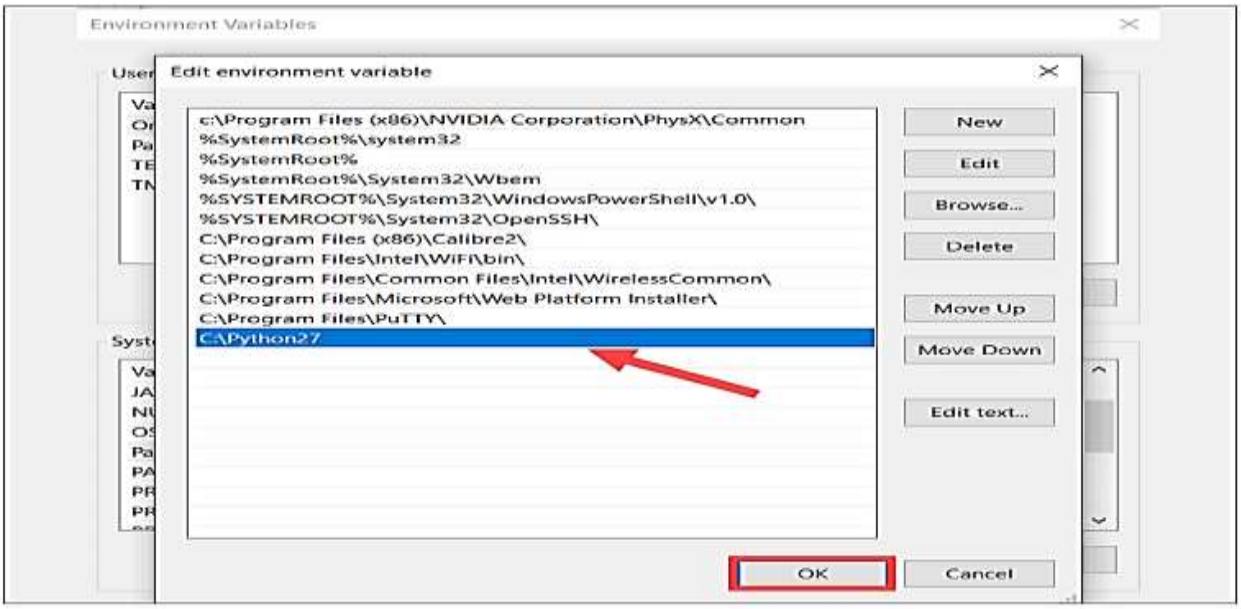
Slika 33: Alat web UI za Spark okruženje

3.1.2. Python, Java, Maven instalacija

Što se Pythona tiče korstili smo verziju Python 2.7.1. Nakon toga treba dodati u Environment varijable i verziju Pythona kao i put do njega. Prilikom svakog dodavanja varijabli veoma je važno potvrditi putanje, jer bez potvrde neće uspješno biti dodata putanja unutar varijabli.

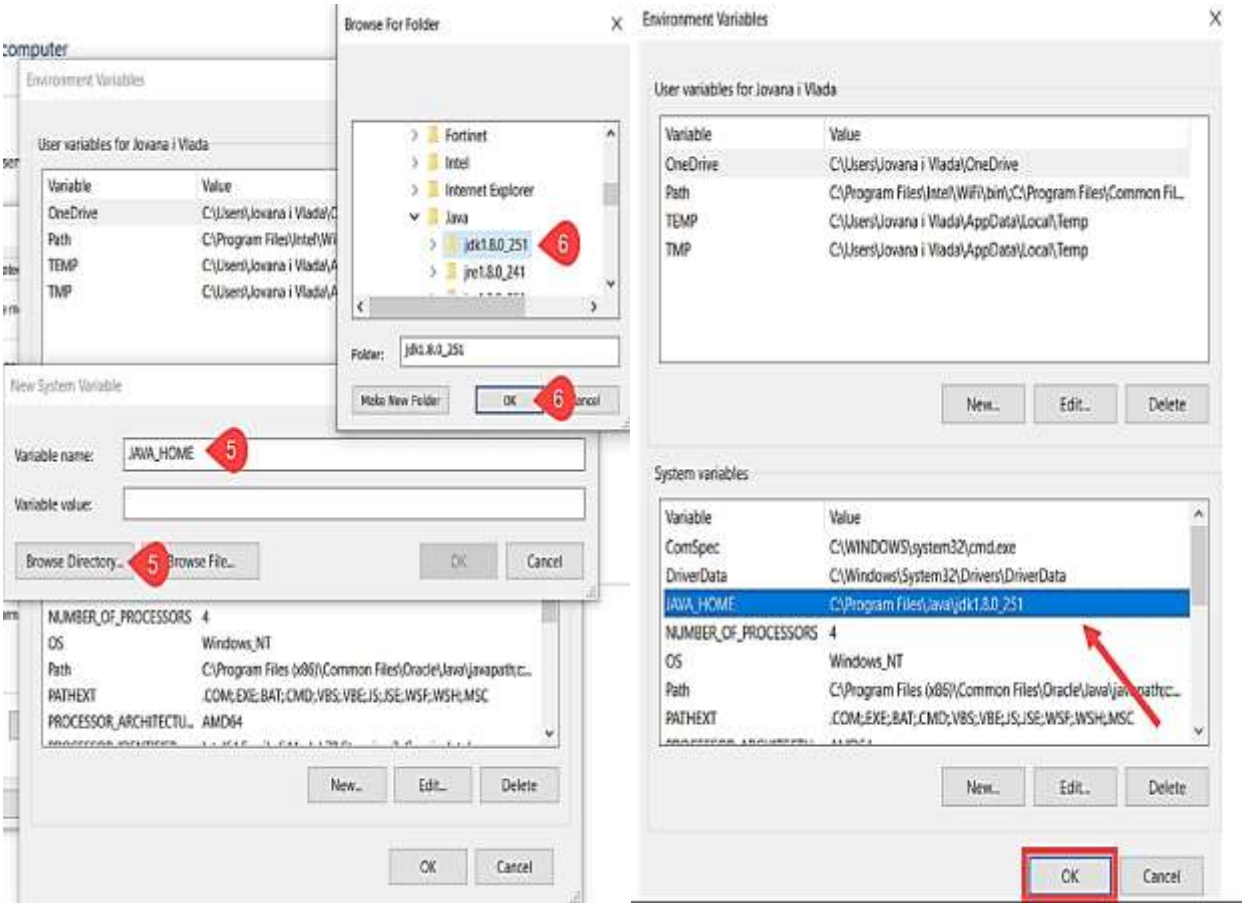


Slika 34: Dodavanje putanje za Python 2.7



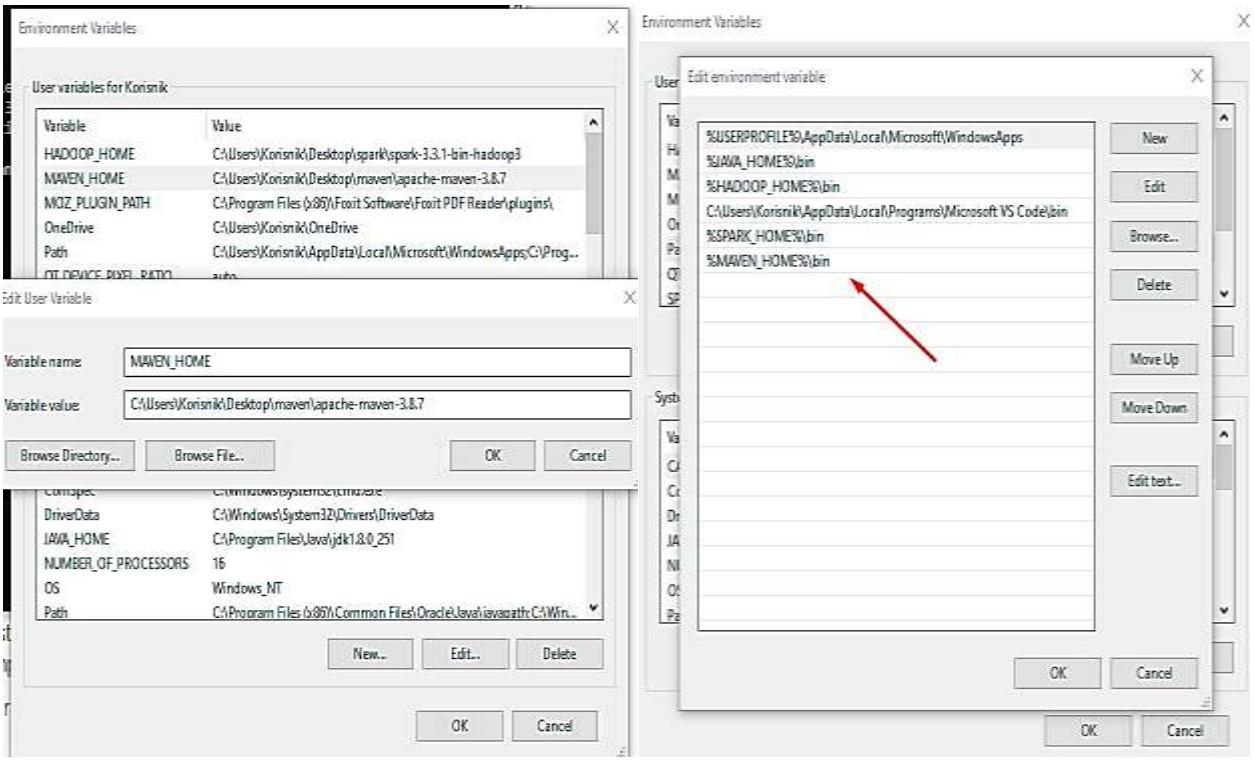
Slika 35: Nova Varijabla na Path-u

Nakon izvršene instalacije dalje je potrebno da se Java instalira. Veoma je važno preuzeti verziju Java SE Development Kit 8u251 koja se nalazi među starijim verzijama, ali nema greške vezane za java-pokretanje (java-runtime). Novije verzije Java SE Development Kit 8u351 će prijavljivati razne greške i neće moći da se pokrene server. Naravno, potrebno je dodati putanju i varijablu isto kao kod Python 2.71.



Slika 36: Dodavanje Java putanje

Maven je veoma lako instalirati. Nakon istalacije da bi se provjerila verzija potrebno je ponovo se pozicionirati unutar Maven/bin instalacionog direktorijuma (u kojem je raspakovan maven). Da bi se mogle globalno koristiti komande iz Maven-a potrebno je dodati u Envirnoment varijable MAVEN_HOME i Path.



Slika 37: Maven Path i Home

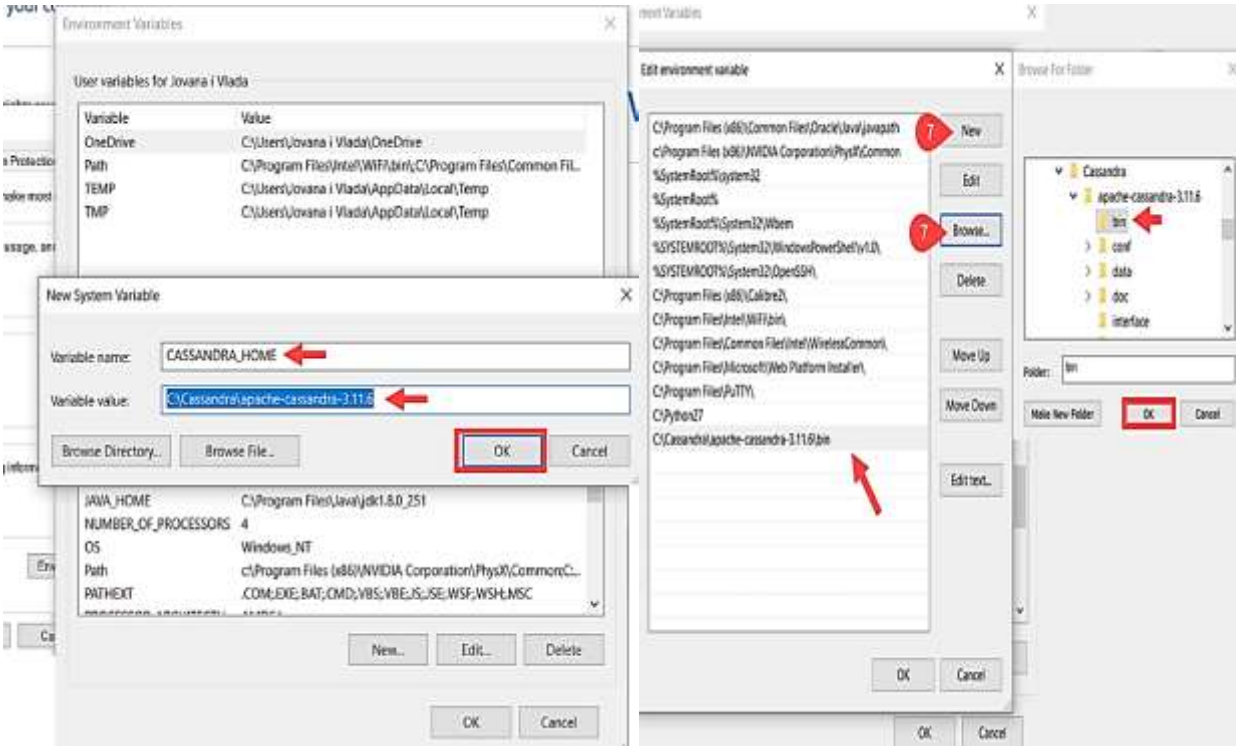
Sada je dalje moguće direktno pristupiti iz CMD-a (Command Prompt) Maven-u.

```
C:\Users\Korisnik\Desktop\maven\apache-maven-3.8.7\bin>mvn -v
Apache Maven 3.8.7 (b89d5959fcde851dcb1c8946a785a163f14e1e29)
Maven home: C:\Users\Korisnik\Desktop\maven\apache-maven-3.8.7
Java version: 1.8.0_251, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_251\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

Slika 38: Maven verzija iz CMDA

3.1.3. Cassandra, Apache Kafka, ZooKeeper instalacija

Na kraju što se Cassandre tiče potrebno je instalirati sa linka iz tabele. Važno je naglasiti da verzija mora da bude 3 i manja od 3. Verzija 4 nema dobru podršku za Windows operativni sistem i ponovo Cassandra neće moći da se koristi. Fajl koji će biti preuzet treba naknadno raspakovati i dodati putanju na Environment.



Slika 39: Dodavnje Path-a i Home-a za Cassandru.

Sada se Cassandra može pokrenuti. U Command Promptu prvo se treba pozicionirati u cassandra/bin folder pa unijeti komandu cassandra.

```
C:\Users\Korisnik>CD C:\Users\Korisnik\Desktop\apache-cassandra-3.11.14\bin

C:\Users\Korisnik\Desktop\apache-cassandra-3.11.14\bin>cassandra
WARNING! Powershell script execution unavailable.
   Please use 'powershell Set-ExecutionPolicy Unrestricted'
   on this user-account to run cassandra with fully featured
   functionality on this platform.
Starting with legacy startup options
Starting Cassandra Server
INFO  [main] 2023-01-18 08:04:45,730 YamlConfigurationLoader.java:93 - Configuration location: file:/C:/Users/Korisnik/Desktop/apache-cassandra-3.
INFO  [main] 2023-01-18 08:04:46,027 Config.java:555 - Node configuration:[allocate_tokens_for_keyspace=null; allow_extra_insecure_udfs=false; all
 auto_bootstrap=true; auto_snapshot=true; back_pressure_enabled=false; back_pressure_strategy=org.apache.cassandra.net.RateBasedBackPressure{high_
old_in_kb=5; batchlog_replay_throttle_in_kb=1024; broadcast_address=null; broadcast_rpc_address=null; buffer_pool_use_heap_if_exhausted=true; cach
pace_check_interval_ms=250; cdc_raw_directory=null; cdc_total_space_in_mb=0; check_for_duplicate_rows_during_compaction=true; check_for_duplicate_
lumn_index_cache_size_in_kb=2; column_index_size_in_kb=64; commit_failure_policy=stop; commitlog_compression=null; commitlog_directory=null; commi
_size_in_mb=32; commitlog_sync=periodic; commitlog_sync_batch_window_in_ms=NaN; commitlog_sync_period_in_ms=10000; commitlog_total_space_in_mb=nul
; concurrent_compactors=null; concurrent_counter_writes=32; concurrent_materialized_view_writes=32; concurrent_reads=32; concurrent_replicates=nul
```

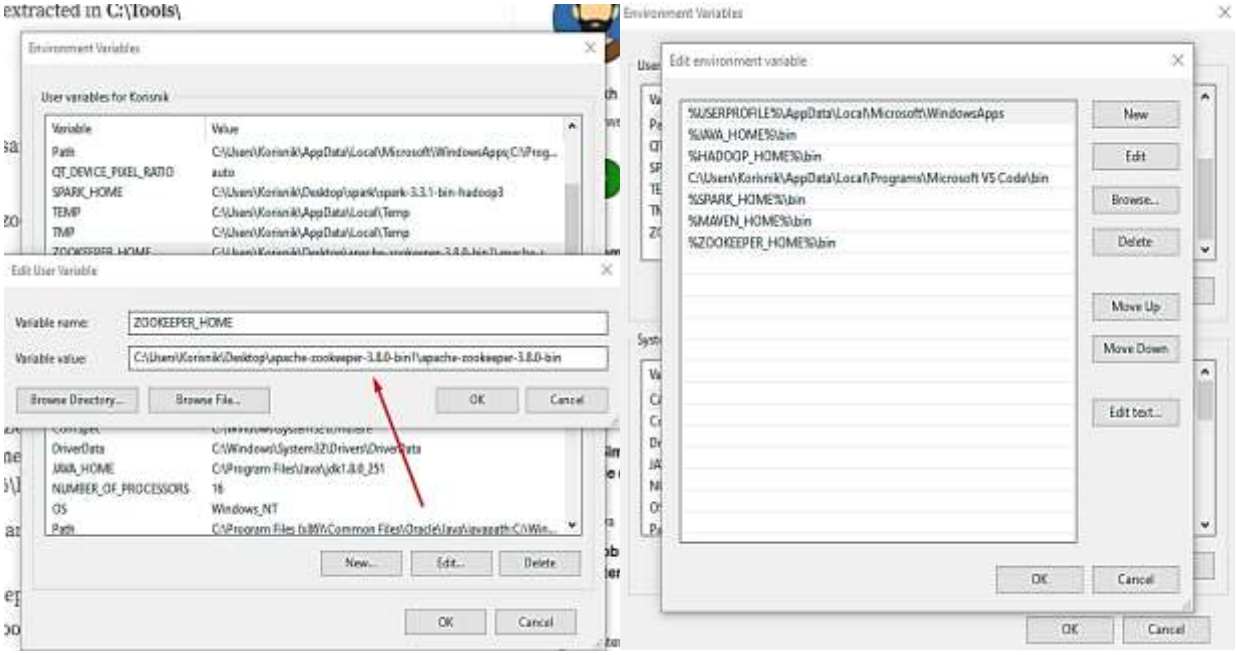
Slika 40: Pokretanje Cassandre unutar CMD-a

Ne treba se CMD prozor zatvarati već u novom prozoru, prvo se treba pozicionirati na isti folder i ukucati komandu cqlsh. Cassandra je spremna za rad.

```
Microsoft Windows [Version 10.0.19044.2486]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Korisnik>cd C:\Users\Korisnik\Desktop\apache-cassandra-3.11.14\bin

C:\Users\Korisnik\Desktop\apache-cassandra-3.11.14\bin>cqlsh

WARNING: console codepage must be set to cp65001 to support utf-8 encoding on Windows platforms.
If you experience encoding problems, change your console codepage with 'chcp 65001' before starting cqlsh.

Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.14 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing.  Install to enable tab completion.
cqlsh>
```

Slika 41: Cassandra spremna za rad

Po istom principu će biti instaliran i Zookeeper. Naravno zarad globalnog korišćenja mora ponovo biti dodat u Envirnoment varijable Path i Home.



Slika 42: Zookeeper Path i Home

Nakon dodavanja moraju se promijeniti određeni fajlovi unutar samog Zookeeper-a da bi on mogao da funkcioniše. Unutar foldera sa instalacijom potrebno je promijeniti fajl %ZOOKEEPER_HOME%/conf/zoo_sample.cfg u zoo.cfg a njegov sadržaj na sledeći način izmijeniti. Prvo kreirati data folder unutar direktorijuma %ZOOKEEPER_HOME% pa izmijeniti

dataDir liniju u konfiguracionom fajlu na sledeći način.

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=C:\Users\Korisnik\Desktop\apache-zookeeper-3.8.0-bin1\apache-zookeeper-3.8.0-bin\data
# the port at which the clients will connect
clientPort=2181
# the maximum number of client connections.
```

Slika 43: Izmjena Zookeeper zoo.conf fajla

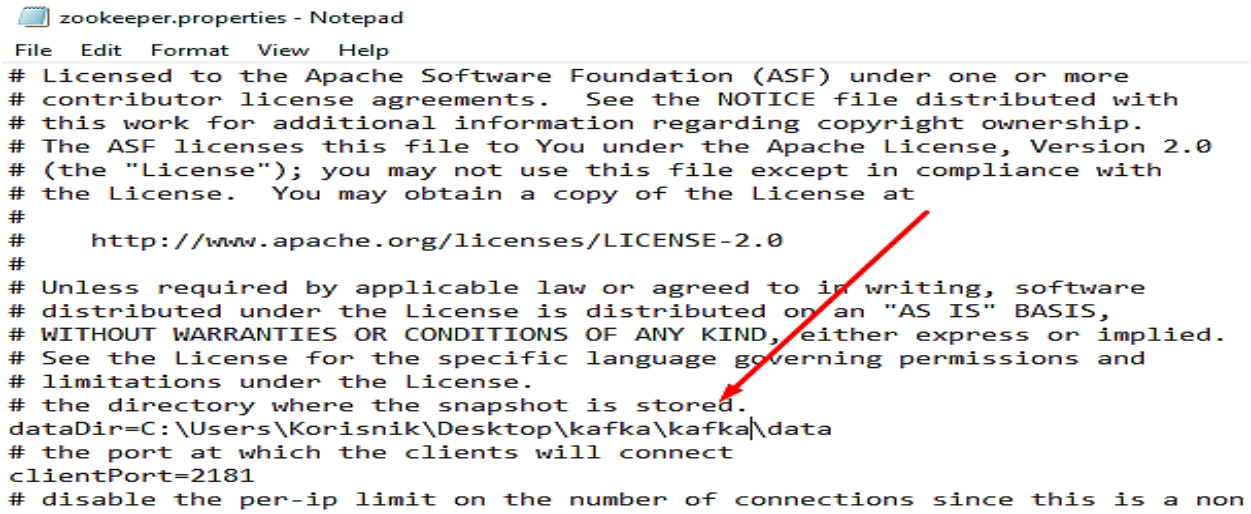
Sada je moguće na CMD-u pokrenuti Zookeeper.

Slika 44: Zookeeper pokrenut u CMD-u

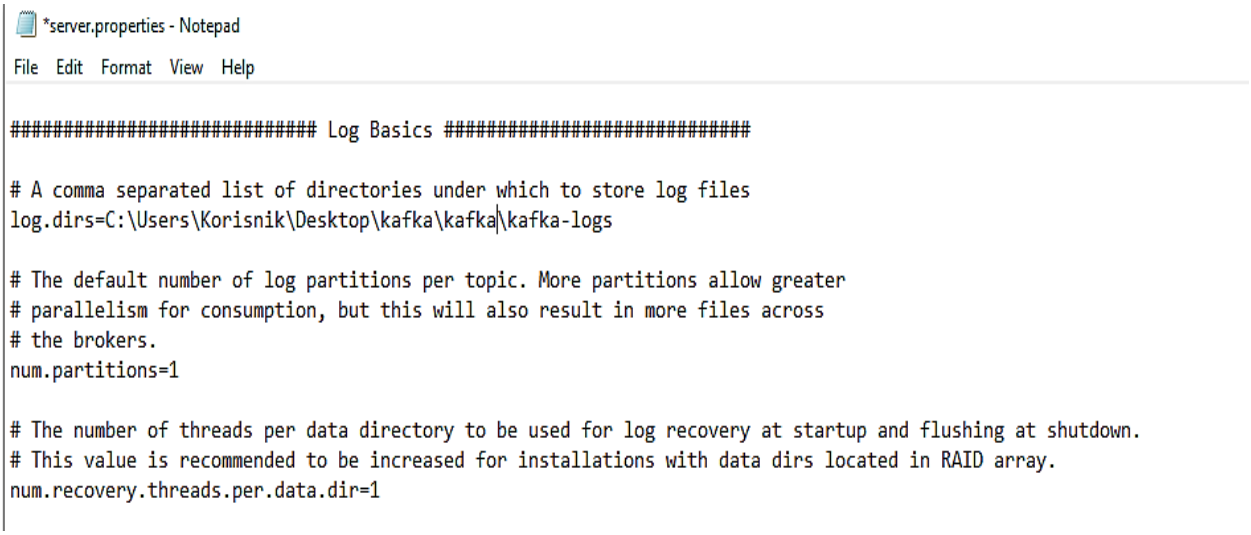
Instalacija Kafke je malo kompleksnija. Treba preuzeti verziju i raspakovati je. Može se desiti da verzija sadrži povlaku u imenu. U tom slučaju samo treba promijeniti ime foldera u kome se nalazi Kafka. Ime koje se može dodijeliti folderu je

Slika 45: Kafka ime foldera

Poslije raspakivanja potrebno je da se kreiraju dva foldera data i logs koje treba dalje prosijediti u konfiguracione fajlove. Ti fajlovi se nalaze u direktorijumu na putanji %KAFKA_HOME%/conf/zookeeper.properties i serverski dio koji se nalazi u folderu na putanji %KAFKA_HOME%/conf/server.properties.



Slika 46: Postavljanje podataka vezanih za Zookeper



Slika 47: Postavljanje podataka vezanih za server

Komandom zookeeper-server-start.bat sa komandom zookeeper.properties pokrenuće se prvo Zookeeper i sada se ne treba zatvarati CMD već ostaviti da radi u pozadini.

Slika 48: Pokretanje Zookepera unutar Kafka alata

Sada još treba izvršiti jednu komandu u novom CMD prozoru. To je kafka-server-start.bat komanda sa Kafka server.properties konfiguracionim fajlom. Rezultat je sljedeći:

Slika 49 : Pokretanje Kafka prozora

Još jednom važno je naglasiti da ime kafka fodera mora biti bez donje crte. Ta donja crta sprečava Windows da podigne Zookeeper i Kafku jer su argumenti putanje predugački.

Nakon instalacije potrebno je u Cassandra alatu kreirati bazu podataka. Prvo je potrebno pokrenuti Cassandra okruženje i u drugom prozoru ukucati komandu cqlsh a sama baza podataka će biti:

```
//Kreiranje keyspace
CREATE KEYSPACE IF NOT EXISTS TrafficKeySpace WITH
 replication = {'class':'SimpleStrategy', 'replication_factor':1};

//Kreiranje tabela
CREATE TABLE TrafficKeySpace.Total_Traffic (routeId text ,  vehicleType text,
totalCount bigint, timeStamp timestamp,recordDate text,PRIMARY KEY
(routeId,recordDate,vehicleType));

CREATE TABLE TrafficKeySpace.Window_Traffic (routeId text , vehicleType text,
totalCount bigint, timeStamp timestamp,recordDate text,PRIMARY KEY
(routeId,recordDate,vehicleType));

CREATE TABLE TrafficKeySpace.Poi_Traffic(vehicleid text , vehicletype text , distance
bigint,  timeStamp timestamp,PRIMARY KEY (vehicleid));

//Selekcija tabela
SELECT * FROM TrafficKeySpace.Total_Traffic;
SELECT * FROM TrafficKeySpace.Window_Traffic;
SELECT * FROM TrafficKeySpace.Poi_Traffic;

//Brisanje sadržaja tabela
TRUNCATE TABLE TrafficKeySpace.Total_Traffic;
TRUNCATE TABLE TrafficKeySpace.Window_Traffic;
TRUNCATE TABLE TrafficKeySpace.Poi_Traffic;
```

Slika 50: Baza podataka

Potrebno je još napraviti temu u alatu Kafka i to se radi na sledeći način.

```
C:\kafka\bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --replication-
factor 1 --partitions 1 --topic iot-data-event
```

Slika 51 : IoT event kreiran preko Kafka alata

To je sve što se tiče instalacionog dijela tehnologija. Dalje će biti više govora o samom projektu. Kao što je prije naglašeno sam projekat je podijeljen na tri zasebne cjeline i prikaz preko Google Maps API, svaka sa svojim posebnostima i zadacima.

4. Razvoj web platforme

Za razvoj web platforme potrebno je implementirati sistem za prikupljanje i analizu podataka o saobraćaju. Ovo uključuje instaliranje IoT uređaja (kao što su kamere i senzori) na različitim lokacijama širom grada radi prikupljanja podataka o obimu saobraćaja, brzini i zagušenostima. Podaci se zatim šalju Apache Kafki, distribuiranoj platformi za striming, koja deluje kao centralno čvorište za obradu i skladištenje podataka.

Mogućnost korišćenja Apache Spark Streaming za analizu podataka u realnom vremenu. Spark Streaming omogućava korisnicima da pišu kod za obradu strima na isti način na koji bi pisali kod za grupnu obradu, ali sa dodatnom mogućnošću obrade podataka u realnom vremenu. Spark Streaming se koristi za izračunavanje metrika kao što su prosječna brzina, obim saobraćaja i nivoi zagušenja, koji se mogu koristiti za donošenje odluka o upravljanju saobraćajem.

Za razvoj ove web platforme koristiće se 3 modula i dodatak karte:

- IoT Producer
- IoT Data Processor
- IoT Data DashBoard
- Google Maps API

4.1. IoT Producer

IoT uređaji imaju mogućnost da primaju velike količine podataka koje generišu vozila. Samim tim podaci koji su generisani će biti vremenski senzitivni i količinski će ih biti veoma mnogo. Podaci dobijeni preko poruka će biti prihvaćeni, filtrirani i ponovo navedeni na stream kao tok podataka. Za generisanje velikog broja podataka će biti korišćen alat Kafka kao IoT Producer.

Maven sam po sebi sadrži Pom.xml fajl za svaki projekat. Ovaj fajl je veoma značajan zbog toga što on u sebi sadrži sve opise konfiguracije vezane za projekat. Takođe kroz svaku zavisnost (dependency) Pom.xml fajl će da opisuje svaki alat, verziju alata koji je korišćen.

```xml
<project xmlns="http://maven.apache.org/POM/4.0"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.iot.app.kafka</groupId>
    <artifactId>iot-kafka-producer</artifactId>
    <version>1.0.0</version>
    <name>IoT Kafka Producer</name>
    <dependencies>
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka_2.10</artifactId>
            <version>0.8.1</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-core</artifactId>
            <version>2.6.6</version>
         </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.6.6</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-annotations</artifactId>
            <version>2.6.6</version>
```

Slika 52: pom.xml fajl sa svim tehnologijama

Kao što je u samom Pom.xml fajlu vidno iako imamo noviju verziju Kafka alata on će u dependency dijelu da koristi stariju verziju ali to neće štetiti samom programu i on će dalje da se izvršava. Primjetne su sve verzije tehnologija koje su korišćene.

Kao format poruka biće primijećen jason format koji će biti poslat preko posebne klase. Ta klasa sadrži string vrijednosti String vehicleId (id vozila), String vehicleType (tip vozila), String routeId (id puta), String longitude (geografska širina), String latitude (geografska dužina).

Na sledećoj slici će biti prikazana sama klasa koja definiše vozila.

```java
public class IoTData implements Serializable{
    private String vehicleId;
    private String vehicleType;
    private String routeId;
    private String longitude;
    private String latitude; }
```

Slika 53: Klasa IoTData

Još će biti definisan i konstruktor na samu klasu. Gdje se može pristupiti svim

vrijednostima .

```
public IoTData(String vehicleId, String vehicleType, String routeId, String
latitude, String longitude,
        Date timestamp, double speed, double fuelLevel) {
        super();
        this.vehicleId = vehicleId;
        this.vehicleType = vehicleType;
        this.routeId = routeId;
        this.longitude = longitude;
        this.latitude = latitude;
        this.timestamp = timestamp;
        this.speed = speed;
        this.fuelLevel = fuelLevel; }
    public String getVehicleId() {
        return vehicleId; }
    public String getVehicleType() {
         return vehicleType;
}
```

Slika 54: IoTData konstruktor sa svim vrijednostima

Pored osnovne klase tu su još IoTDataProducer i IoTDataEncoder klasa. IoTDataProducer klasa će kreirati podatke u JSON formatu. Mora se napraviti specijalna Kafka klasa za serijalizaciju koja će serijalizovati IoT objekte.

```
public class IoTDataEncoder implements Encoder<IoTData> {
      public byte[] toBytes(IoTData iotEvent) {
        try {
        String msg = objectMapper.writeValueAsString(iotEvent);
        return msg.getBytes();
      } catch (JsonProcessingException e) {}
 }
 }
```

Slika 55: IoTDataEncoder

Kao specijalni metod koristiće se *toBytes()* iz kafka.serializer.Encoder interfejsa. Poslednja klasa koju sadrži prvi Maven projekat je IoTDataProducer klasa. U njoj se određeni elementi i resursi čuvaju u resource folderu projekta. Na samom početku kreirana je tema u Kafka alatu. IoTDataProducer će kreirati poruke vezane za samu temu.

```
public class IoTDataProducer {
Properties properties = new Properties();
        properties.put("zookeeper.connect", zookeeper);
        properties.put("metadata.broker.list", brokerList);
        properties.put("request.required.acks", "1");
        properties.put("serializer.class","com.iot.app.kafka.util.IoTDataEncoder";

Producer<String, IoTData> producer = new Producer<String, IoTData>(new
ProducerConfig(properties));
        IoTDataProducer iotProducer = new IoTDataProducer();
        iotProducer.generateIoTEvent(producer,topic); }
```

Slika 56: IoTDataProducer

Dalje su ostali prikazi pojedinih interesantnih funkcija. To su metodi kojima se generišu slučajni putevi kao i širina i dužina koji će kasnije biti proslijeđeni kao podaci.

```
//Metode za generisanje slučajnih puteva i dužine i širine koje se prosleđuju
    private String  getCoordinates(String routeId)   {
        Random rand = new Random();
        int latPrefix = 0;
        int longPrefix = -0;
        if (routeId.equals("Route-37")) {
            latPrefix = 32;
            longPrefix = -95;
        } else if (routeId.equals("Route-43")) {
            latPrefix = 33;
            longPrefix = -96;
        } else if (routeId.equals("Route-44")) {
            latPrefix = 34;
            longPrefix = -97;
        }  else if (routeId.equals("Route-69")) {
            latPrefix = 35;
            longPrefix = -98;
        } else if (routeId.equals("Route-82")) {
            latPrefix = 36;
            longPrefix = -99; }
        Float lati = latPrefix + rand.nextFloat();
        Float longi = longPrefix + rand.nextFloat();
        return lati + "," + longi; }
```

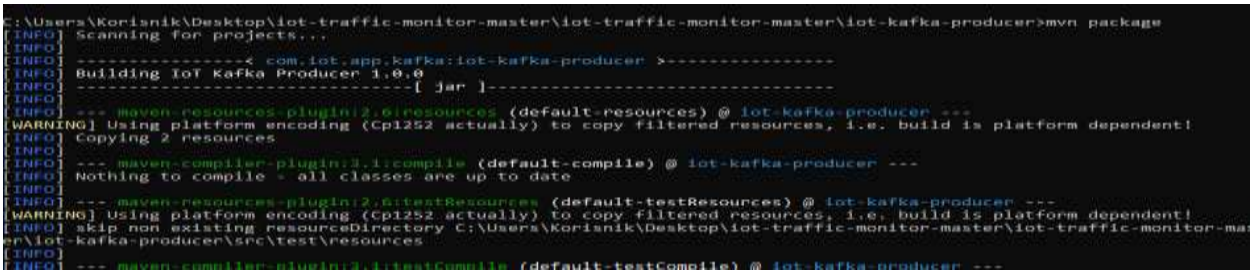
Slika 57: Generisanje puta, geografske širine i dužine

```
        private void generateIoTEvent(Producer<String, IoTData> producer, String topic) throws
        InterruptedException {
        List<String> routeList = Arrays.asList(new String[]{"Route-37", "Route-43", "Route-44",
"Route-69", "Route-82"});
        List<String> vehicleTypeList = Arrays.asList(new String[]{"Veliki kamion", "Mali kamion",
"Auto", "Autobus", "Taxi"});
        Random rand = new Random();
         while (true) {  // generiši događaj unutar petlje
          List<IoTData> eventList = new ArrayList<IoTData>();
         for (int i = 0; i < 200; i++) {// kreiraj 200 vozila
           String vehicleId = UUID.randomUUID().toString();
           String vehicleType = vehicleTypeList.get(rand.nextInt(5));
           String routeId = routeList.get(rand.nextInt(5));
           double speed = rand.nextInt(100 - 20) + 20;// slučajna brzina između 20 i 100
              double fuelLevel = rand.nextInt(40 - 10) + 10;
              for (int j = 0; j < 5; j++) {// dodaj 5 događaja za svako vozilo
                String coords = getCoordinates(routeId);
                String latitude = coords.substring(0, coords.indexOf(","));
                String longitude = coords.substring(coords.indexOf(",") + 1,
                IoTData event = new IoTData(vehicleId, vehicleType, routeId, latitude, longitude,
null, speed, fuelLevel); // generiši podatke
eventList.add(event); }}
     Collections.shuffle(eventList);// nasumice odaberi događaje
     for (IoTData event : eventList) {
       event.setTimestamp(new Date());
KeyedMessage<String, IoTData> data = new KeyedMessage<String,IoTData>(topic, event);
       producer.send(data);
       Thread.sleep(rand.nextInt(1000 - 500) + 500);//slučajno kašnjenje između 0.5 i 1 se}
```

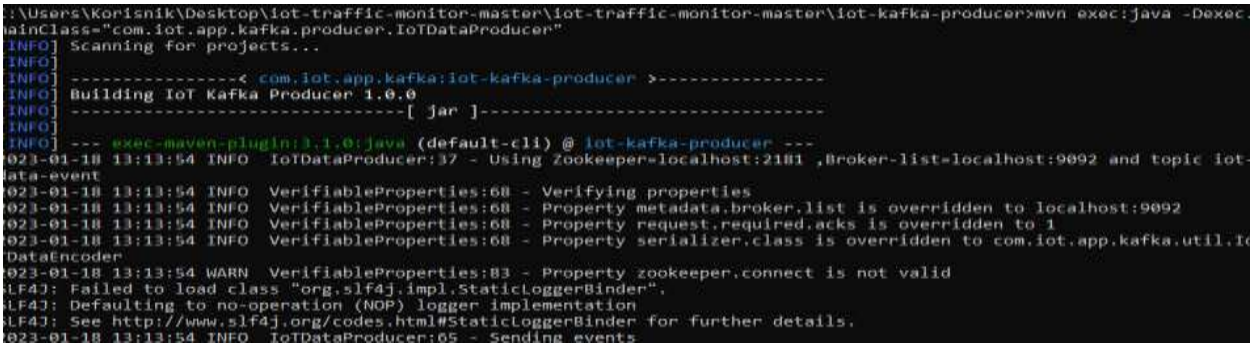
Slika 58: Generisanje nivoa goriva, brzine, vremena i samih vozila.

Koriste se različite putanje. Postoji 5 različitih vozila, a to su kamion (veliki i mali), taksi, autobus i automobil. Vozila šalju event-e u redovnim vremenskim intervalima dok se kreću putem. Svako vozilo će poslati pet događaja nasumičnim redosljedom sa zakašnjenjem od 0.5 do 1 sekunde. Za genereisanje atributa koristi se Java random klasa.

Na kraju treba pokrenuti prvu projektnu cjelinu. To se postiže tako što se pozicionira unutar foldera sa projektom preko CMD-a i iskoristi komanda mvn package. Nakon kompajliranja je potrebno ukucati komandu mvn exec:java -Dexec.mainClass="com.iot.app.kafka.producer.IoTDataProducer".



Slika 59: Pokretanje Mvn paketa



Slika 60: mvn exec:java -Dexec.mainClass="com.iot.app.kafka.producer.IoTDataProducer".

Sada će da se generišu razni fajlovi u vidu streama, konstantno na određenim vremenskim intervalima.

Slika 61: Stream generisan IotDataProducer-om

Naravno ovo generisanje nije moguće bez prethodno uključenih tehnologija. Uključena je Kafka, Zookeeper (Preko kafke) , Cassandra (zarad baze podataka) i na kraju je tek Maven izvršen.

Slika 62: Cmd prozori neophodni za izvršenje IotDataProducer-A

4.2. IoT Data Processor

Sledeći modul koji treba da bude obrađen je IoT Data Processor platforma naravno definisana u Mavenu. Kao i u prvom dijelu prvo će biti prikazan pom.xml fajl sa svim zavisnostima.

```
<project xmlns="http://maven.apache.org/POM/4.0."xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.iot.app.spark</groupId>
    <artifactId>iot-spark-processor</artifactId>
    <version>1.0.0</version>
    <name>IoT Spark Processor</name>
           <dependencies>
        <!-- Spark dependencies -->
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_2.10</artifactId>
            <version>1.6.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-streaming_2.10</artifactId>
            <version>1.6.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-streaming-kafka_2.10</artifactId>
            <version>1.6.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-sql_2.10</artifactId>
            <version>1.6.2</version>
        </dependency>
        <dependency>  <!-- Spark cassandra -->
            <groupId>com.datastax.spark</groupId>
            <artifactId>spark-cassandra-connector_2.10</artifactId>
            <version>1.6.0</version>
        </dependency>
```

Slika 63: pom.xml. Data Producer Maven platforme

Primjetno će biti da pored standardnih Apache Spark i Kafka alata tu je još i konektor (connector) za Cassandru koji će da transformiše određene podatke u korisne i smješta ih u Cassandra bazu podataka.

U prošloj Maven platformi je izvršena serijalizacija podataka da bi se napravio stream koji će simulirati priliv podataka u realnom sistemu. Sada treba izvršiti postupak deserijalizacije odnosno od JSON stringa će da nastane IoT Data objekat. To se vrši pomoću klase IoTDataDecoder koja ima sledeću sintaksu. Sada će postojati komplementaran metod *fromBytes*.

```
public class IoTDataDecoder implements Decoder<IoTData> {
   private static ObjectMapper objectMapper = new ObjectMapper();
   public IoTDataDecoder(VerifiableProperties verifiableProperties) {}
   public IoTData fromBytes(byte[] bytes) {
      try {
      return objectMapper.readValue(bytes, IoTData.class);
    } catch (Exception e) {
     e.printStackTrace();}
     return null;         }                              }
```

Slika 64: Spark FromBytes() metod

U resource folderu će biti IoT-spark.properties koji će da upari konfiguracije Kafka alata, Spark alata i Cassandra alata. Cijeli IoTDataProcessor će biti ispisan koristeći Spark API-je. Dalje treba postaviti Cassandra i Spark osobine. Kreira se novi objekat SparkConf koji će da povezuje Cassandra i Spark alate.

```
 public static void main(String[] args) throws Exception {
         //čita Spark i Cassandra properties i kreira SparkConf
         Properties prop = PropertyFileReader.readPropertyFile();
         SparkConf conf = new SparkConf()
                 .setAppName(prop.getProperty("com.iot.app.spark.app.name"))
                 .setMaster(prop.getProperty("com.iot.app.spark.master"))
                 .set("spark.cassandra.connection.host",
prop.getProperty("com.iot.app.cassandra.host"))
                 .set("spark.cassandra.connection.port",
prop.getProperty("com.iot.app.cassandra.port"))
                 .set("spark.cassandra.connection.keep_alive_ms",
prop.getProperty("com.iot.app.cassandra.keep_alive"));
         //interval batch/serija od 5 sekundi za dolazeći stream
         JavaStreamingContext jssc = new JavaStreamingContext(conf,
Durations.seconds(5));
          //kreira kontrolnu tačku/checkpoint
          jssc.checkpoint(prop.getProperty("com.iot.app.spark.checkpoint.dir"))
;
```

Slika 65: SparkConf()

Platforma će da preuzima fajlove sa streama svakih 5 sekundi. Ti podaci se dalje mogu pročitati preko *KafkaUtils.createDirectStream()*.

```
//čita i podešava Kafka properties
        Map<String, String> kafkaParams = new HashMap<String, String>();
        kafkaParams.put("zookeeper.connect",
prop.getProperty("com.iot.app.kafka.zookeeper"));
prop.getProperty("com.iot.app.kafka.brokerlist"));
        String topic = prop.getProperty("com.iot.app.kafka.topic");
        Set<String> topicsSet = new HashSet<String>();
        topicsSet.add(topic);
        //kreira direct kafka stream
        JavaPairInputDStream<String, IoTData> directKafkaStream =
KafkaUtils.createDirectStream(
            jssc,
            String.class,
            IoTData.class,
            StringDecoder.class,
            IoTDataDecoder.class,
            kafkaParams,
            topicsSet        );
logger.info("Starting Stream Processing");
```

Slika 66: Čitanje Kafka osobina kao i postavljanje Kafka streama

Prvo će se izvršiti transformacija koristeći *map* operacije da bi bio dobijen Dstream od IoTData objekata. Par od Dstream objekta kod koga će ključ da bude vehicleId a vrijednost će da bude sami IoTData objekat i za ovo se koristi *mapToPair* transformacija. Sada za svako vozilo se može desiti više događaja i za to koristimo funkciju *ReduceByKey*.

```
//koristimo nefiltrirani tok za izračunavanje podataka za poi traffic
JavaDStream<IoTData> nonFilteredIotDataStream = directKafkaStream.map(tuple -> tuple._2());
//koristimo filtrirani tok za izračunavanje total and traffic data
JavaPairDStream<String,IoTData> iotDataPairStream = nonFilteredIotDataStream.mapToPair(iot
-> new             Tuple2<String,IoTData>(iot.getVehicleId(),iot)).reduceByKey((a, b) -> a
);
// provjeri vehicle Id da li je procesuiran
JavaMapWithStateDStream<String, IoTData, Boolean, Tuple2<IoTData,Boolean>>
iotDStreamWithStatePairs = iotDataPairStream
.mapWithState(StateSpec.function(processedVehicleFunc).timeout(Durations.seconds(3600)));//
zadrži stanje jedan sat
// filtriraj procesuirane podatke i ostavi neobrađene
JavaDStream<Tuple2<IoTData,Boolean>> filteredIotDStreams =
iotDStreamWithStatePairs.map(tuple2 -> tuple2)
.filter(tuple -> tuple._2.equals(Boolean.FALSE));
// Preuzmi stream od IoTdata
```

```
        JavaDStream<IoTData> filteredIotDataStream =
filteredIotDStreams.map(tuple -> tuple._1);
        //keširaj stream kako je korišćen u proračunu total i window
        filteredIotDataStream.cache();
        //process data
        IoTTrafficDataProcessor iotTrafficProcessor = new
IoTTrafficDataProcessor();
        iotTrafficProcessor.processTotalTrafficData(filteredIotDataStream);
        iotTrafficProcessor.processWindowTrafficData(filteredIotDataStream);
```

Slika 67: mapToPair transformacija

Da bi bio izračunat broj vozila tokom vremena, potrebno je voditi evidenciju o vozilima koja su već obrađena u prethodnim Dstreamovima. Da bi to bilo postignuto, koristiće se Spark-ova operacija stanja. Koristi operacija mapWithState dostupna za sve Dstream parove ključ-vrijednosti. MapWithState operacija koristi *StateSpec.function* za ovaj dio platforme.

```
//Funkcija za provjeru procesuiranih vozila.
    private static final Function3<String, Optional<IoTData>, State<Boolean>,
Tuple2<IoTData,Boolean>> processedVehicleFunc = (String, iot, state) -> {
            Tuple2<IoTData,Boolean> vehicle = new Tuple2<>(iot.get(),false);
            if(state.exists()){
                vehicle = new Tuple2<>(iot.get(),true);
            }else{
                state.update(Boolean.TRUE);
            }
            return vehicle;
        };
```

Slika 68: Evidencija o obrađenim vozilima

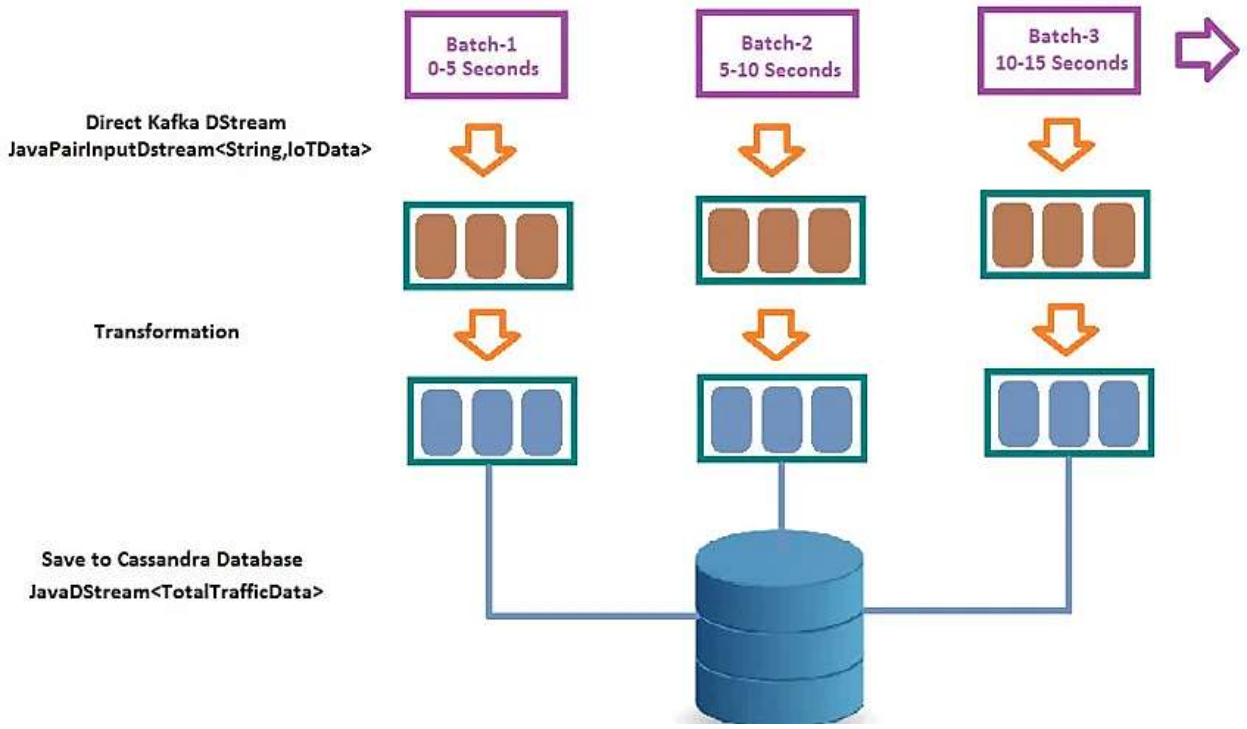
Na kraju što se tiče samih operacija stanja Spark će čuvati RDD stanja na svim mašinama klastera. Ako se platforma dugo koristi može doći do prevelike memorije korišćene na worker čvorovima (nodovima) mašina. Zbog toga treba na neki način da se filtrira stream podataka. To se radi preko dvije operacije koje su već prikazane na slici 66. U suštini vrijeme treba ograničiti za koje će biti procesuirani podaci. Ovdje u funkciji to vrijeme je ograničeno na 3600 sekundi (1 sat).

Sve do sada prikazane funkcije IoTDataProcessor dijela platforme su sami process dio u fajlu IoTDataProcessor.java.

4.3. Ukupan procesuirani saobraćaj

Svi procesuirani podaci sa IoTData streama će biti podijeljeni u tri grupe u zavisnosti od tipa vozila na svakom putu. Vozila će biti identifikovana na osnovu routeId i vehicleType tako da je kreirana klasa AggregateKey koja ima ova dva atributa.

U Kafka strimingu, "batch" predstavlja kolekciju zapisa koji se procesuiraju zajedno. Sesije su posebno važne u platformama za obradu toka koje zahtijevaju obradu stanja, gdje se zapisi moraju grupisati i obraditi zajedno da bi se održalo stanje platforme. U ovim slučajevima, grupe se često koriste za agregiranje podataka u vremenskom prozoru ili ključu i za ažuriranje stanja platforme na osnovu sadržaja serije.



Slika 69: Podjela svih vozila u grupe

AggregateKey objekat će ponovo da bude ključ u paru Dstream koji je prije u IoTDataProcessor.java fajlu bolje objašnjen. Sada će u paru Dstream da se broje vozila. Ponovo će biti korišćena mapToPair transformacija stim da sada se koristi za broj vozila. Na kraju će biti izvršena funkcija ReduceByKey da bi se kombinovao broj istih AggregateKey-eva. Na kraju se mora održavati broj vozila i podaci o saobraćaju.

```
// Moramo dobiti broj grupa vozila prema routeId i vehicleType
 JavaPairDStream<AggregateKey, Long> countDStreamPair = filteredIotDataStream
 .mapToPair(iot -> new Tuple2<>(new AggregateKey(iot.getRouteId(),
iot.getVehicleType()), 1L))
 .reduceByKey((a, b) -> a + b);

// Potrebno je zadržati stanje za ukupan broj
 JavaMapWithStateDStream<AggregateKey, Long, Long, Tuple2<AggregateKey, Long>>
countDStreamWithStatePair = countDStreamPair
 .mapWithState(StateSpec.function(totalSumFunc).timeout(Durations.seconds(3600)));
//zadrži stanje jedan sat
//Transformiši u Dstream TrafficData
 JavaDStream<Tuple2<AggregateKey, Long>> countDStream =
countDStreamWithStatePair.map(tuple2 -> tuple2);
 JavaDStream<TotalTrafficData> trafficDStream =
countDStream.map(totalTrafficDataFunc);
```

Slika 70: AggregateKey sa transformacijom dstream-a

Nizom transformacija će biti dobijen routeId, vehicleType i ukupan broj, što čini korisne podatke u saobraćaju. Ti podaci sada treba da budu poslati u Cassandra bazu podataka na čuvanje da bi treća Maven platforma Dashboard mogla da ih efikasno koristi.

Na sledećoj slici je dalje prikazano kako se podaci čuvaju u Cassandra bazi podataka. Ponovo se vrši mapiranje i sada sa metodom put ubacuju se pojedini podaci.

```
// Mapa Cassandra tabele kolone
        Map<String, String> columnNameMappings = new HashMap<String, String>();
        columnNameMappings.put("vehicleId", "vehicleid");
        columnNameMappings.put("distance", "distance");
        columnNameMappings.put("vehicleType", "vehicletype");
        columnNameMappings.put("timeStamp", "timestamp");

// poziv CassandraStreamingJavaUtil funkcije za cuvanje u bazu
       javaFunctions(trafficDStream)
       .writerBuilder("traffickeyspace",
"poi_traffic",CassandraJavaUtil.mapToRow(POITrafficData.class,
columnNameMappings))
       .withConstantTTL(120)//cuvanje podataka 2 minuta
       .saveToCassandra();
```

Slika 71:Veza sa Cassandra bazom podataka

Prije čuvanja podaci se moraju na neki način transformisati od prethodne transformacije u JavaDStream<TotalTrafficData > koji je pogodan za Cassandra bazu podataka.

```
//Funkcija za kreiranje TotalTrafficData objekta od IoT data
    private static final Function<Tuple2<AggregateKey, Long>, TotalTrafficData>
totalTrafficDataFunc = (tuple -> {

logger.debug("Total Count : " + "key " + tuple._1().getRouteId() + "-" +
tuple._1().getVehicleType() + " value "+ tuple._2());

        TotalTrafficData trafficData = new TotalTrafficData();
        trafficData.setRouteId(tuple._1().getRouteId());
        trafficData.setVehicleType(tuple._1().getVehicleType());
        trafficData.setTotalCount(tuple._2());
        trafficData.setTimeStamp(new Date());

        trafficData.setRecordDate(new SimpleDateFormat("yyyy-MM-dd").format(new
Date()));

        return trafficData;
    });
```

Slika 72: TotalTrafficData objekat

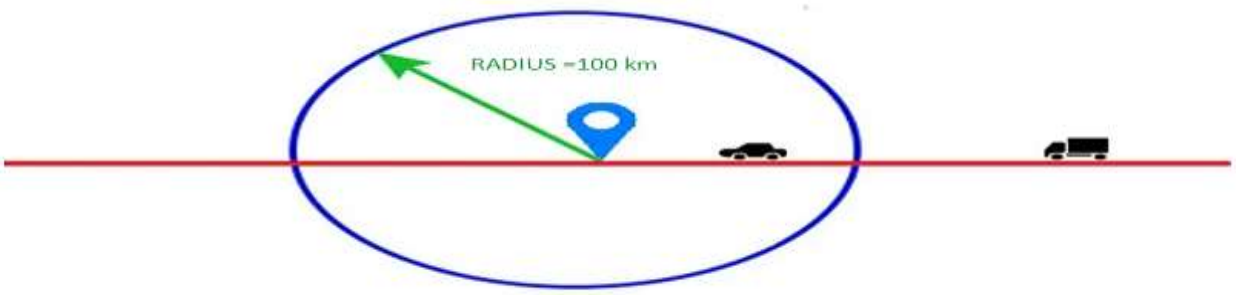
Pored ove transformacije takođe mora se voditi računa o saobraćaju u prozoru od poslednjih 5 minuta. Pošto je potrebno neko vrijeme za procesuiranje samih podataka i komunikaciju natrag ka vozilima preko funkcije *reduceByKey* uveden će biti i slajd interval od 10 sekundi. Podaci se neće transformisati preko stanja i čuvati u samom dijelu platforme već će biti poslati samoj Cassandra bazi podataka. Princip će biti isti kao u funkciji TotalTrafficData.

```
// reduce by key and window (5 minuta prozor i 10 sekundi slide).
        JavaPairDStream<AggregateKey, Long> countDStreamPair =
filteredIotDataStream

           .mapToPair(iot -> new Tuple2<>(new AggregateKey(iot.getRouteId(),
iot.getVehicleType()), 1L))

           .reduceByKeyAndWindow((a, b) -> a + b, Durations.seconds(300),
Durations.seconds(10));
```

Slika 73: ReducebyKey

4.4. Tačke od značaja u procesuiranju saobraćaja

Naravno da bi platforma imala svoje pokriće u realnom sistemu, ona mora se držati realnih parametara. Naime ne mogu sva vozila da imaju isti značaj. Zbog toga se uvodi rastojanje od 100km za koje je značajno znati podatke o saobraćaju.



Slika 74: Vozila u radijusu od 100 km

Dalje treba da se definiše rastojanje. IotDataProducer preko Kafka alata kreira podatke za geografsku širinu i dužinu. Ova platforma će koristiti Harversine formulu za izračunavanje rastojanja preko geografske širine i dužine. Ne uzima trasu puta i putnu konfiguraciju pa je zgodna kao simulator .

```
//Funkcija za kreiranje POITrafficData objekta od IoT data
    private static final Function<Tuple2<IoTData, POIData>, POITrafficData>
    poiTrafficDataFunc = (tuple -> {
    POITrafficData poiTraffic = new POITrafficData();
    poiTraffic.setVehicleId(tuple._1.getVehicleId());
    poiTraffic.setVehicleType(tuple._1.getVehicleType());
    poiTraffic.setTimeStamp(new Date());
double distance =
GeoDistanceCalculator.getDistance(Double.valueOf(tuple._1.getLatitude()).doubleValue()
Double.valueOf(tuple._1.getLongitude()).doubleValue(), tuple._2.getLatitude(),
tuple._2.getLongitude());
tuple._1.getLongitude() + ","+ tuple._2.getLatitude() + "," + tuple._2.getLongitude()
+ " = " + distance);
poiTraffic.setDistance(distance);
return poiTraffic; });
```

Slika 75: Kreiranje POI objekta

```
public static double getDistance(double lat1, double lon1, double lat2, double lon2) {
        final int r = 6371;          //Zemljin radius u KM
        Double latDistance = Math.toRadians(lat2 - lat1);
        Double lonDistance = Math.toRadians(lon2 - lon1);
        Double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2) +
Math.cos(Math.toRadians(lat1))
                * Math.cos(Math.toRadians(lat2)) * Math.sin(lonDistance / 2) *
Math.sin(lonDistance / 2);
        Double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
        double distance = r * c;
        return distance; } * Metod za provjeru da li je lokacija u radius od (POI) lokacije
     * @param currentLat geografska širina trenutne lokacije
     * @param currentLon geografska dužina trenutne lokacije
     * @param poiLat geografska širina POI lokacije
     * @param poiLon geografska dužina POI lokacije
     * @param radius radius u kilometrima od POI
     * @return vraća vrijednost ako je unutar radiusa usuprotnom ne
    public static boolean isInPOIRadius(double currentLat, double currentLon, double
poiLat, double poiLon,double radius){
        double distance = getDistance(currentLat,currentLon,poiLat,poiLon);
         if(distance <= radius){
             return true; }
        return false; }
```

Slika 76: Funkcija getDistance() preko Haversine formule

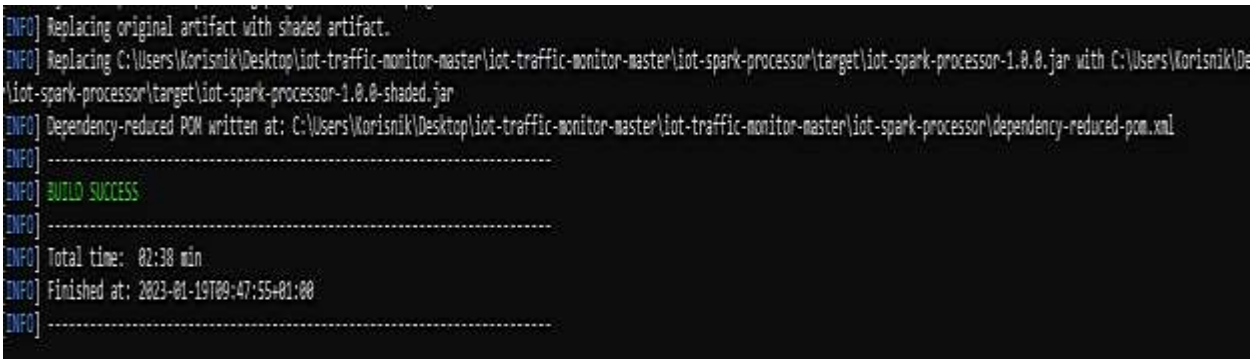
Dalje je potrebno filtrirati podatke na specijalne puteve (Put 37, Put 43, Put 44, Put 69 Put 82) i takođe preko id vozila (napr. kamion, taksi, privatni automobil). Ove filtrirane IotData i POIData objekte treba zadržati u paru i koristiti ih za transformaciju u objekat entiteta za (POITrafficData) Poi_Traffic tabelu baze podataka Cassandra. To je prikazano na slici 74.

Čuvanje u Cassandra bazi podataka se vrši preko trafficDstream. Dstream će se čuvati samo određeno vrijeme. Izvršava se uz pomoć metoda withConstantTTL (120) što znači da će se podaci čuvati 120 sekundi (2 minuta).

```
// poziv CassandraStreamingJavaUtil funkcije za cuvanje u bazu
        javaFunctions(trafficDStream)
                .writerBuilder("traffickeyspace",
"poi_traffic",CassandraJavaUtil.mapToRow(POITrafficData.class,
columnNameMappings))
                .withConstantTTL(120)//cuvanje podataka 2 minuta
                .saveToCassandra();
    }
```

Slika 77: Cassandra za POI vozila

Alat IoTDataprocessor je spreman. Sada treba otvoriti prozor Command prompta i pozicionirati se unutar foldera sa projektom. Nakon toga treba ukucati komandu mvn package.



Slika 78: mvn package komanda

Komandom mvn package napravljen je u iot-kafka-processor target fajlu jar fajl koji je zatim potrebno pokrenuti pomoću Spark streaming alata, koji pruža da podatke koje smo pomoću producera napravili da budu proslijeđeni u bazu podataka Cassandru.

Sledeći korak je spark-submit --class "com.iot.app.spark.processor.IoTDataProcessor" iot-spark-processor-1.0.0.jar, koja prosleđuje podatke generisane i popunjava tabelu podataka već ranije napravljenu.

Slika 79: Spark-sumbit prosleđivanje podataka

Izvršavanjem komandi u cqlsh bazi :

- SELECT * FROM TrafficKeySpace.Total_Traffic;
- SELECT * FROM TrafficKeySpace.Window_Traffic;
- SELECT * FROM TrafficKeySpace.Poi_Traffic;

Dobijamo prikaz kako je Spark Streaming alat popunio bazu podataka Cassandru, proslijedio podatke od producera prema topicu.

```
cqlsh> SELECT * FROM TrafficKeySpace.Total_Traffic;

 routeid   | recorddate | vehicletype   | timestamp                 | totalcount
-----------+------------+---------------+---------------------------+------------
 Route-69  | 2023-03-07 |          Auto | 2023-03-07 09:01:10+0000  |         15
 Route-69  | 2023-03-07 |       Autobus | 2023-03-07 09:00:50+0000  |          7
 Route-69  | 2023-03-07 |   Mali kamion | 2023-03-07 09:00:55+0000  |         15
 Route-69  | 2023-03-07 |          Taxi | 2023-03-07 09:01:10+0000  |         15
 Route-69  | 2023-03-07 | Veliki kamion | 2023-03-07 09:00:15+0000  |          7
 Route-44  | 2023-03-07 |          Auto | 2023-03-07 09:01:10+0000  |          8
 Route-44  | 2023-03-07 |       Autobus | 2023-03-07 09:01:05+0000  |         12
 Route-44  | 2023-03-07 |   Mali kamion | 2023-03-07 09:01:20+0000  |          8
 Route-44  | 2023-03-07 |          Taxi | 2023-03-07 09:01:15+0000  |          6
 Route-44  | 2023-03-07 | Veliki kamion | 2023-03-07 09:01:15+0000  |         11
 Route-37  | 2023-03-07 |          Auto | 2023-03-07 09:00:55+0000  |         10
 Route-37  | 2023-03-07 |       Autobus | 2023-03-07 09:00:05+0000  |          7
 Route-37  | 2023-03-07 |   Mali kamion | 2023-03-07 08:59:55+0000  |         10
 Route-37  | 2023-03-07 |          Taxi | 2023-03-07 09:00:50+0000  |         10
 Route-37  | 2023-03-07 | Veliki kamion | 2023-03-07 09:00:10+0000  |          5
 Route-82  | 2023-03-07 |          Auto | 2023-03-07 09:01:10+0000  |         10
 Route-82  | 2023-03-07 |       Autobus | 2023-03-07 09:01:20+0000  |         14
 Route-82  | 2023-03-07 |   Mali kamion | 2023-03-07 09:00:35+0000  |          3
 Route-82  | 2023-03-07 |          Taxi | 2023-03-07 09:00:40+0000  |          9
 Route-82  | 2023-03-07 | Veliki kamion | 2023-03-07 09:01:05+0000  |         12
 Route-43  | 2023-03-07 |          Auto | 2023-03-07 09:00:50+0000  |          7
 Route-43  | 2023-03-07 |       Autobus | 2023-03-07 09:00:55+0000  |          8
 Route-43  | 2023-03-07 |   Mali kamion | 2023-03-07 09:01:05+0000  |          9
 Route-43  | 2023-03-07 |          Taxi | 2023-03-07 09:00:35+0000  |          5
 Route-43  | 2023-03-07 | Veliki kamion | 2023-03-07 09:01:10+0000  |          8

(25 rows)
cqlsh> SELECT * FROM TrafficKeySpace.Window_Traffic;

 routeid   | recorddate | vehicletype   | timestamp                 | totalcount
-----------+------------+---------------+---------------------------+------------
 Route-69  | 2023-03-07 |          Auto | 2023-03-07 09:01:20+0000  |         15
 Route-69  | 2023-03-07 |       Autobus | 2023-03-07 09:01:20+0000  |          7
 Route-69  | 2023-03-07 |   Mali kamion | 2023-03-07 09:01:20+0000  |         15
 Route-69  | 2023-03-07 |          Taxi | 2023-03-07 09:01:20+0000  |         15
 Route-69  | 2023-03-07 | Veliki kamion | 2023-03-07 09:01:20+0000  |          7
 Route-44  | 2023-03-07 |          Auto | 2023-03-07 09:01:20+0000  |          8
 Route-44  | 2023-03-07 |       Autobus | 2023-03-07 09:01:20+0000  |         12
 Route-44  | 2023-03-07 |   Mali kamion | 2023-03-07 09:01:20+0000  |          8
 Route-44  | 2023-03-07 |          Taxi | 2023-03-07 09:01:20+0000  |          6
 Route-44  | 2023-03-07 | Veliki kamion | 2023-03-07 08:01:20+0000  |         11
```

Slika 80: Baza podataka Cassandra nakon Spark Sumbit

Završena je instalacija drugog dijela projekta za monitoring saobraćaja. Sada je ostala IoT Data DashBoard (kontrolna tabla na kojoj treba da budu prikazani najznačajniji podaci).

4.5. IoT Data DashBoard

Za Dashboard dio platforme će biti korišćen Spring Boot. Kao tehnologija Spring Boot će imati veliku podršku u radu sa Cassandra bazom podataka u kojoj su smješteni podaci iz drugog dijela tj. IoTDataProcessor. Kao i do sada na samom početku će biti prikazan pom.xml fajl sa zavisnostima (tehnologijama koje će DashBoard platforma da koristi).

```
<!-- Importuh dependency management od Spring Boot -->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.3.5.RELEASE</version>
        <relativePath />
    </parent>

    <dependencies>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>1.6.2</version>
    </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-websocket</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-logging</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-cassandra</artifactId>
        </dependency>
        <!-- Ostali dependencies -->
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
        </dependency>
```

Slika 81: Sadržaj pom.xml fajla DashBoard platforme

Primjetno je korišćenje prvenstveno Spring Boot-a. Dalje u fajlu se definišu razni pluginovi zarad povezivanja Maven-a i Spring Boot-a.

Sada će biti kreirane tri klase entiteta za tri tabele iz Cassandra baze podataka. To će biti Total_Traffic, Window_Traffic i Poi_Traffic tabele. Biće kreirani DAO (Data Access Object) interfejsi za sva tri entiteta. Takođe, moguće je dodati poseban Cassandra upit za preuzimanje

potrebnih podataka iz baze.

```
@Repository
public interface TotalTrafficDataRepository extends
CassandraRepository<TotalTrafficData>{
     @Query("SELECT * FROM traffickeyspace.total_traffic WHERE recorddate = ?0
ALLOW FILTERING")
     Iterable<TotalTrafficData> findTrafficDataByDate(String date);  }
```

Slika 82: Upit za TotalTrafficData

Dalje je potrebna konfiguracija CassandraConfig klase koja se povezuje sa Cassandra klasterom zarad daljih operacija.

```
public class CassandraConfig extends AbstractCassandraConfiguration {
    private static final Logger logger = Logger.getLogger(CassandraConfig.class);
    @Autowired
    private Environment environment;
    @Bean
    public CassandraClusterFactoryBean cluster() {
    CassandraClusterFactoryBean cluster = new CassandraClusterFactoryBean();
    String cassandraHost = environment.getProperty("com.iot.app.cassandra.host");
      if (System.getProperty("com.iot.app.cassandra.host") != null) {
         cassandraHost = System.getProperty("com.iot.app.cassandra.host"); }
    String cassandraPort = environment.getProperty("com.iot.app.cassandra.port");
      if (System.getProperty("com.iot.app.cassandra.port") != null) {
          cassandraPort = System.getProperty("com.iot.app.cassandra.port");  }
    @Bean
    public CassandraMappingContext cassandraMapping(){
    return new BasicCassandraMappingContext(); }
    @Override
    @Bean
    protected String getKeyspaceName() {
    return environment.getProperty("com.iot.app.cassandra.keyspace"); }
```

Slika 83: CassandraConfig klasa

Na tabli naše platforme treba da se podaci osvježavaju periodično. Tu funkcionalnost obavlja klasa webSocketConfig. Podaci se šalju preko SockJS html stranici.

```
public class WebSocketConfig extends AbstractWebSocketMessageBrokerConfigurer {
    //sockJS može dobiti poruku upotrebom ove krajnje tačke
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/stomp").withSockJS();
    }
    //konfigurišite brokera poruka
    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker("/topic");
    }
```

Slika 84: Slanje poruka preko SockJS

Podatke dalje treba slati svakih 5 sekundi. TrafficDataService klasa će obezbijediti povlačenje podataka iz Cassandra tabela preko repozitorijumskih interfejsa. Svakih 5 sekundi slaće se odgovor kontrolnoj tabli. Klasa će koristiti Spring-ov SimpMessagingTemplate za slanje poruka.

```
public class TrafficDataService {
//Metod šalje traffic data poruke svakih 5 sekundi.
    @Scheduled(fixedRate = 5000)
    public void trigger() {
        List<TotalTrafficData> totalTrafficList = new
ArrayList<TotalTrafficData>();
        List<WindowTrafficData> windowTrafficList = new
ArrayList<WindowTrafficData>();
        List<POITrafficData> poiTrafficList = new ArrayList<POITrafficData>();
        //pozovi dao metod
        totalRepository.findTrafficDataByDate(sdf.format(new Date())).forEach(e ->
totalTrafficList.add(e));
        windowRepository.findTrafficDataByDate(sdf.format(new Date())).forEach(e -
> windowTrafficList.add(e));
        poiRepository.findAll().forEach(e -> poiTrafficList.add(e));
        //pripremi odgovor
        Response response = new Response();
        response.setTotalTraffic(totalTrafficList);
        response.setWindowTraffic(windowTrafficList);
        response.setPoiTraffic(poiTrafficList);
        logger.info("Sending to UI "+response);
        //pošalji prema ui
        this.template.convertAndSend("/topic/trafficData", response); }
```

Slika 85: TrafficDataService klasa

Spring Boot platforma će biti uspostavljena na portu 8080.

```
@SpringBootApplication
@EnableScheduling
@ComponentScan
(basePackages = {"com.iot.app.springboot.dashboard",
"com.iot.app.springboot.dao"})

public class IoTDataDashboard {
      public static void main(String[] args) {
            SpringApplication.run(IoTDataDashboard.class, args);
        }
```

Slika 86: Klasa IotDataDashboard

Na kraju ponovo se treba pozicionirati unutar novog prozora u folder sa platformom. Nakon toga treba ukucati komandu mvn package.

```
[INFO]
[INFO] --- maven-jar-plugin:2.5:jar (default-jar) @ iot-springboot-dashboard ---
[INFO]
[INFO] --- spring-boot-maven-plugin:1.3.5.RELEASE:repackage (default) @ iot-springboot-dashboard ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.278 s
[INFO] Finished at: 2023-03-07T10:05:21+01:00
[INFO] ------------------------------------------------------------------------

mcatEmbeddedServletContainer - Tomcat started on port(s): 8080 (http)
4305 [com.iot.app.springboot.dashboard.IoTDataDashboard.main()] INFO com.iot.app.springboot.dashboard.IoTDataDashboard -
 Started IoTDataDashboard in 4.22 seconds (JVM running for 8.73)
2023-03-07 10:07:26 INFO  TrafficDataService:62 - Sending to UI com.iot.app.springboot.vo.Response@bf61a70
2023-03-07 10:07:31 INFO  TrafficDataService:62 - Sending to UI com.iot.app.springboot.vo.Response@351e9016
2023-03-07 10:07:36 INFO  TrafficDataService:62 - Sending to UI com.iot.app.springboot.vo.Response@55316f59
2023-03-07 10:07:41 INFO  TrafficDataService:62 - Sending to UI com.iot.app.springboot.vo.Response@517f56c9
2023-03-07 10:07:46 INFO  TrafficDataService:62 - Sending to UI com.iot.app.springboot.vo.Response@77c775eb
2023-03-07 10:07:51 INFO  TrafficDataService:62 - Sending to UI com.iot.app.springboot.vo.Response@18aa63f8
```

Slika 87: Kreiranja IoTDataBoard platforme

Sledeća komanda koju treba otkucati je mvn exec:java -Dexec.mainClass=" com.iot.app.springboot.dashboard.IoTDataDashboard" isto u folderu u kome se nalazi sam projekat. Sada je moguće pristupiti samoj web platformi preko local hosta na adresi http://localhost:8080. Dobija se web platforma po kojoj se može vršiti monitoring saobraćaja.

Pored toga na http://localhost:4040 se istovremeno podiže Spark Procesor koji nudi razne opcije kao što su pregled Jobs, Stages, Storage, Environment, Executors i Streaming.

5. Korisnički interfejs

Ova web platforma koristi web Sockets i jQuery da šalje podatke na web stranicu u fiksnim intervalima tako da će se podaci automatski osvježavati. Kontrolna tabla prikazuje podatke u grafikonima i tabelama. Ova web stranica koristi bootstrap.js za responzivni web dizajn, tako da je dostupna kako na desktopu tako i na mobilnim uređajima.

Podaci vezani za stranice ostali su u resources/static folderu. Dalje treba dodati JQuery, Sockjs i Stomp javascript bibiloteke da stranica može da primi podatke sa web Socket-a.

Izveli smo seriju transformacija bez stanja i stanja pomoću API-ja za striming Spark na Dstreamovima i držali ih u Cassandra tabelama baze podataka. Razvili smo prilagodljivu kontrolnu tablu za praćenje web saobraćaja koristeći Spring Boot, SockJs i Bootstrap koja traži podatke iz baze podataka Cassandra i šalje do korisničkog interfejsa koristeći web socket.

```
<script type="text/javascript" src="js/jquery-1.12.4.min.js"></script>
    <script type="text/javascript" src="js/sockjs-1.1.1.min.js"></script>
    <script type="text/javascript" src="js/stomp.min.js"></script>
    <script type="text/javascript" src="js/bootstrap.min.js"></script>`
    <script type="text/javascript" src="js/Chart.min.js"></script>
    <script type="text/javascript">

     var totalTrafficChartData={
                labels : ["Vehicle"],
                datasets : [{
                    label : "Route",
                    data : [1]  }]};
          var route37TrafficChartData={
                labels : ["Vehicle"],
                datasets : [{
                    data : [1]  }
]
};
```

Slika 88: Linkovi ka Skript fajlovima unutar Index.html

- jquery-1.12.4.min.js: Ovo je verzija jQuery biblioteke, JavaScript biblioteka koja uprošćava proces selekcije i manipulacije HTML elementima, upravljanje događajima i donošenjem pravih odluka.
- sockjs-1.1.1.min.js: Ovo je verzija od SockJS biblioteke, JavaScript biblioteki koja nudi objekat nalik webSocket u slučaju kada webSocket nije dostupan u korisničkom web browseru.
- stomp.min.js: Ovo je JavaScript biblioteka koja pruža jednostavan način za

komunikaciju preko webSocketa, omogućavajući u realnom vremenu, bi-direkcionu komunikaciju između web klijenta i servera.

- bootstrap.min.js: Ovo je verzija od Bootstrap biblioteke, popularni front-end frejmwork koji pruža mogućnosti unaprijed-ugrađenih CSS stilova i JavaScript plugina za kreiranje odgovarajućih i pokretnih web stranica i aplikacija
- Chart.min.js: Ovo je JavaScript biblioteka koja pruža lak način za kreiranje animacija i interaktivnih grafikona i chartova korišćenjem HTML platna.

Dalje je potrebno preuzeti fajlove iz web Socket-a i parsirati ih kao JSON podatke.

```
    //koristi sockjs
    var socket = new SockJS('/stomp');
    var stompClient = Stomp.over(socket);

    stompClient.connect({ }, function(frame)
{
    //subskrajb "/topic/trafficData" poruku
    stompClient.subscribe("/topic/trafficData", function(data)
{
          var dataList = data.body;
          var resp=jQuery.parseJSON(dataList);
          //Total traffic
          var totalOutput='';
          jQuery.each(resp.totalTraffic, function(i,vh) {
          totalOutput +="<tbody><tr><td>"+
vh.routeId+"</td><td>"+vh.vehicleType+"</td><td>"
+vh.totalCount+"</td><td>"+vh.timeStamp+"</td></tr></tbody>";
                    }
);
          var t_tabl_start = "<table class='table table-bordered table-condensed
table-hover innerTable'>
<thead><tr>
<th>Route</th><th>Vehicle</th><th>Count</th><th>Time</th></tr></thead>";
var t_tabl_end = "</table>";
                    totalTrafficList.html(t_tabl_start+totalOutput+t_tabl_end);
```

Slika 89: Konektovanje i preuzimanje sa web Socket-a

Sada na kraju samo ostaje dodavanje skripti vezanih za grafikone i prikaz podataka. Za to se koristi Chart.Js bibiloteka. Ova bibiloteka sadrži različite API-je za različite tipove korisnih grafika.

5.1. Dizajn

Za dizajn je korišćen JavaScript kod koji definiše tri objekta grafikona i tabela koristeći biblioteku Chart.Js. Grafikoni se prikazuju pomoću tri različite vrste grafikona: bar grafikon, kartu krofne i radarski grafikon.

```
//tabele
 var totalTrafficList = jQuery("#total_traffic");
 var windowTrafficList = jQuery("#window_traffic");
 var poiTrafficList = jQuery("#poi_traffic");
```

Slika 90: Kreiranja Tabela

```
var totalTrafficChartData={
    labels : ["Vehicle"],
    datasets : [{
    label : "Route",
    data : [1]
                }
               ]
            };

var route37TrafficChartData={
    labels : ["Vehicle"],
    datasets : [{
    data : [1]
                }
               ]
            };

var poiTrafficChartData={
    labels : ["Vehicle"],
    datasets : [{
    data : [1]
                }
               ]
            };
```

```
//Grafici
  var ctx1 =
document.getElementById("totalTrafficChart").ge
tContext("2d");
  window.tChart = new Chart(ctx1, {
     type: 'bar',
     data: totalTrafficChartData
});

  var ctx2 =
document.getElementById("route37TrafficChart").
getContext("2d");
  window.wChart = new Chart(ctx2, {
     type: 'doughnut',
      data: route37TrafficChartData
});

  var ctx3 =
document.getElementById("poiTrafficChart").getC
ontext("2d");
  window.pChart = new Chart(ctx3, {
      type: 'radar',
      data: poiTrafficChartData
});
```

Slika 91: Kreiranja Grafika za vizualizaciju

Podaci za svaki grafikon definisani su korišćenjem objekata sa svojstvom labels i datasets. Labels određuje etikete za svaku tačku podataka u grafikonu, a dataset je niz objekata koji određuju podatke o podacima za svaki skup podataka.

Svaki grafikon se zatim kreira pomoću biblioteke Chart.js i dodijeljen je promjenljivoj u prozoru. Ove promjenljive se zatim mogu koristiti za ažuriranje ili interakciju sa podacima o

grafikonu ili opcijama u kasnijem dijelu koda.

Kod takođe uključuje funkciju jQuery koja se izvršava kada je dokument spreman. Ova funkcija inicijalizuje tri objekta grafikona i postavlja njihove početne podatke koristeći ranije definisane podatke.

```
//Total traffic
    var totalOutput='';
    jQuery.each(resp.totalTraffic, function(i,vh) {
    totalOutput +="<tbody><tr><td>"+
vh.routeId+"</td><td>"+vh.vehicleType+"</td><td>"+vh.totalCount+"</td><td>"+vh.timeStamp+"<
/td></tr></tbody>";});
    var t_tabl_start = "<table class='table table-bordered table-condensed table-hover
innerTable'><thead><tr><th>Route</th><th>Vehicle</th><th>Count</th><th>Time</th></tr></thea
d>";
    var t_tabl_end = "</table>";
    totalTrafficList.html(t_tabl_start+totalOutput+t_tabl_end);

//Window traffic
     var windowOutput='';
     jQuery.each(resp.windowTraffic, function(i,vh) {
     windowOutput +="<tbody><tr><td>"+
vh.routeId+"</td><td>"+vh.vehicleType+"</td><td>"+vh.totalCount+"</td><td>"+vh.timeStamp+"<
/td></tr></tbody>";});
     var w_tabl_start = "<table class='table table-bordered table-condensed table-hover
innerTable'><thead><tr><th>Route</th><th>Vehicle</th><th>Count</th><th>Time</th></tr></thea
d>";
     var w_tabl_end = "</table>";
     windowTrafficList.html(w_tabl_start+windowOutput+w_tabl_end);
//POI data
     var poiOutput='';
     jQuery.each(resp.poiTraffic, function(i,vh) {
     poiOutput +="<tbody><tr><td>"+
vh.vehicleId+"</td><td>"+vh.vehicleType+"</td><td>"+vh.distance+"</td><td>"+vh.timeStamp+"<
/td></tr></tbody>";});
     var p_tabl_start = "<table class='table table-bordered table-condensed table-hover
innerTable'><thead><tr><th>Vehicle
Id</th><th>Vehicle</th><th>Distance</th><th>Time</th></tr></thead>";
     var p_tabl_end = "</table>";
     poiTrafficList.html(p_tabl_start+poiOutput+p_tabl_end);
```

Slika 92: Kreiranja Tabela za vizualizaciju

U ovom dijelu je kod koji ažurira sadržaj tri HTML elemenata na stranici. Total_traffic, Window_traffic i Poi_traffic. Ažurirani sadržaj uključuje podatke o saobraćaju preuzete iz objekata.

Kod koristi jQuery funkciju na iteraciju Total_traffic, Window_traffic i Poi_traffic nizove u respozitorijum objektu. Za svaki objekat u svakom nizu, novi red HTML-a dodaje se u odgovarajuću HTML tabelu.

HTML za svaku tabelu je definisan korišćenjem promjenljivih koji čuvaju početne i krajnje oznake za tabele, kao i sve potrebne tabele zaglavlja. HTML redovi za svaku tabelu se zatim dodaju odgovarajućoj tabeli HTML-a koristeći jQuery metod.

Rezultat toga je da se sadržaj tri HTML elementa na stranici ažurira sa podacima o saobraćaju preuzetim iz odgovora servera. Ovo omogućava korisnicima da pregledaju podatke o saobraćaju u tabelarnom formatu pored tri charta definisana ranije u kodu.

```
function drawBarChart(trafficDetail,trafficChartData){
        //Pripremi podatke za total traffic grafik
        var chartLabel = [ "Veliki kamion", "Mali kamion", "Auto", "Autobus", "Taxi"];
        var routeName = ["Route-37", "Route-43", "Route-44", "Route-69", "Route-82"];
        var chartData0 =[0,0,0,0,0], chartData1 =[0,0,0,0,0], chartData2 =[0,0,0,0,0],
chartData3 =[0,0,0,0,0], chartData4 =[0,0,0,0,0];
        jQuery.each(trafficDetail, function(i,vh) {
        if(vh.routeId == routeName[0]){
        chartData0.splice(chartLabel.indexOf(vh.vehicleType),1,vh.totalCount);}
        if(vh.routeId == routeName[1]){
        chartData1.splice(chartLabel.indexOf(vh.vehicleType),1,vh.totalCount)}
        if(vh.routeId == routeName[2]){
        chartData2.splice(chartLabel.indexOf(vh.vehicleType),1,vh.totalCount);}
        if(vh.routeId == routeName[3]){
        chartData3.splice(chartLabel.indexOf(vh.vehicleType),1,vh.totalCount); }
        if(vh.routeId == routeName[4]){
        chartData4.splice(chartLabel.indexOf(vh.vehicleType),1,vh.totalCount) }});
```

Slika 93: Kreiranja modula za ukupni saobraćaj

```
var trafficData = {
    labels : chartLabel,
    datasets : [{
       label                : routeName[0],
       borderColor          : "#f75821",
       backgroundColor      : "#f75821",
       data                 : chartData0    },{
       label                : routeName[1],
       borderColor          : "#202951",
       backgroundColor      : "#202951",
       data                 : chartData1    },{
       label                : routeName[2],
       borderColor          : "#33b242",
       backgroundColor      : "#33b242",
       data                 : chartData     },{
```

Slika 94: Kreiranja modula za ukupni saobraćaj

U ovom dijelu koristimo dvije vrste parametara a to su trafficDetail i trafficChartData.

Svrha ove funkcije je priprema podataka za bar grafikon koji će prikazati detalje o saobraćaju za različite rute i vrste vozila.

Funkcija prvo inicijalizuje neke nizove za chart labels i data, a zatim prolazi kroz iteracije putem svake stavke u trafficDetail. Za svaku stavku funkcija provjerava svojstva routeId i vehicleType i ažurira odgovarajući indeks u odgovarajućem nizu podataka.

Nakon ažuriranja niza podataka, funkcija stvara novi objekt TrafficData koji sadrži ažurirane data i labels i ažurira datasets i labels objekta TrafficChartData sa novim podacima.

Ova funkcija je odgovorna za ažuriranje data i labels bar grafikona koja prikazuje detalje o saobraćaju za različite rute i tipove vozila.

Na sličan način su odrađeni i ostali prozori dok kao dodatak za prikaz karte je korišćena Google Maps API.

```
<table class="table outerTable">
        <tbody>
           <td>
             <table class="table table-bordered table-condensed table-hover
innerTable">
                <tr>
                   <thead>
<iframe
src="https://www.google.com/maps/d/embed?mid=1j5EZpMkM1vFL4km96vBzTeiDfdztA5Y&ehbc
=2E312F"
 width="100%" height="1000" ></iframe>
                        </thead>
                </tr>
             </table>
           </td>
        </tbody>
     </table>
```

Slika 95: Kreiranja modula Google Maps API

API za Google mape je API (skraćeno od „interfejs za programiranje aplikacije“) koji omogućava programerima da pristupe podacima i funkcionalnosti Google mapa za sopstvene projekte. Za našu platformu koristićemo prilagođenu Google mapu sa prikazom različitih putanja koji su objašnjeni u prethodnim modulima.

Trenutno, platforma Google mapa nudi više API-ja za različite aspekte svoje usluge. Postoji statički API mape za jednostavno ugrađivanje Google mapa, Maps JavaScript API za interaktivne i prilagodljive mape, API mjesta za pristup podacima o interesantnim mjestima i API uputstva za obezbjeđivanje ruta do lokacije, da spomenemo samo neke.

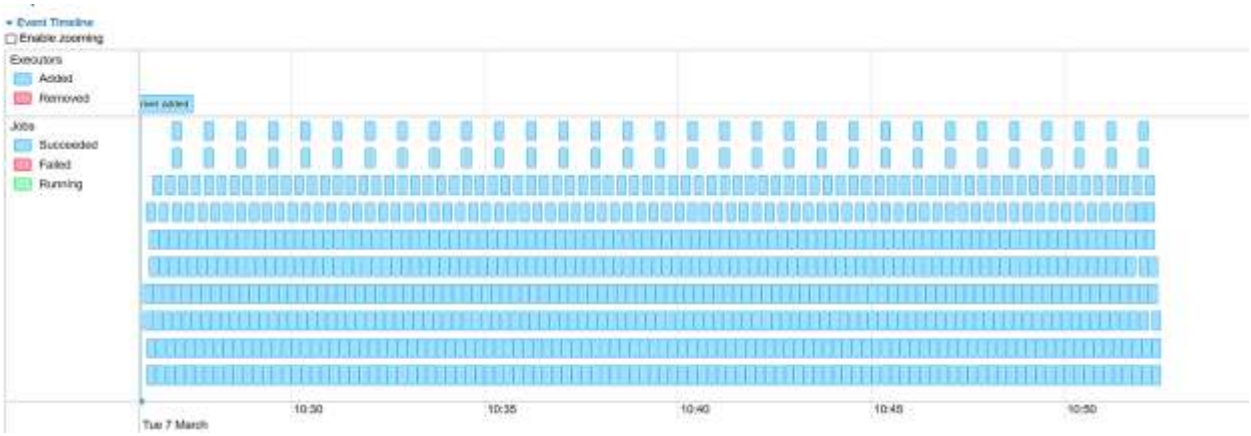
5.2. Navigacija kroz sistem

Apache Spark je alat za procesuiranje podataka, a njegov web UI na adresi http://localhost:4040/ nudi mnogo mogućnosti za praćenje toka procesuiranja i vizualizacije podataka.

Ovdje su neke od glavnih prednosti:

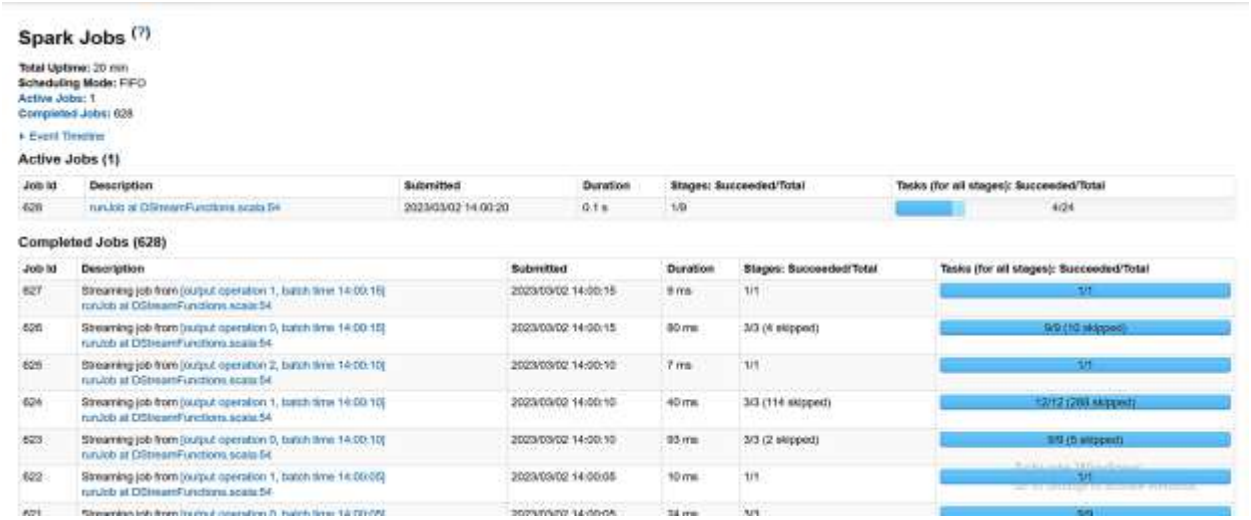
1. Praćenje procesa izvršavanja: Spark web UI pruža detaljne informacije o izvršavanju Spark poslova, procesa i faza (stage). Korisnici mogu koristiti ove informacije kako bi pratili tok procesuiranja i dijagnostifikovali probleme u izvršavanju platformi.
2. Vizualizacija podataka: web UI pruža različite vrste grafikona i vizualizacija kako bi korisnici lakše razumjeli i pratili podatke. Na primjer, korisnici mogu koristiti grafikone kako bi pratili trendove performansi platforme tokom vremena.
3. Upravljanje platformama: Korisnici mogu koristiti Spark web UI kako bi upravljali platformama i zaustavljali ih preko web browsera. Ovo je vrlo korisno za administratore koji žele kontrolisati platforme iz udobnosti svog web pregledača.
4. Dostupnost: Spark web UI je vrlo lako dostupan korisnicima jer se može pristupiti putem web browsera. To znači da korisnici mogu pristupiti Spark web UI-ju sa bilo kojeg mjesta i uređaja, što čini ovaj alat vrlo pristupačnim i praktičnim.
5. Optimizacija performansi: Spark web UI pruža korisne informacije koje programeri i administratori mogu koristiti za optimizaciju performansi svojih platformi. Na primjer, korisnici mogu identifikovati spore faze u izvršavanju i optimizovati ih kako bi poboljšali ukupne performanse platforme.

Spark web UI takođe nudi i karticu Event timeline koja prikazuje hronološkim redom događaje koji se odnose na izvršioce (dodate, uklonjene) i poslove, što je korisno za praćenje izvršavanja Spark poslova, identifikaciju ograničenja u performansama i optimizaciju izvršavanja Spark platformi.



Slika 96: Prikaz Event timeline

Kartica Spark jobs nudi detalje poslova grupisanih prema statusu: Prikazuje detaljne informacije o poslovima uključujući ID posla, opis (sa vezom do detaljne stranice posla), predato vrijeme, trajanje, sažetak faza i traku napredovanja zadataka.



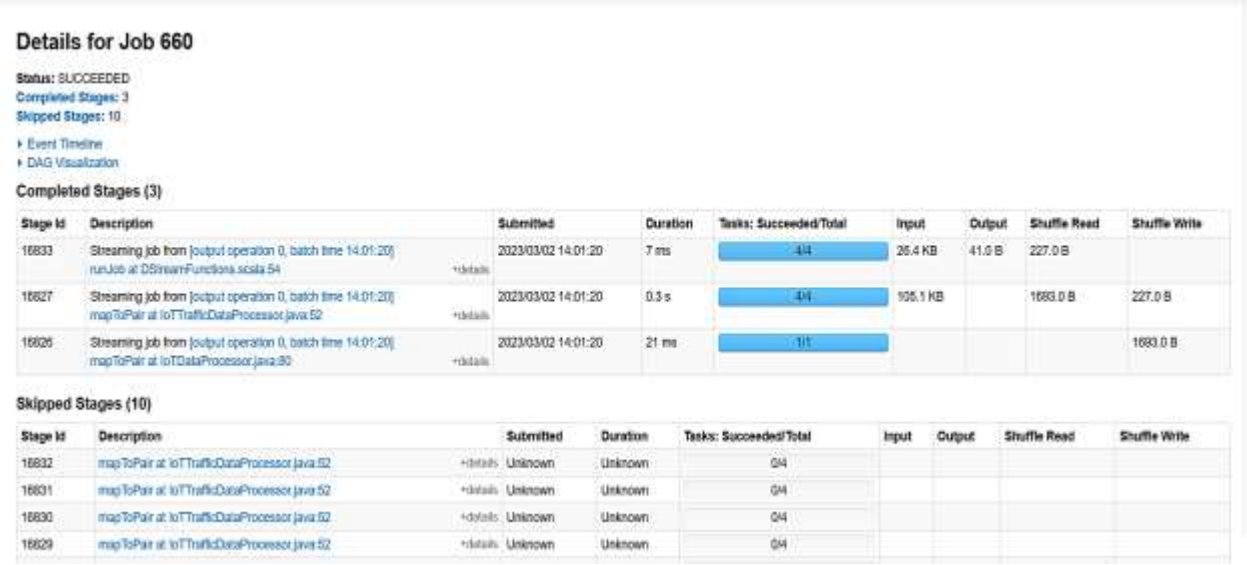
Slika 97: Prikaz Spark Processor jobs

Kada kliknete na određeni posao, možete vidjeti detaljne informacije o ovom poslu.

Details for Job stranica prikazuje detalje određenog posla identifikovanog njegovim ID-om posla:

- Status: (pokrenut, uspjeo, neuspjeo);
- Completed Stages (aktivan, na čekanju, završen, preskočen, neuspješno);
- Vremenska linija događaja: Prikazuje hronološkim redom događaje koji se odnose

na izvršioce (dodate, uklonjene) i faze posla.

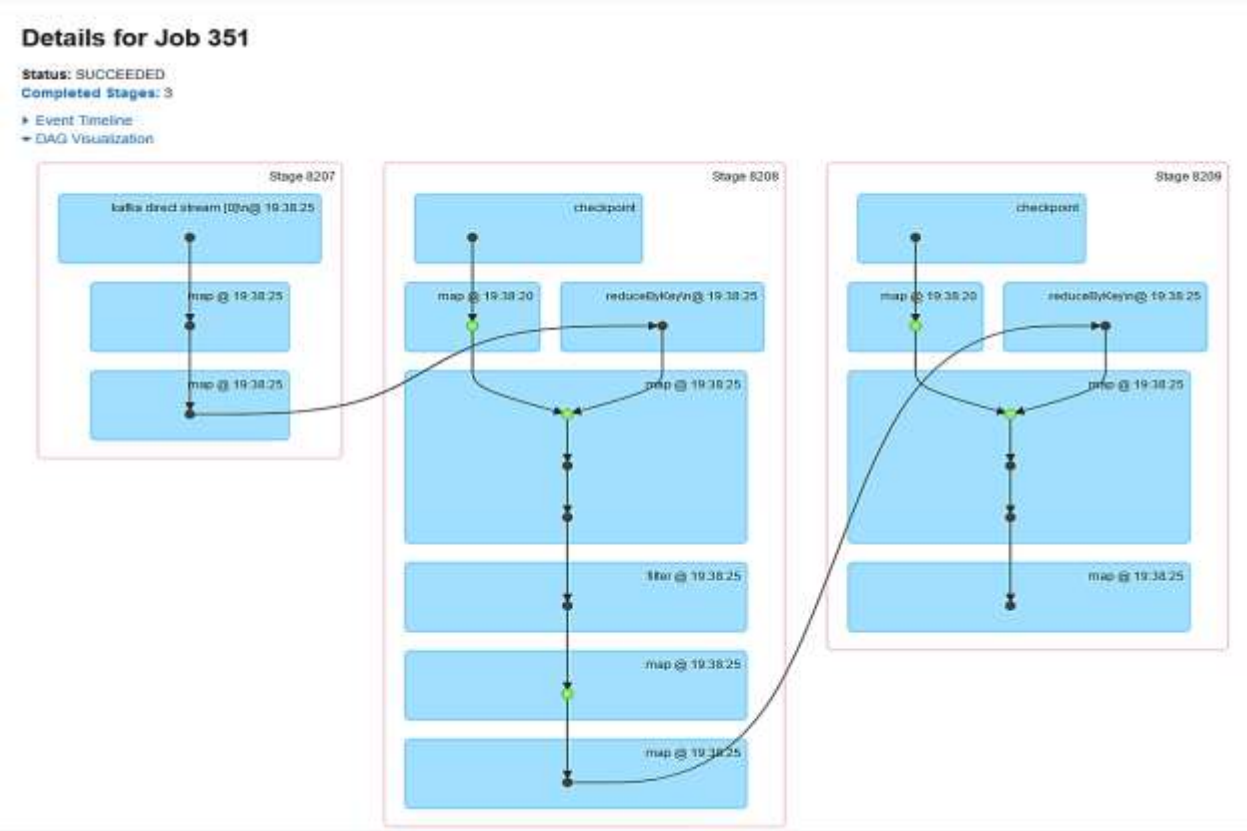


Slika 98: Prikaz Detalja Job 660

DAG (Directed Acyclic Graph) vizualizacija se odnosi na grafički prikaz strukture i zavisnosti između različitih elemenata u DAG modelu. DAG je matematički model koji se koristi u računarstvu za opisivanje složenih procesa i algoritama, a koji se sastoji od čvorova i usmjerenih veza između njih.

Vizualizacija DAG-a može biti korisna za razumijevanje i analizu složenih procesa i algoritama, jer omogućava da se jasno vide zavisnosti i tokovi između različitih čvorova. U zavisnosti od konkretnog modela ili sistema, DAG vizualizacija može biti korišćena za optimizaciju performansi, identifikovanje ograničenja i grešaka, ili jednostavno za bolje razumijevanje rada sistema.

Primjena DAG vizualizacije je česta u različitim oblastima računarstva, kao što su distribuirani sistemi, obrada podataka, mašinsko učenje, procesuiranje slike i drugi. Vizuelni prikaz usmjerenog acikličkog grafa ovog posla gdje vrhovi predstavljaju RDD-ove ili okvire podataka, a ivice predstavljaju operaciju koja se primjenjuje na RDD.



Slika 99: Prikaz Spark Processor DAG vizualizacije 3 stages

DAG vizuelizacija prikazuje ključne djelove ove platforme:

- Kafka Direct Stream -> Map -> MapToPair -> ReduceByKey;
- Checkpoint -> MapWithState -> ReduceByKey -> Map -> Filter.

Kafka direct stream: Prvi korak u procesu je kreiranje direktnog strima iz Kafkine teme. Ovaj tok će sadržati podatke o saobraćaju koje prikupljaju IoT uređaji.

Map: Funkcija map se koristi za primjenu transformacije na tok. U ovom slučaju, funkcija konvertuje neobrađene podatke u svakom zapisu u DStream IotData objekat.

MapToPair: Funkcija mapToPair se koristi za kreiranje novog DStream-a koji se sastoji od parova ključ-objekat. U ovom slučaju, funkcija se koristi za izdvajanje argumenata tipa ID-a vozila a vrijednost će biti IoT Data objekat kreiranog u prethodnom koraku.

Checkpoint: Funkcija kontrolne tačke se koristi za periodično čuvanje stanja platforme za strimovanje u sistemu za skladištenje koji je otporan na greške kao što je HDFS ili Cassandra. Ovo je važno kako bi se osiguralo da se platforma može oporaviti od grešaka i nastaviti obradu tamo gdje je stala.

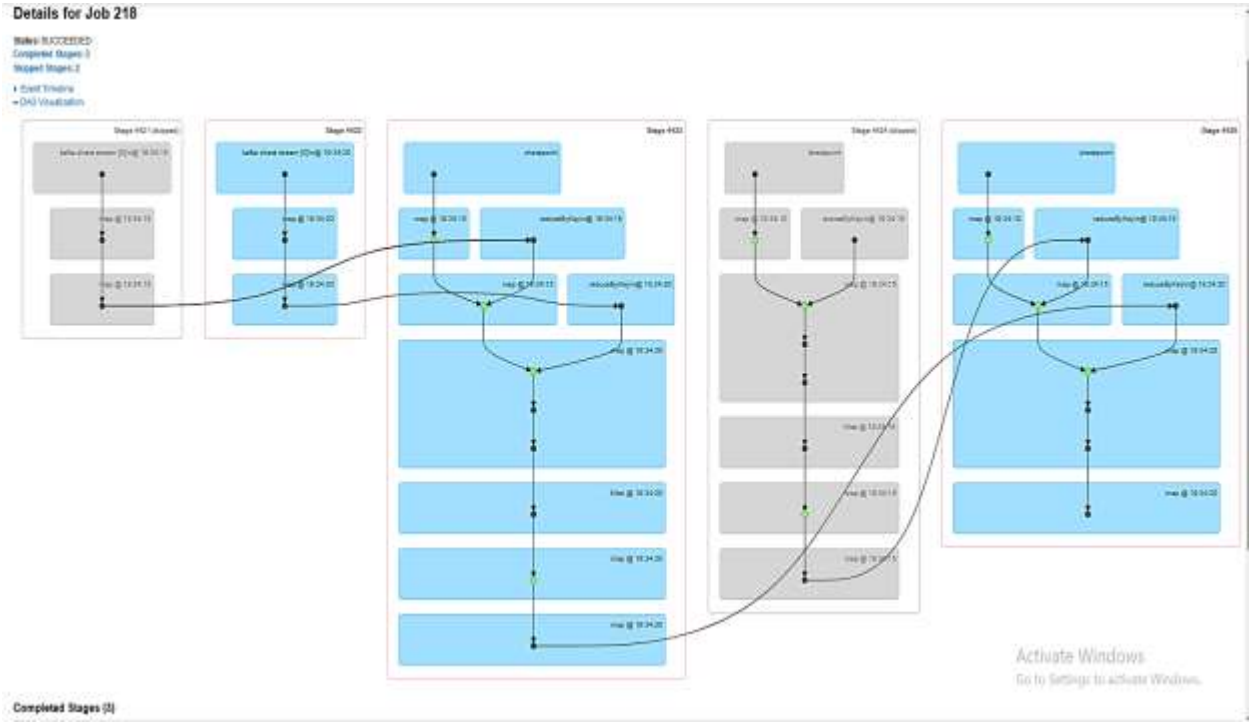
MapWithState: Funkcija mapWithState se koristi za održavanje stanja u više serija podataka. U ovom slučaju, funkcija se koristi za praćenje ukupnog broja vozila koja su prošla kroz svaku rutu tokom svakog vremenskog prozora.

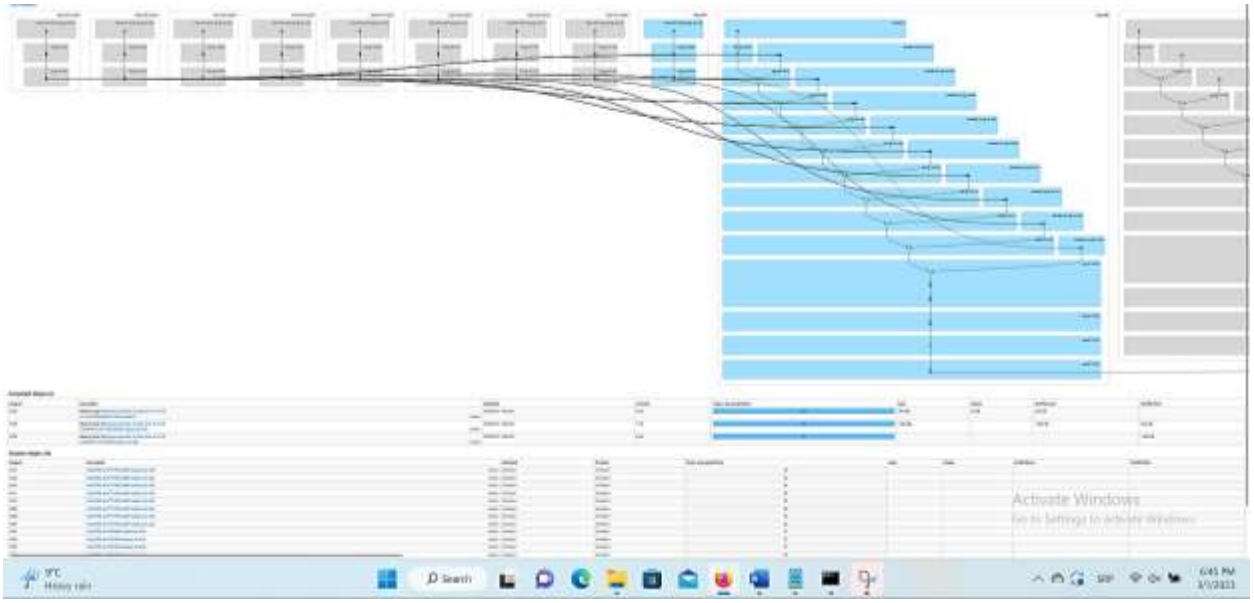
ReduceByKey: Funkcija reduceByKey se koristi za agregiranje podataka na osnovu ključeva u parovima ključ/vrijednost. U ovom slučaju, funkcija se koristi za agregiranja vrijednosti za svaki ključ jer želimo jedinstvena vozila po dolaznoj partiji. U ovoj platformi zadržaćemo vozilo u memoriji jedan sat.

Map: Funkcija map se koristi za primjenu transformacije na podatke. U ovom slučaju, funkcija se koristi za pretvaranje podataka u JSON objekat koji se lako može vizuelizovati.

Filter: Funkcija filtera se koristi za uklanjanje svih podataka koji ne ispunjavaju određene kriterijume. U ovom slučaju, funkcija se koristi za dobijanje vozila koja nijesu obrađena.

Sve u svemu, ovi metodi pokazuje kako se obrađuju i analiziraju podaci o saobraćaju u realnom vremenu koristeći Apache Kafka, Spark Streaming. Postupak uzima neobrađene podatke o saobraćaju, pretvara ih u format ključ/vrijednost, agregira podatke, a zatim ih transformiše u JSON objekat koji se može vizuelizovati. Takođe uključuje mehanizme za toleranciju grešaka i održavanje stanja u više serija podataka.





Slika 100: Prikaz Spark Processor DAG vizualizacije 3 stages

Summary Metrics for 4 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	2 ms	2 ms	2 ms	7 ms	7 ms
Scheduler Delay	0 ms	1 ms	1 ms	1 ms	1 ms
Task Deserialization Time	1 ms	1 ms	1 ms	1 ms	1 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Result Serialization Time	0 ms	0 ms	0 ms	0 ms	0 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	800.0 B	800.0 B
Input Size / Records	4.6 KB / 1	4.9 KB / 1	5.1 KB / 1	5.1 KB / 1	5.1 KB / 1
Output Size / Records	0.0 B / 0	0.0 B / 0	0.0 B / 0	38.0 B / 1	38.0 B / 1
Shuffle Read Blocked Time	0 ms	0 ms	0 ms	0 ms	0 ms
Shuffle Read Size / Records	0.0 B / 0	0.0 B / 0	0.0 B / 0	224.0 B / 1	224.0 B / 1
Shuffle Remote Reads	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B

Slika 101: Detaljan prikaz Spark Processor DAG vizualizacije

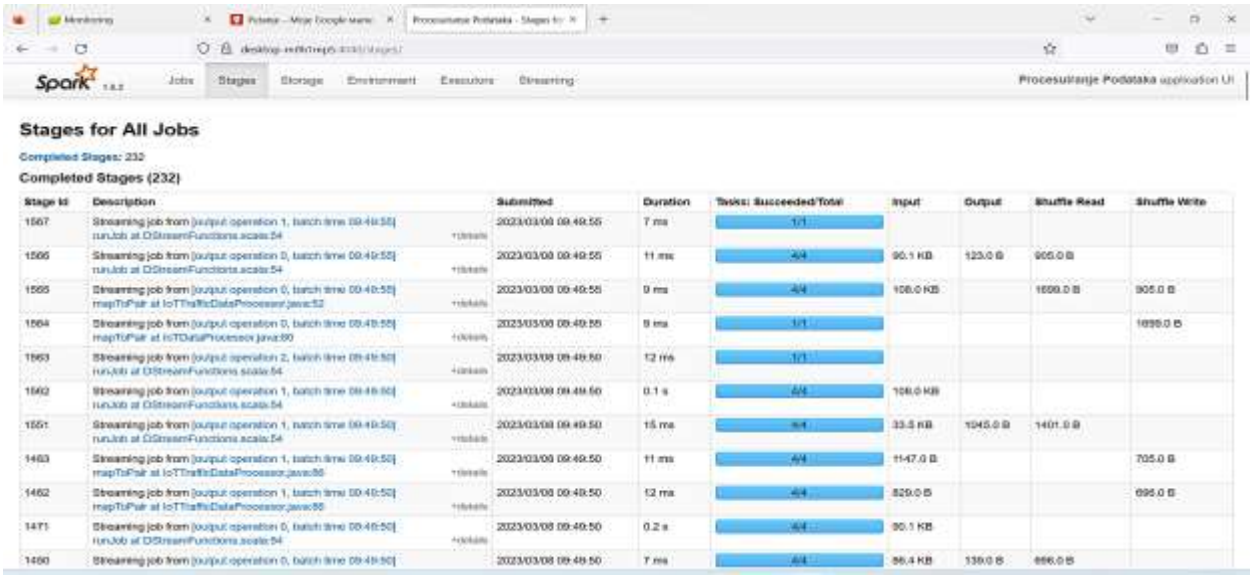
Rezime metrike za sve zadatke su predstavljene u tabeli Summary Metrics for 4 Completed Tasks i na vremenskoj liniji.

- Tasks deserialization time - vrijeme deserializacije zadataka;
- Duration of tasks - trajanje zadataka;

- GC time je ukupno vrijeme sakupljanja smeća JVM-a;
- Result serialization time je vrijeme potrošeno na serijalizaciju rezultata zadatka na izvršiocu prije nego što ga pošalje nazad drajveru;
- Getting result time je vrijeme koje vozač troši na preuzimanje rezultata zadatka od radnika;
- Scheduler delay je vrijeme kada zadatak čeka da bude zakazan za izvršenje;
- Peak execution memory je maksimalna memorija koju koriste unutrašnje strukture podataka kreirane tokom miješanja, agregiranja i spajanja;
- Shuffle Read Size / Records. Ukupan broj čitanih bajtova nasumice uključuje i podatke koji se čitaju lokalno i podatke čitane sa udaljenih izvršilaca;
- Shuffle Read Fetch Wait Time je vrijeme koje su zadaci proveli blokirani čekajući da se podaci nasumice pročitaju sa udaljenih mašina;
- Shuffle Remote Reads je ukupan broj bajtova nasumice pročitanih sa udaljenih izvršilaca;
- Shuffle Write Time je vrijeme koje su zadaci potrošili na pisanje nasumičnih podataka;
- Shuffle spill (memorija) je veličina deserializovanog oblika izmiješanih podataka u memoriji;
- Shuffle spill (disk) je veličina serijalizovanog oblika podataka na disku.

Kartica Stages Apache Spark u okviru Apache Spark web UI pruža pregled informacija o radnim etapama (engl. stages) u okviru Spark platforme. Radne etape su logičke jedinice izvršavanja zadataka unutar Spark-a, koje se izvršavaju u sklopu određene akcije (engl. action) ili transformacije (engl. transformation) nad RDD ili DataFrame objektima.

Na ovoj kartici možete pratiti informacije o vremenu izvršavanja svake etape, broju zadataka koji se izvršavaju unutar svake etape, kao i o stanju izvršavanja etape (u toku, uspješno završena, neuspješno završena). Osim toga, možete pregledati informacije o korišćenju resursa (CPU i memorije) u okviru svake etape, kao i informacije o podacima koji se prenose između etapa (engl. shuffle) i vrijeme potrebno za prenos podataka.



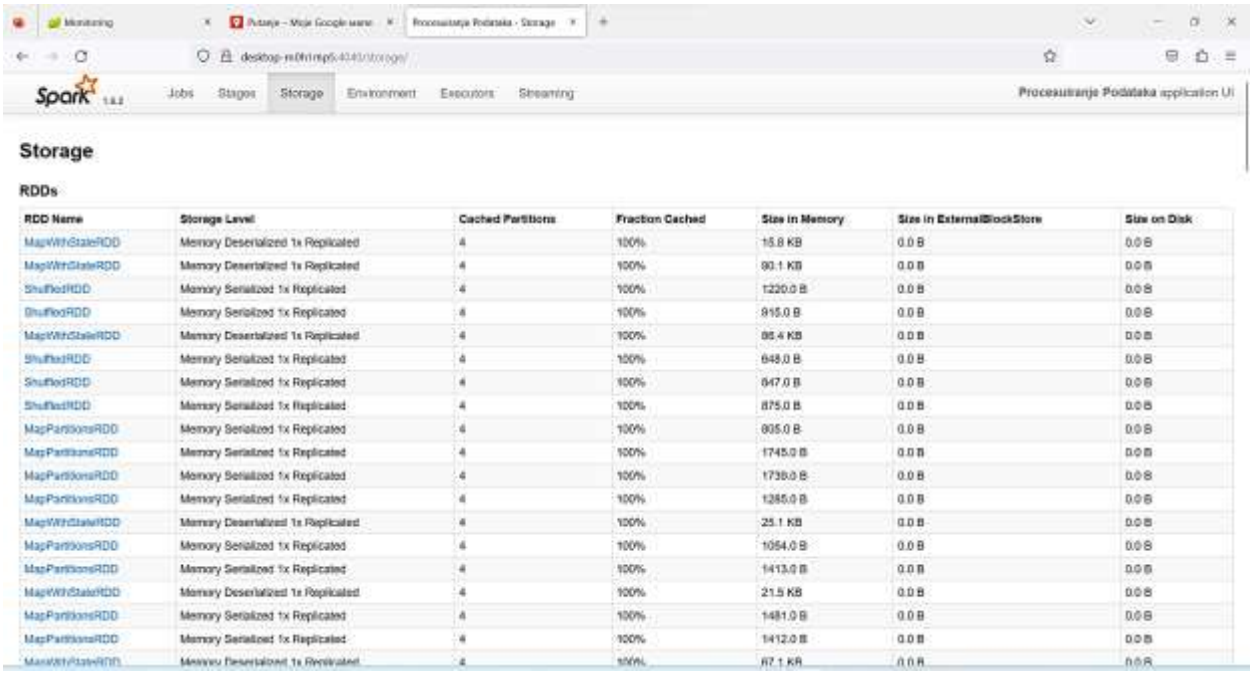
Slika 102: Prikaz Spark Processor Stages

Kartica Storage u Apache Spark web UI pruža informacije o upotrebi memorijskog prostora za skladištenje i distribuciji podataka na čvorovima u klasteru. Pomaže u praćenju upotrebe memorijskog prostora za skladištenje, identifikaciji potencijalnih curenja memorije i optimizaciji korištenja memorijskog prostora.

Kartica Storage prikazuje sljedeće informacije:

1. RDD-ovi: prikazuje listu RDD-ova koji su trenutno u memoriji, njihovu veličinu i broj particija.
2. Memorija za skladištenje: prikazuje količinu memorije koju koristi skladištenje, dostupnu količinu memorije i količinu memorije korišćenu za predmemorisanje.
3. Izvršitelji: prikazuje broj izvršitelja i njihovu upotrebu memorije. Takođe prikazuje broj zadataka koji se trenutno izvršavaju na svakom izvršitelju.
4. Upravljanje blokovima: prikazuje informacije o upotrebi memorijskog prostora za skladištenje, poput količine memorije korištene za predmemorisanje, količine memorije korišćene za skladištenje i količine memorije korišćene za razvrstavanje.

Praćenjem upotrebe memorijskog prostora za skladištenje i optimizacijom korišćenja memorijskog prostora, možemo osigurati da se Spark zadaci izvršavaju učinkovito i bez problema sa memorijom.



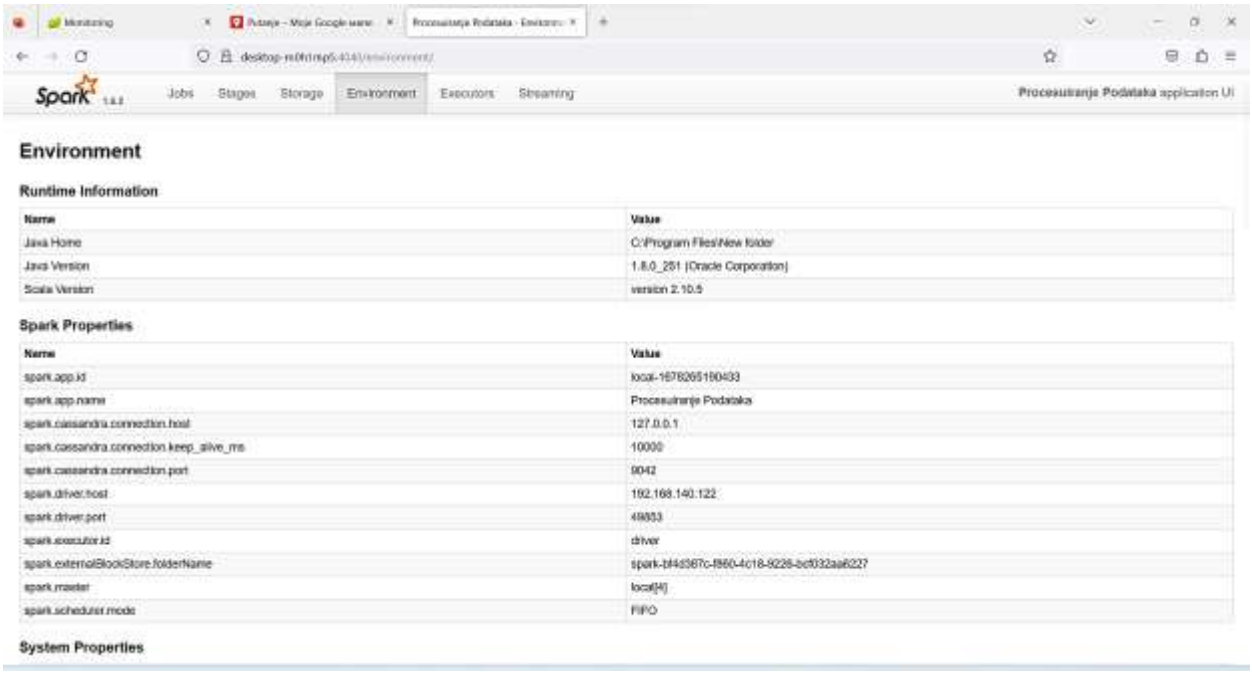
Slika 103: Prikaz Spark Processor Storage

Kartica Environment u Apache Spark web korisničkom interfejsu prikazuje informacije o konfiguraciji okruženja u kojem se Spark izvršava. Ova kartica je korisna za provjeru da li su svi potrebni resursi dostupni i da li su podešavanja pravilno konfigurisana.

Kartica Environment prikazuje sljedeće informacije:

- Spark Properties: Prikazuje listu postavki konfigurisanih za Spark i njihove vrijednosti.
- Environment Variables: Prikazuje listu varijabli okruženja konfigurisanih za Spark i njihove vrijednosti.
- Classpath Entries: Prikazuje listu putanja do biblioteka koje su dostupne Spark-u za izvršavanje.
- System Properties: Prikazuje listu sistemskih svojstava konfigurisanih za Spark i njihove vrijednosti.
- Other Resources: Prikazuje listu ostalih resursa, kao što su Python biblioteke, konfiguracione datoteke itd.

Prikazane informacije na kartici Environment su korisne za rješavanje problema u radu sa Spark-om i optimizaciju konfiguracije za postizanje najboljih performansi.



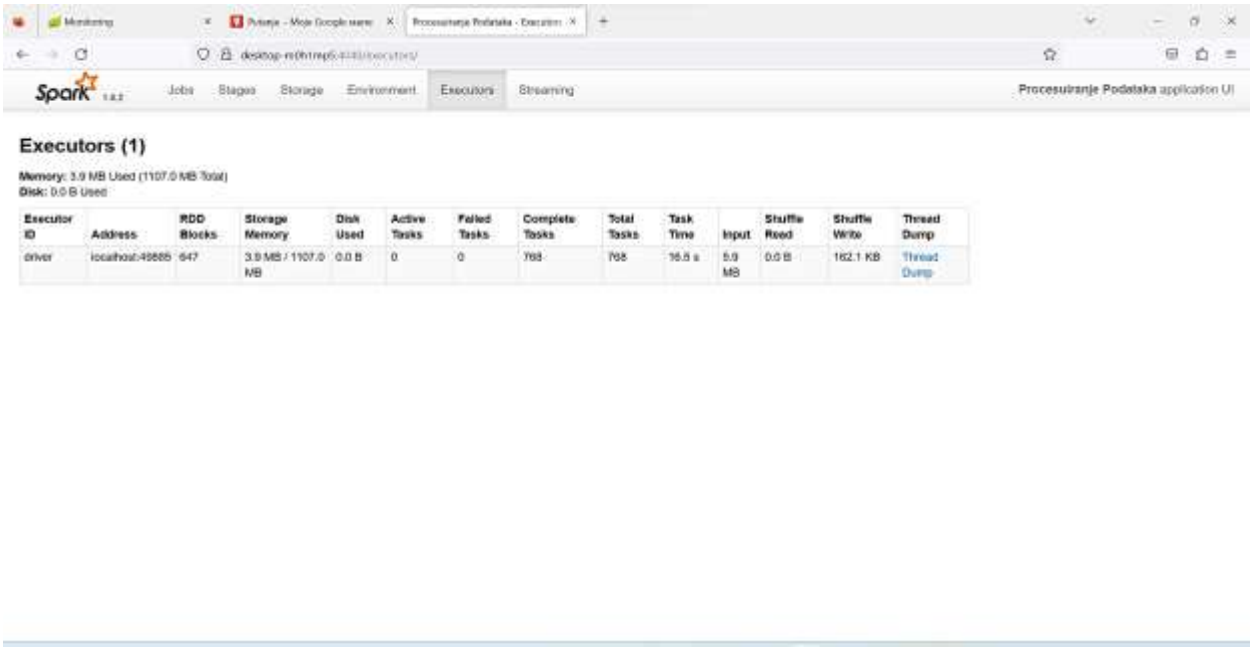
Slika 104: Prikaz Spark Processor Environment

Kartica Executors u Apache Spark web UI prikazuje informacije o izvršiocima (eng. Executors) koji se koriste za obradu zadataka u klasteru. To uključuje informacije o broju izvršioca, njihovoj memoriji, broju zadataka koje trenutno izvršavaju i njihovoj upotrebi resursa.

Konkretno, kartica Executors uključuje sljedeće informacije:

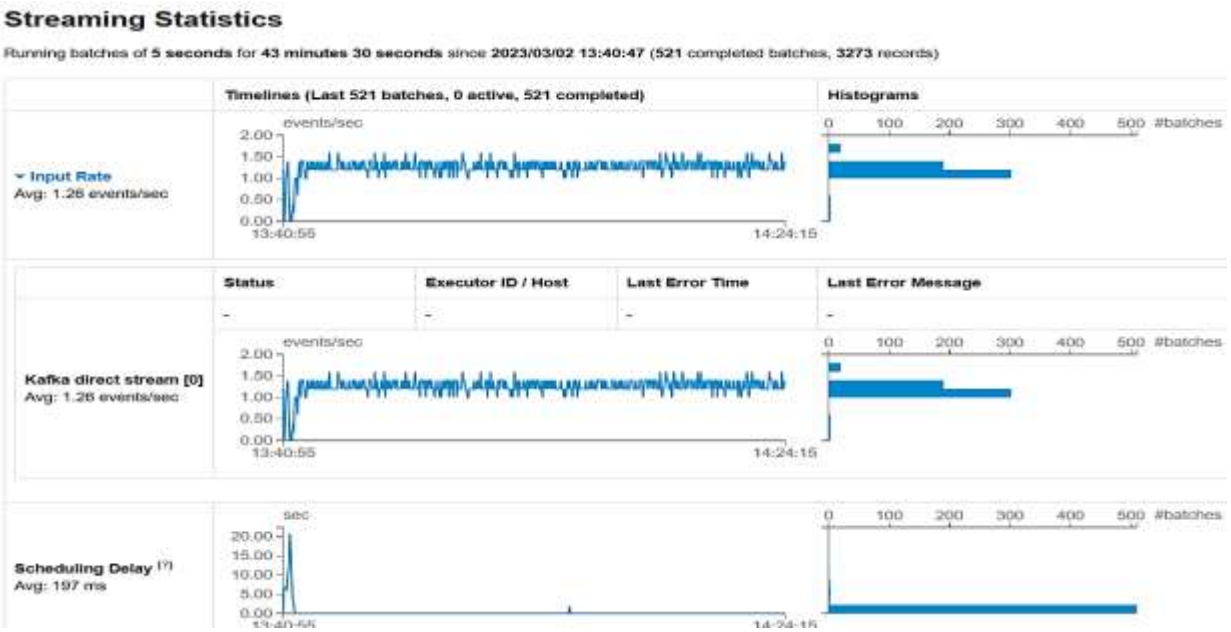
- Broj izvršilaca: Prikazuje ukupan broj izvršilaca u klasteru, kao i broj aktivnih izvršilaca.
- Memorija izvršilaca: Prikazuje ukupnu količinu memorije koju koriste izvršioci u klasteru, kao i koliko memorije je dostupno.
- Korisnički zadaci: Prikazuje trenutno izvršavane zadatke po izvršiocu, kao i vremensku statistiku o izvršavanju svakog zadatka.

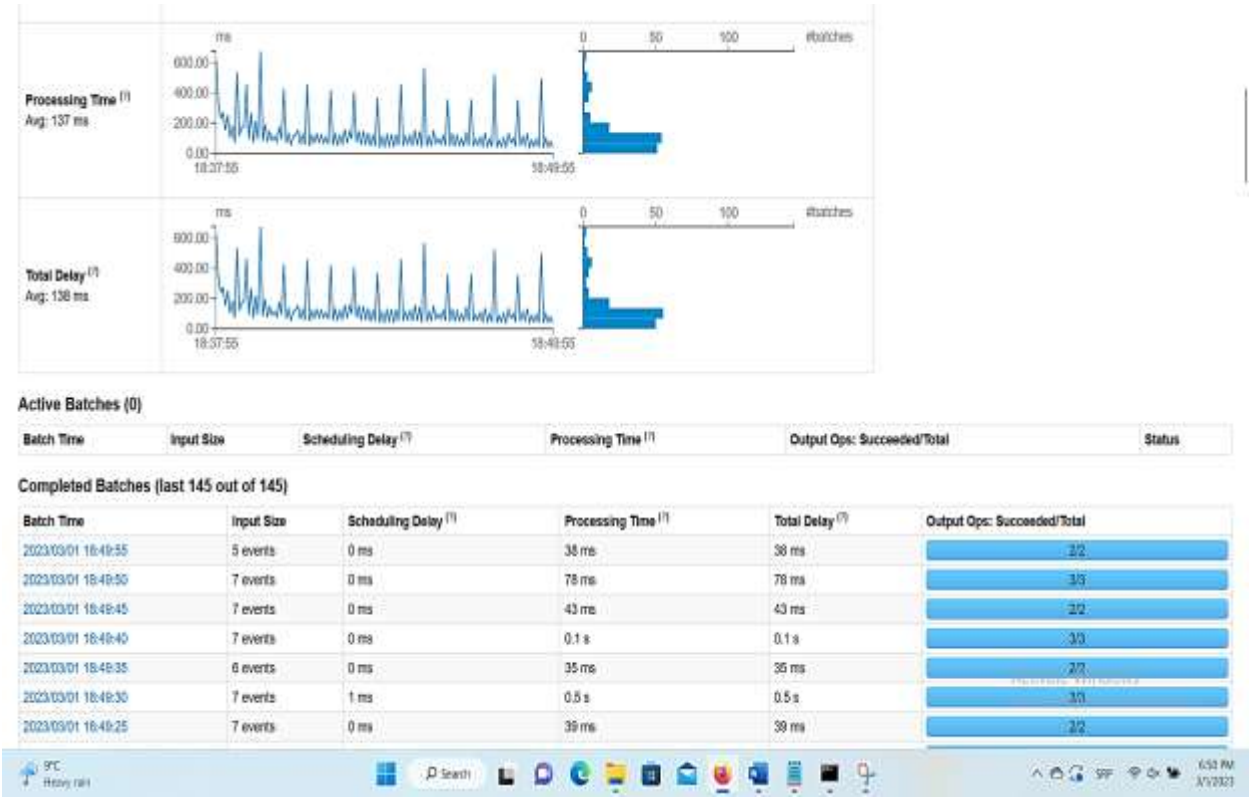
Kartica Executors pruža korisne informacije o izvršiocima u klasteru, što pomaže u praćenju performansi klastera i identifikaciji problema s izvršiocima koji mogu usporiti obradu podataka.



Slika 105: Prikaz Spark Processor Executors

Kartica Streaming u Apache Spark web UI-u pruža informacije o statusu streaming platforme, brzini obrade i detaljima obrade u paketima. To pomaže u praćenju streaming platforme i identifikaciji potencijalnih problema koji se mogu pojaviti tokom procesa streaminga.





Slika 106: Prikaz Spark Processor Streaming

Kartica Streaming prikazuje sljedeće informacije:

1. Batch Duration: prikazuje trajanje svakog paketa u streaming platformi.
2. Input Rate: prikazuje stopu kojom se ulazni podaci primaju u streaming platformu.
3. Processing Time: prikazuje vrijeme potrebno za obradu svakog paketa.
4. Scheduling Delay: prikazuje vrijeme potrebno za zakazivanje sljedećeg paketa.
5. Processing Details: prikazuje detalje svakog paketa, kao što su vrijeme obrade, stopa ulaza, broj obrađenih zapisa i bilo kakve greške koje su se pojavile.

Praćenjem kartice Streaming, možemo osigurati da streaming platforma radi efikasno i bez problema. Također, možemo identifikovati bilo kakva ograničenja i optimizovati obradu kako bismo poboljšali performanse streaming platforme.

6. Rezultati i analiza

Naša platforma se sastoji od dva glavna dijela: server baze podataka i vizuelni dio.

Server baza podataka se koristi za prikupljanje i skladištenje podataka o saobraćaju, uključujući informacije o vozilima, putanjama i vremenima putovanja. Ovi podaci se kontinuirano ažuriraju kako bi se osigurala tačnost i pouzdanost podataka.

Vizuelni dio platforme koristi ove podatke za prikazivanje informacija o saobraćaju na pregledan i razumljiv način. Korisnicima se omogućava da pregledaju informacije o saobraćaju na različitim lokacijama i putanjama, koristeći interaktivnu mapu i grafičke prikaze. Takođe se pružaju prozori za filtriranje podataka i analizu saobraćajnih tokova, što omogućava donosiocima odluka da donose bolje i informisanije odluke.

Kombinacija server baze podataka i vizuelnog dijela platforme pruža sveobuhvatno rješenje za upravljanje saobraćajem, koje je fleksibilno i prilagodljivo za različite potrebe korisnika.

U ovom radu smo razvili funkcionalnu platformu za praćenje saobraćaja i podršku održivoj mobilnosti.

Objašnjenje arhitekture platforme:

IoT senzori: IoT senzori su postavljeni duž puteva i koriste se za prikupljanje podataka o saobraćaju. Oni mogu mjeriti brzinu vozila, broj vozila na putu, gustinu saobraćaja i druge parametre. Podaci se prikupljaju u stvarnom vremenu i šalju dalje na obradu. U cilju prikupljanja podataka, korišćeni su unaprijed generisani podaci, zbog nedovoljno dostupnih izvora.

Kafka: Kafka je distribuirana platforma za obradu podataka u stvarnom vremenu. Podaci koji su prikupljeni od strane IoT senzora se šalju na Kafka topic, koji funkcioniše kao ulazna tačka sistema. Kafka omogućava brz i pouzdan protok podataka između različitih komponenti sistema.

Spark Streaming: Spark Streaming je tehnologija koja omogućava obradu podataka u stvarnom vremenu. Podaci koji su pristigli na Kafka topic se preuzimaju od strane Spark Streaminga i dalje se obrađuju za potrebe sistema za praćenje saobraćaja. Spark Streaming pruža mogućnost obrade podataka u realnom vremenu, omogućavajući analizu, agregaciju i transformaciju podataka.

Baza podataka: Obradjeni podaci se mogu skladištiti u bazu podataka radi dalje analize i prikaza. Baza podataka omogućava dugoročno čuvanje podataka o saobraćaju i omogućava korisnicima da pristupe istorijskim podacima i analiziraju trendove u saobraćaju.

Korisnički interfejs: Web platforma ima intuitivan korisnički interfejs koji se sastoji

od grafikona, tabela i mape, koji omogućavaju da se vizualizuju podaci o saobraćaju. Interfejs prikazuje različite informacije o saobraćaju, uključujući ukupan saobraćaj, vrste vozila, putanje i vrijeme kretanja vozila.

Objašnjenje funkcionalnosti platforme:

Prikaz ukupnog saobraćaja: Korisnici mogu pregledati ukupan saobraćaj na odabranom području. Ovaj prikaz može biti koristan za donosioce odluka u planiranju i upravljanju saobraćajem.

Prikaz vrsta vozila: Korisnici mogu vidjeti koje vrste vozila su prisutne na putevima, kao što su automobili, kamioni, autobusi i sl. Ova informacija može pomoći u analizi prometa i planiranju saobraćajnih odluka.

Prikaz putanja: Korisnici mogu vidjeti putanje kojima se vozila kreću na mapi. Ovo može pomoći u identifikaciji glavnih prometnih putanja i potencijalnih zona zagušenja.

Prikaz vremena kretanja vozila: Korisnici mogu vidjeti koliko vremena je potrebno vozilima da se kreću između određenih tačaka. Ova informacija može biti korisna za planiranje putovanja i izbjegavanje zagušenja.

Takođe, platforma će omogućiti detaljan pregled pojedinačnih putanja sa mogućnošću prebacivanja na druge putanje, kao i pregled saobraćaja u posljednjih nekoliko minuta. Ovo omogućava donosiocima odluka da reaguju na zagušenje saobraćaja i ubrzaju vrijeme putovanja.

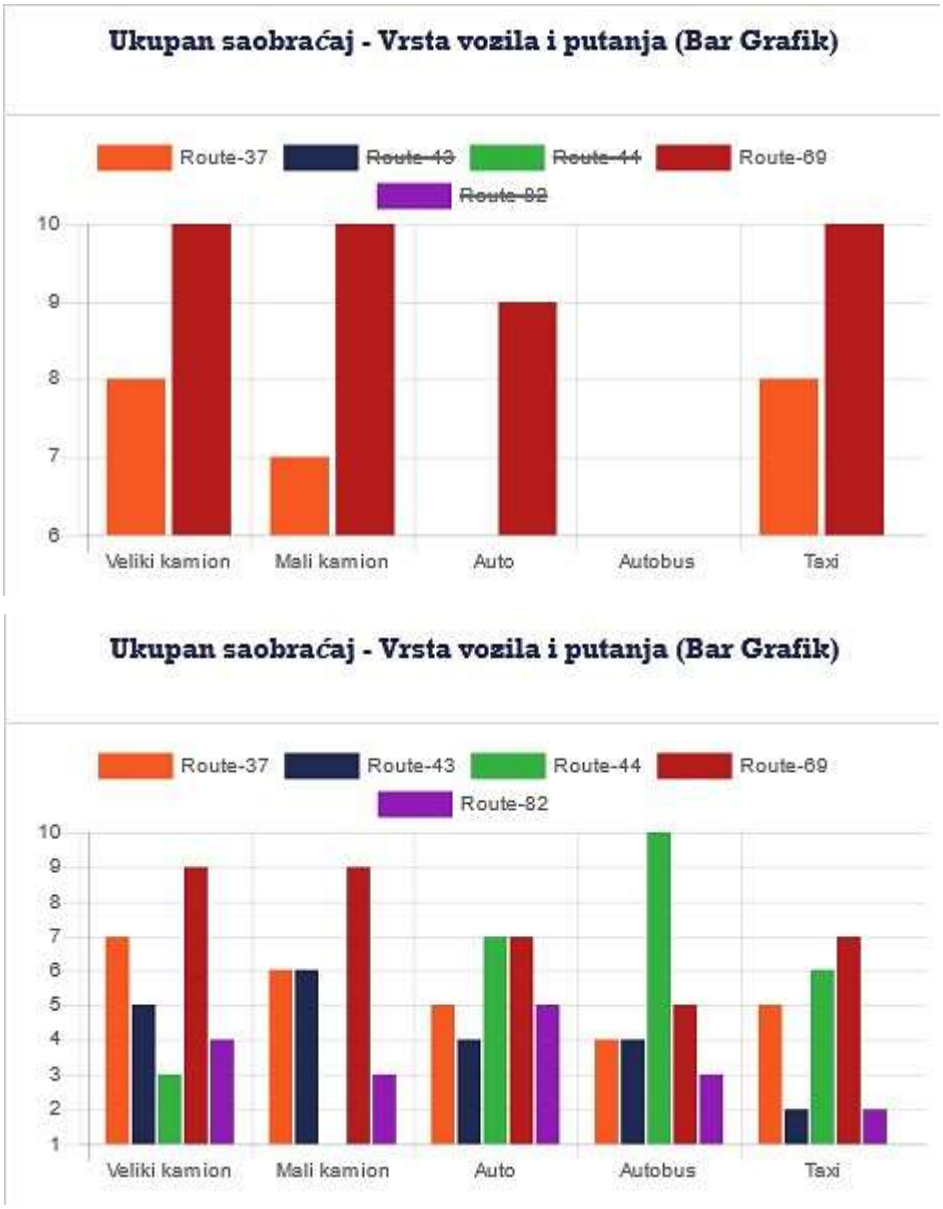
Platforma će takođe pružati informacije o udaljenosti vozila od tačaka interesa, putanjama koje su prekinute ili neprohodne. Ova informacija omogućava donosiocima odluka da brzo i efikasno reaguju na prepreke na putu i obezbijede neometan transport.

Kroz sve ove funkcionalnosti, web platforma za praćenje saobraćaja pruža korisnicima alat za planiranje putovanja i unapređenje održive mobilnosti. Platforma će biti lako dostupna donosiocima odluka, uz intuitivan i jednostavan korisnički interfejs koji omogućava lako pristupanje i analizu relevantnih informacija o saobraćaju.

Platforma se sastoji od tri modula sa jedinstvenim funkcionalnostima, kao i modula za prikaz putem karte.

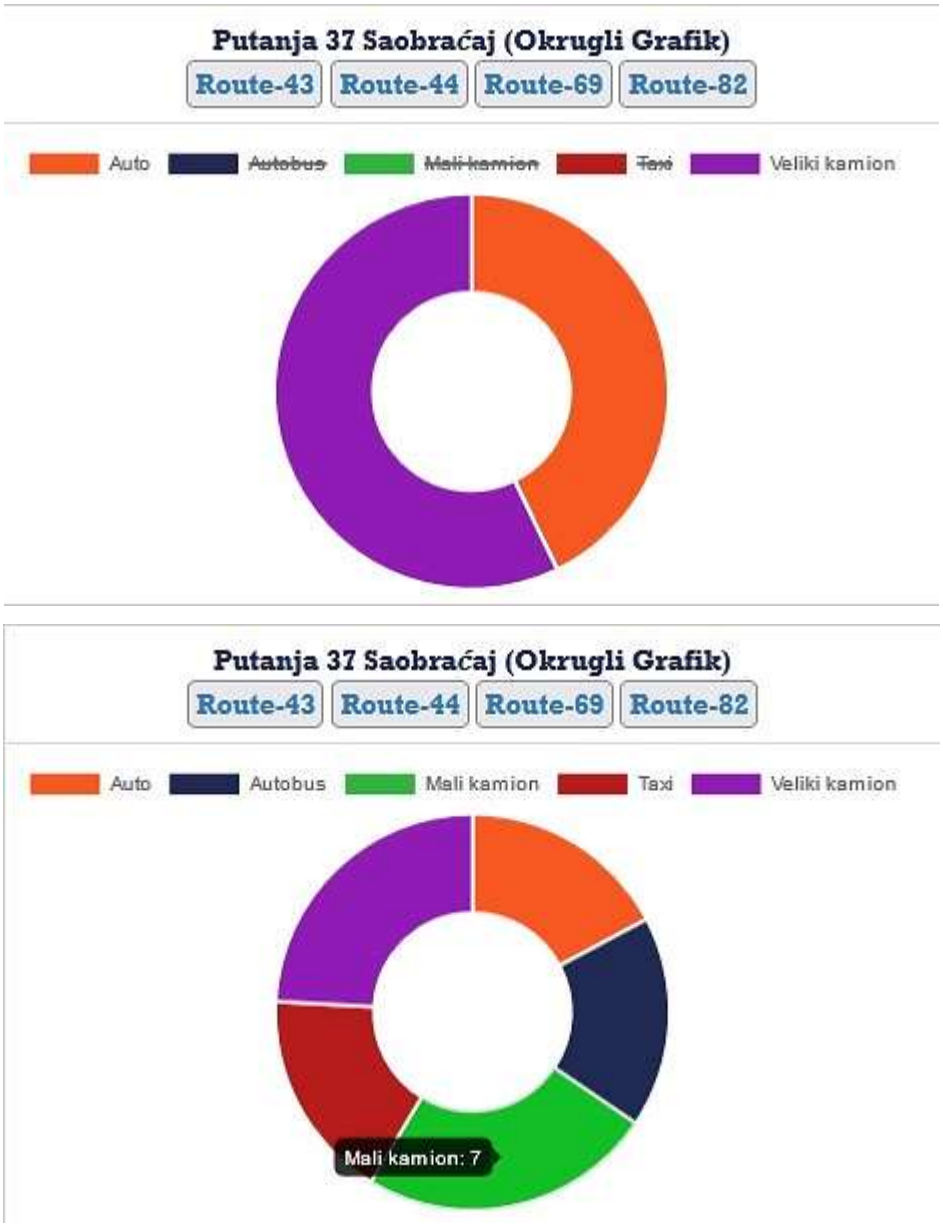
U nastavku su detaljnije opisani moduli i njihove funkcionalnosti:

Modul za ukupan saobraćaj: Ovaj modul prikazuje informacije o ukupnom saobraćaju, vrstama vozila, putanjama kojima se vozila kreću i vremenima kretanja vozila. Pruža dublje razumijevanje saobraćajnih tokova i pomaže u planiranju efikasnijeg transporta.



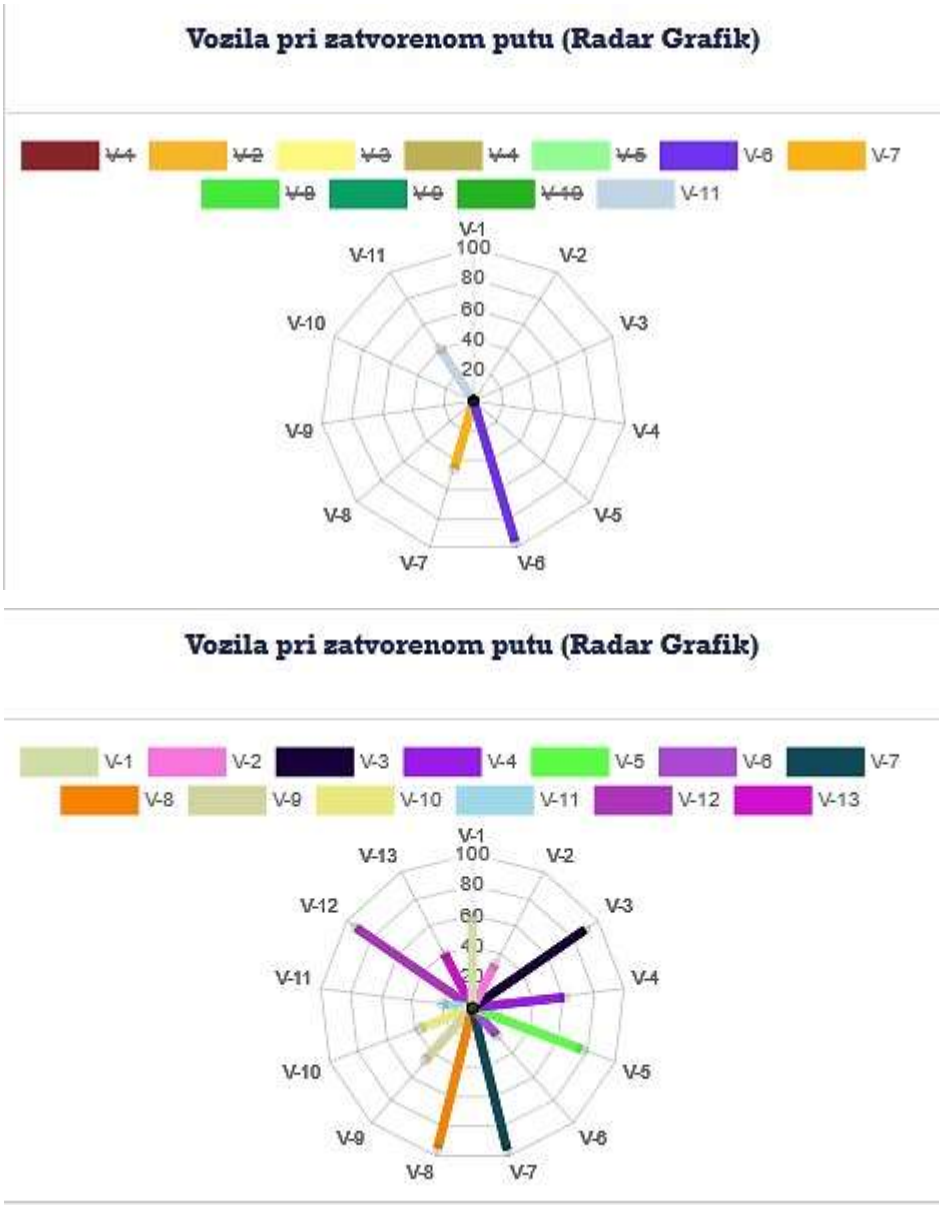
Slika 107: Prikaz modula za ukupan saobraćaj

Modul za detaljan pregled putanja: Ovaj modul omogućava detaljan pregled pojedinačnih putanja sa mogućnošću prebacivanja na druge putanje. Takođe pruža pregled saobraćaja u posljednjih 5 minuta. Ovaj modul je dragocjeni alat za donosioce odluka koji žele brzo reagovati na zagušenje saobraćaja i smanjiti vrijeme putovanja između dvije tačke.



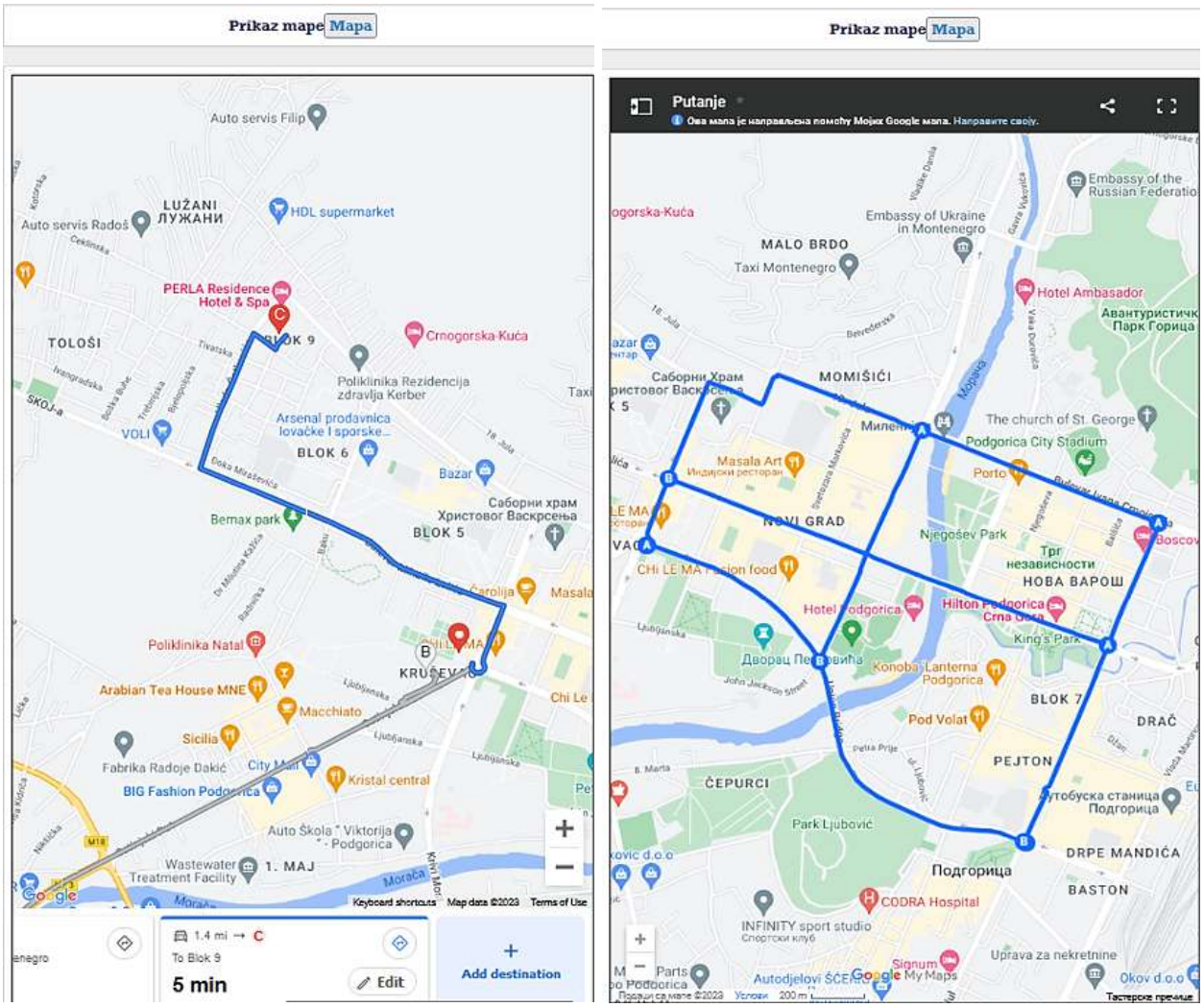
Slika 108: Prikaz modula za pregled putanja

Modul za prepreke na putu: Ovaj modul se fokusira na udaljenost vozila od tačke interesa i prikazuje putanje koje su prekinute ili neprohodne. Ove informacije omogućavaju donosiocima odluka da brzo i efikasno reagiraju na prepreke na putu i osiguraju nesmetan transport.



Slika 109: Prikaz modula za prepreke na putu

Modul za Google Maps API: Platforma takođe sadrži modul koji prikazuje Google Maps API sa mogućnostima odabira putanja i prikazivanja vremena potrebnog za putovanje od tačke A do tačke B. Ovaj modul omogućava korisnicima da planiraju putovanje na najefikasniji način, pružajući detaljne informacije o vremenskom trajanju putovanja na različitim rutama.



Slika 110: Prikaz modula za Google Maps API

Naša platforma pruža nekoliko ključnih prednosti i doprinosa:

- Vizualizacija podataka o saobraćaju kroz interaktivni korisnički interfejs omogućava korisnicima lako pristupanje i analizu relevantnih informacija.
- Informacije o saobraćaju koje se nude na platformi omogućavaju donosiocima odluka brzo i efikasno donošenje odluka o saobraćaju.
- Efikasnost i skalabilnost platforme omogućuju rukovanje velikom količinom podataka koje se prikupljaju i obrađuju.

Ova platforma predstavlja alat za planiranje putovanja i unapređenje održive mobilnosti, pružajući lako dostupne informacije i intuitivan korisnički interfejs.

Ukupan saobraćaj - Vrsta vozila i putanja (Tabelarno)

Route	Vehicle	Count	Time
Route-69	Auto	13	2023-06-14 18:06:25
Route-69	Autobus	9	2023-06-14 18:09:10
Route-69	Mali kamion	10	2023-06-14 18:07:25
Route-69	Taxi	12	2023-06-14 18:08:25
Route-69	Veliki kamion	11	2023-06-14 18:06:05
Route-44	Auto	10	2023-06-14 18:06:15
Route-44	Autobus	12	2023-06-14 18:07:55
Route-44	Mali kamion	6	2023-06-14 18:09:10
Route-44	Taxi	11	2023-06-14 18:08:00
Route-44	Veliki kamion	8	2023-06-14 18:08:50
Route-37	Auto	9	2023-06-14 18:08:50
Route-37	Autobus	8	2023-06-14 18:06:40
Route-37	Mali kamion	10	2023-06-14 18:08:50
Route-37	Taxi	9	2023-06-14 18:08:15
Route-37	Veliki kamion	12	2023-06-14 18:09:15
Route-82	Auto	12	2023-06-14 18:

Poslednjih 5 minuta prozor (Tabelarno)

Route	Vehicle	Count	Time
Route-69	Auto	13	2023-06-14 18:09:10
Route-69	Autobus	9	2023-06-14 18:09:10
Route-69	Mali kamion	10	2023-06-14 18:09:10
Route-69	Taxi	12	2023-06-14 18:09:10
Route-69	Veliki kamion	11	2023-06-14 18:09:10
Route-44	Auto	10	2023-06-14 18:09:10
Route-44	Autobus	12	2023-06-14 18:09:10
Route-44	Mali kamion	6	2023-06-14 18:09:10
Route-44	Taxi	11	2023-06-14 18:09:10
Route-44	Veliki kamion	8	2023-06-14 18:09:10
Route-37	Auto	9	2023-06-14 18:09:10
Route-37	Autobus	8	2023-06-14 18:09:10
Route-37	Mali kamion	10	2023-06-14 18:09:10
Route-37	Taxi	9	2023-06-14 18:09:10
Route-37	Veliki kamion	11	2023-06-14 18:09:10
Route-82	Auto	12	2023-06-14 18:

Kamioni pri zatvorenom putu (Tabelarno)

Vehicle Id	Vehicle	Distance	Time
0687b56d-2939-49e6-954f-c3184909be4d	Mali kamion	68	2023-06-14 18:09:00
5d2b9af2-4c26-452c-a4bc-b1044f8d32ae	Veliki kamion	49	2023-06-14 18:08:35
4a1eed9d-57d7-4056-85e2-19f2656414f0	Veliki kamion	41	2023-06-14 18:07:45
ebaf36a9-5e3f-47ff-b691-1f94ce45b236	Veliki kamion	65	2023-06-14 18:08:30
af5cf31f-b024-40be-90c6-030e4b74f9b2	Mali kamion	14	2023-06-14 18:08:50
b776252e-4d54-477f-bd66-cba98d027492	Veliki kamion	96	2023-06-14 18:07:30
39743c66-8319-4c7f-96e2-a1940f660943	Mali kamion	44	2023-06-14 18:09:00
12492031-ec0b-42f9-8a27-25ffbaec07c0	Veliki kamion	27	2023-06-14 18:09:15
314b43b9-6cfb-449f-ad66-1f21f2537f53	Veliki kamion	65	2023-06-14 18:09:00
5a2563a5-2756-484d-8933-40ea745c3819	Mali kamion	95	2023-06-14 18:08:30
8bd05989-f655-4974-92b9-ca93c4810255	Mali kamion	70	2023-06-14 18:08:55

Slika 111: Prikaz modula preko tabelarnog prikaza

Ukupan saobraćaj: vrste vozila i putanje: Na kontrolnoj tabli prikazuje se tabela koja prikazuje ukupan saobraćaj na različitim putanjama i vrstama vozila. Svaka putanja i vrsta vozila predstavljene su u obliku reda i kolone u tabeli. Ovaj prikaz omogućava korisnicima da dobiju pregled o tome koliko vozila prolazi kroz određenu putanju i koje vrste vozila su najzastupljenije. Na taj način, korisnici mogu identifikovati prometne čvorove i područja s većim opterećenjem.

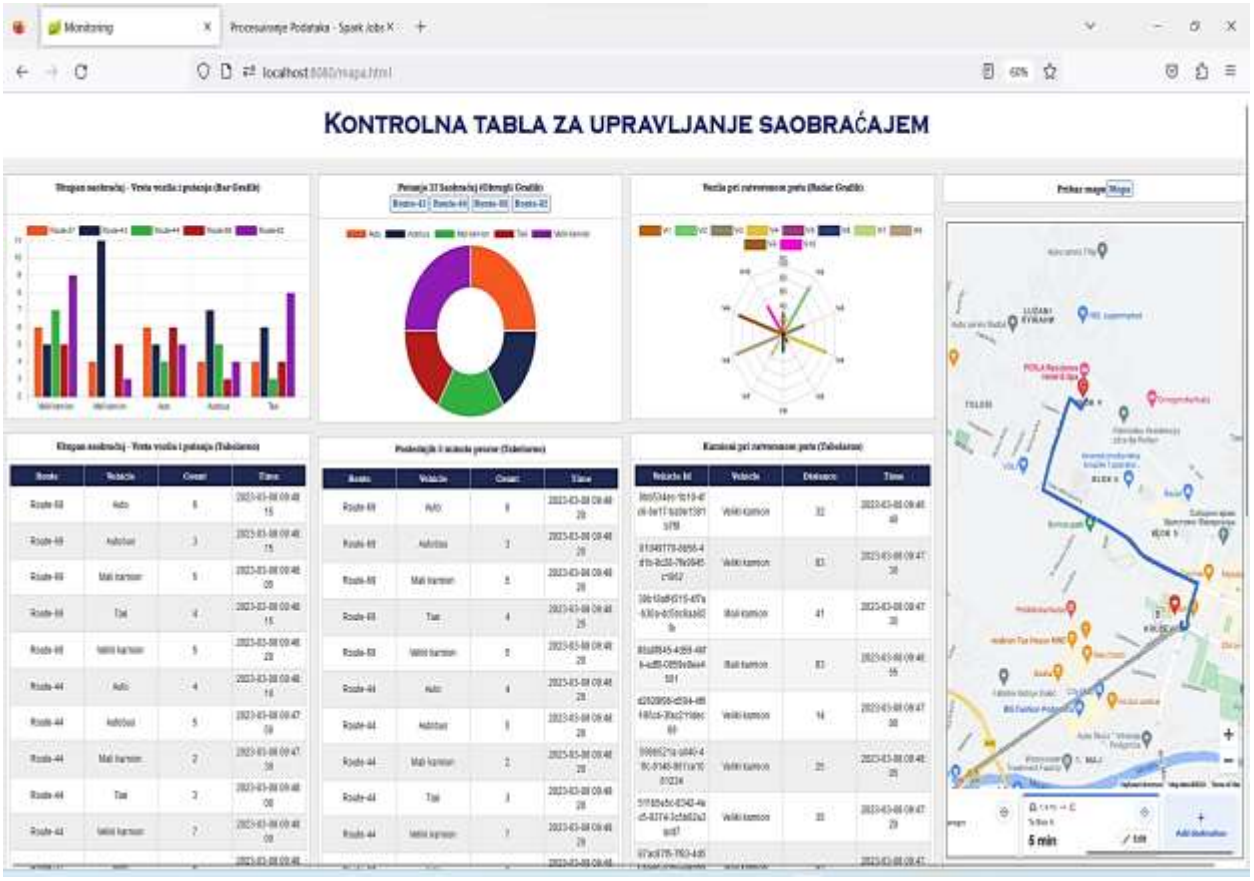
Broj vozila i vrijeme: Tabela takođe prikazuje broj vozila koji prolazi kroz određene putanje i vremenske oznake. Ova informacija omogućava korisnicima da prate promjene u broju vozila tokom vremena. Takođe, vremenske oznake mogu biti korisne za analizu saobraćajnih gužvi i planiranje putovanja u skladu s tim.

Prozor od poslednjih pet minuta: Pored ukupnog prikaza saobraćaja, platforma ima i prozor koji prikazuje saobraćaj u poslednjih pet minuta. Ovaj prozor omogućava korisnicima da

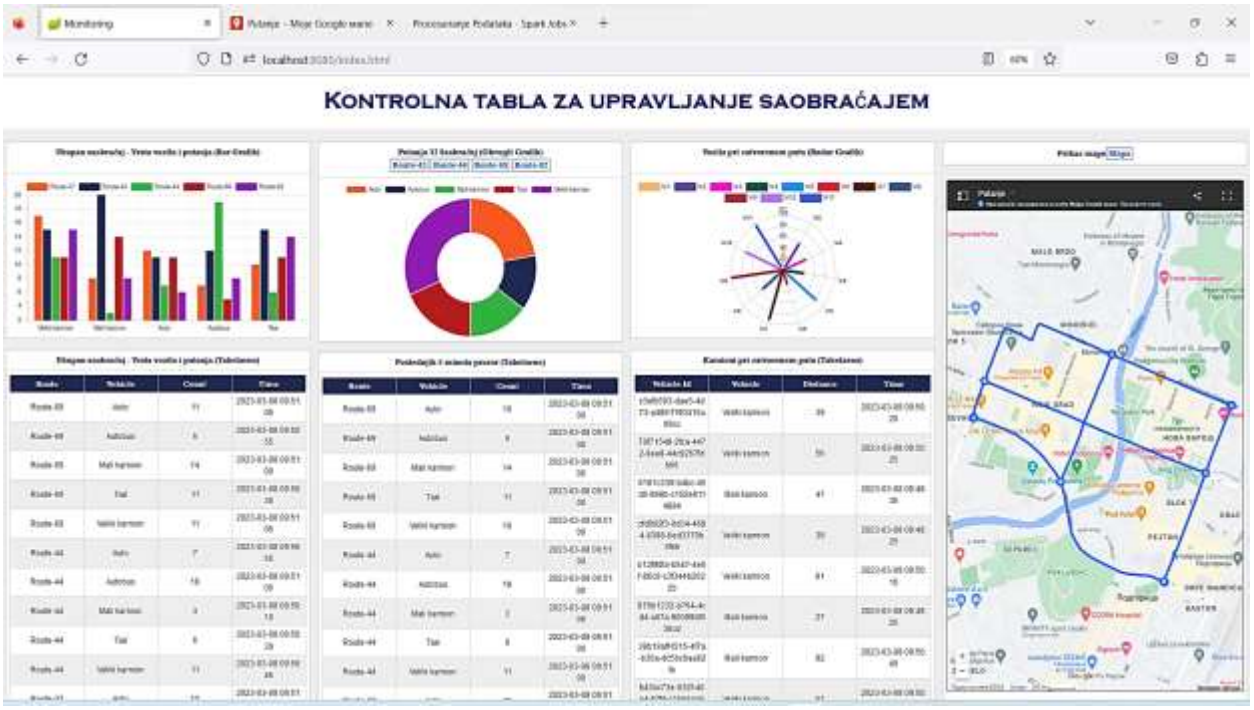
vide trenutno stanje saobraćaja i brzo reaguju na eventualne promjene ili zagušenja. Na taj način, korisnici mogu donositi odluke u realnom vremenu kako bi optimizovali svoje putovanje ili upravljali saobraćajem.

Prozor Zatvoreni put: Ovaj prozor prikazuje udaljenost određene vrste vozila do zatvorenog puta. Kada postoji zatvorena putanja ili prepreka na putu, korisnici mogu vidjeti koliko su udaljeni od tog zatvorenog puta. Ova informacija pomaže donosiocima odluka da brzo identifikuju alternative putanje i prilagode svoje rute kako bi izbjegli prepreke i zagušenja.

Ove funkcionalnosti omogućavaju korisnicima da imaju detaljan pregled o saobraćaju i da donose informisane odluke. Pored toga, korisnici mogu koristiti opciju izbora putanja i prikaza vremena putovanja od tačke A do tačke B pomoću Google Maps API-ja. Ovaj modul omogućava korisnicima da planiraju svoje putovanje na najefikasniji način.



Slika 112: web platforma za upravljanje saobraćajem

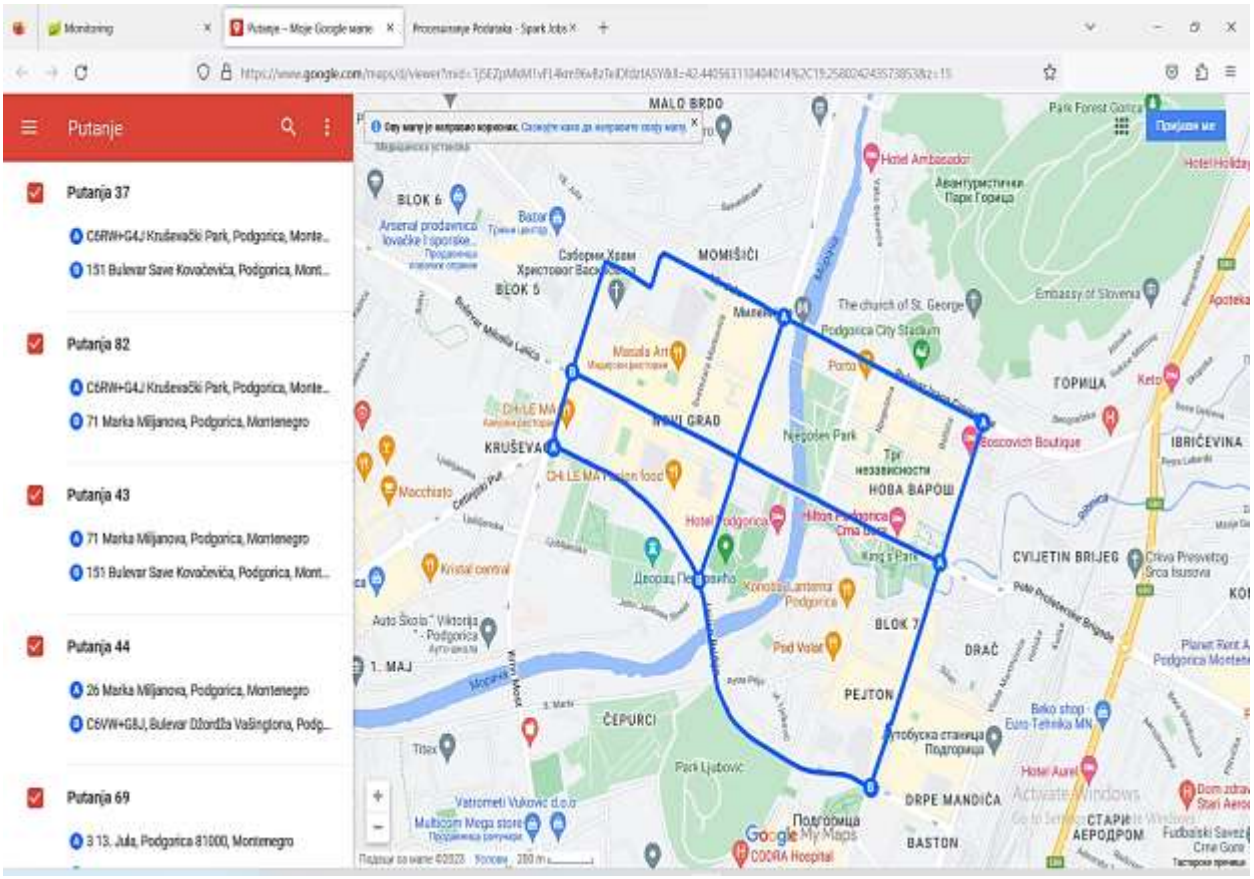


Slika 113: web platforma uz odabrane putanje

Naša platforma također nudi mogućnost otvaranja i prikaza na većem prozoru, što omogućava detaljniji pregled i prilagođavanje individualnim potrebama korisnika. Ovim prilagođavanjem korisnici mogu dublje analizirati putanje, vidjeti sve važne detalje na putanji i donijeti odluke u skladu sa svojim potrebama. Ova opcija omogućava korisnicima da bolje razumiju svoje okruženje i da donose pametnije odluke u vezi sa svojim putovanjima, uključujući smanjenje vremena putovanja, smanjenje gužvi na cestama i smanjenje emisija štetnih gasova.

Naša platforma ima za cilj povećanje efikasnosti i efektivnosti upravljanja saobraćajem. Korištenjem najsavremenijih tehnologija i alata za prikupljanje, analizu i vizualizaciju podataka o saobraćaju, korisnicima se omogućava da dobiju informacije o saobraćaju u realnom vremenu i prilagode svoje putanje u skladu sa najnovijim podacima.

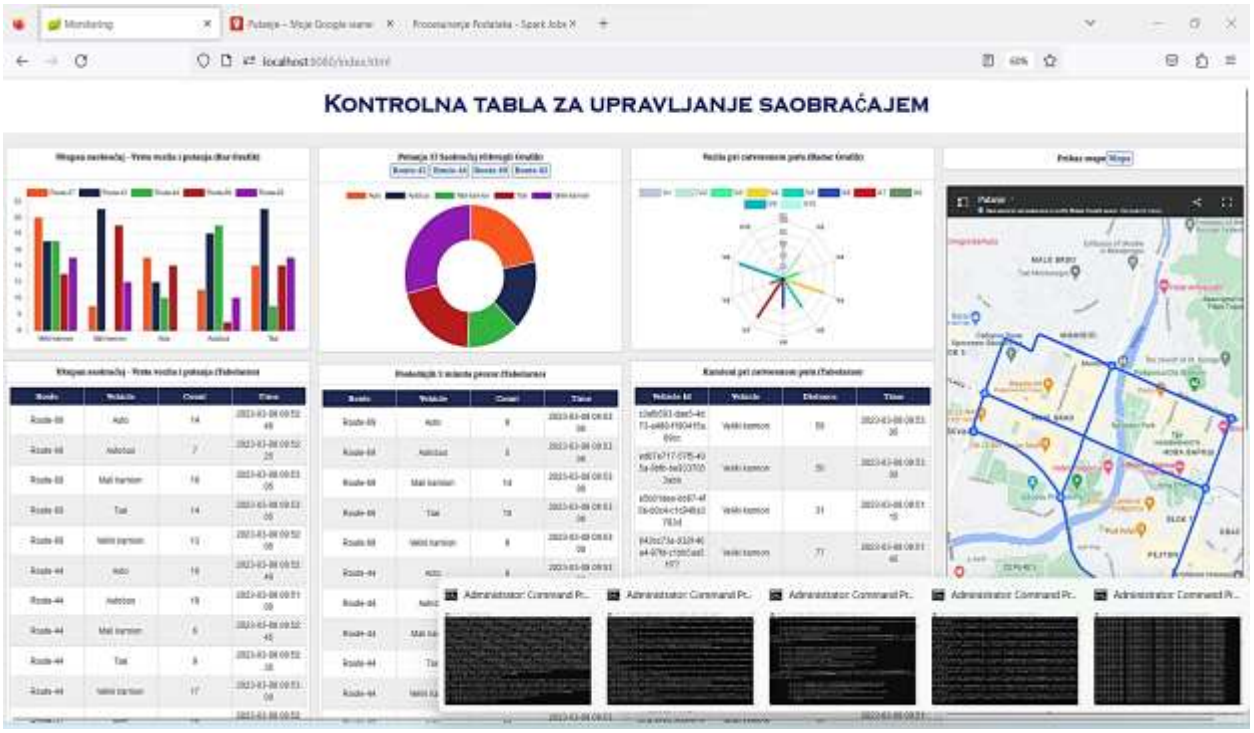
Ovo omogućava brže i efikasnije putovanje, smanjenje vremena potrebnog za putovanje i smanjenje stresa zbog gužvi u saobraćaju. Uz to, naša platforma predstavlja dobru podršku donosiocima odluka u planiranju održive mobilnosti, što doprinosi smanjenju emisija štetnih gasova i stvaranju čistijeg i zdravijeg okruženja. Kao takva, naša platforma ima potencijal da donese značajne pozitivne promjene u našim gradovima i društvima.



Slika 114: Detaljan prikaz putanja

Na slici 114. možete vidjeti simulaciju naše platforme koja se primjenjuje u gradu Podgorici. Odabrali smo pet različitih putanja koje su označene kao Putanja 37, Putanja 82, Putanja 43, Putanja 44 i Putanja 69. Putanja 37 se kreće od kružnog kod Dječjeg parka na Kruševcu do Bulevara Save Kovačevića, Putanja 82 od Dječjeg parka na Kruševcu do 71 Marka Miljanova, Putanja 43 od 71 Marka Miljanova do 151 Bulevara Save Kovačevića, Putanja 44 od 26 Marka Miljanova do Bulevara Džordža Vašingtona, a Putanja 69 od 13. Jula do 6 Džona Džeksona.

Naša platforma omogućava pojedinačni pregled svake putanje, kao i pregled svih važnih tačaka na tim putanjama. Takođe, simulacija na slici 114. prikazuje kako naša platforma zapravo funkcioniše, pružajući donosiocima odluka i planerima potrebne informacije za efikasno planiranje puta i optimizaciju vremena putovanja. Sve ove mogućnosti čine našu platformu idealnim alatom za planiranje putovanja i unapređenje održive mobilnosti u gradovima poput Podgorice.



Slika 115: Prikaz platforme uz otvorene pozadinske alate

Postoji niz mogućih prijedloga za buduća istraživanja. Neke od mogućnosti uključuju:

1. Integracija drugih vrsta podataka: Iako ovo istraživanje uključuje integraciju različitih vrsta podataka o saobraćaju, postoji mogućnost integracije drugih vrsta podataka, poput meteoroloških podataka, podataka o kvalitetu vazduha, podataka o gužvama u javnom prevozu, itd. Integracija ovih dodatnih podataka može omogućiti preciznije prognoziranje saobraćaja i pružanje korisnijih informacija donosiocima odluka.
2. Razvoj novih tehnologija za prikupljanje podataka o saobraćaju: Postoji mogućnost razvoja novih tehnologija za prikupljanje podataka, poput novih senzora, čipova za automobilsku industriju, itd. Integracija ovih novih tehnologija može omogućiti bolje prikupljanje podataka o saobraćaju i pružanje korisnijih informacija.
3. Analiza uticaja promjena u saobraćaju: Buduća istraživanja mogu se fokusirati na analizu uticaja promjena u saobraćaju, poput izgradnje novih puteva ili uvođenja novih vozila u saobraćaj. Ova analiza može pružiti korisne informacije o efektima promjena u saobraćaju i pomoći u planiranju budućih promjena.

7. Zaključak

U našem istraživanju, uspješno smo razvili platformu za monitoring saobraćaja. U radu su postignuti sljedeći ciljevi: detaljno je opisan sistem za nadgledanje saobraćaja koji koristi Kafka i Spark Streaming tehnologije. Prikazan je arhitekturalni model sistema i objašnjena uloga svake komponente. Takođe, istaknuti su različiti aspekti sistema kao što su kako se ove tehnologije mogu integrirati za efikasno prikupljanje, obradu i analizu podataka o saobraćaju.

Kroz implementaciju četiri modula, uspješno smo ostvarili ciljeve naše platforme. Moduli pružaju detaljne informacije o ukupnom saobraćaju, vrstama vozila, putanjama kretanja vozila i vremenu kretanja. Takođe omogućavaju pregled izdvojenih putanja, tabelarni prikaz saobraćaja u posljednjih 5 minuta, informacije o udaljenosti vozila od tačke interesa i putanji koje su u prekidu. Integrisali smo Google Maps API za odabir putanja i prikaz vremenskog trajanja putovanja od tačke A do tačke B.

Naša platforma uspješno odgovara na istraživačka pitanja koja smo postavili. Detaljno su opisane funkcionalnosti i uloge svake tehnologije, kao i njihova međusobna integracija u sistem za nadgledanje saobraćaja. Platforma pruža relevantne informacije koje pomažu u identifikaciji zagušenja saobraćaja i omogućavaju efikasno planiranje putovanja. Takođe smo koristili tehnologiju vizualizacije podataka kako bismo prikazali simulaciju platforme u gradu Podgorici.

Najznačajniji doprinos našeg rada je razvoj ove kompleksne platforme koja pruža korisne informacije i alate za efikasno upravljanje saobraćajem i planiranje održive mobilnosti. Naša platforma omogućava donosiocima odluka da identifikuju zagušenja, donose informisane odluke o upravljanju saobraćajem i planiraju efikasne rute za uštedu vremena i resursa. Time doprinosimo poboljšanju efikasnosti i održivosti saobraćaja u gradu.

Naš rad je postigao ciljeve razvoja platforme za monitoring saobraćaja i podršku odlučivanju, odgovorio na istraživačka pitanja i pružio značajan doprinos u poboljšanju efikasnosti i održivosti saobraćaja kroz integraciju relevantnih informacija i alata za donošenje odluka.

Literatura

[1] Saraswathi, A., Mummoorthy A, Anantha Raman G R and K. P. Porkodi. “Real-Time Traffic Monitoring System Using Spark.” 2019 International Conference on Emerging Trends in Science and Engineering (ICESE) 1 (2019): 1-6.

[2] Po, Laura, Federica Rollo, Chiara Bachechi and Alberto Corni. “From Sensors Data to Urban Traffic Flow Analysis.” 2019 IEEE International Smart Cities Conference (ISC2) (2019): 478-485.

[3] Anveshrithaa, S and K. Lavanya. “Real-Time Vehicle Traffic Analysis using Long Short Term Memory Networks in Apache Spark.” 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE) (2020): 1-5.

[4] Kul, Seda, Isabek Tashiev, Ali Sentas and Ahmet Sayar. “Event-Based Microservices With Apache Kafka Streams: A Real-Time Vehicle Detection System Based on Type, Color, and Speed Attributes.” IEEE Access 9 (2021): 83137-83148.

[5] Bosurgi, Gaetano, Dario Bruneo, Fabrizio De Vita, Orazio Pellegrino and Giuseppe Sollazzo. “A web platform for the management of road survey and maintenance information: A preliminary step towards smart road management systems.” Structural Control and Health Monitoring 29 (2021): n. pag.

[6] Anusha, Divya, Sufiya and Mrs. Shilpashri. “IoT Based Traffic Monitoring System with Priority to Emergency Vehicles.” (2019).

[7] Badidi, Elarbi and Muthucumaru Maheswaran. “Towards a Platform for Urban Data Management, Integration and Processing.” International Conference on Internet of Things, Big Data and Security (2018).

[8] Gupta, Tanvi and Madhavi Damle. “Decision Making For A Proposed Traffic Management Solution With The Application Of IOT Technology.” 2022 International Conference on Decision Aid Sciences and Applications (DASA) (2022): 991-998.

[9] Patel, Minal Parimalbhai, Akash Mehta and Narendra C. Chauhan. “Design of Smart Dashboard based on IoT & Fog Computing for Smart Cities.” 2021 5th International Conference on Trends in Electronics and Informatics (ICOEI) (2021): 458-462.

[10] León-Paredes, Gabriel A., Omar Gustavo Bravo-Quezada, Erwin J. Sacoto-Cabrera, Wilson

F. Calle-Siavichay, Ledys L. Jimenez-Gonzalez and Juan Aguirre-Benalcazar. “Virtual Reality Platform for Sustainable Road Education among Users of Urban Mobility in Cuenca, Ecuador.” International Journal of Advanced Computer Science and Applications (2022): n. pag.

[11] Vo, Minh-Tri, Trieu-Vu Tran, Duc-The Pham and Trong-Hop Do. “A Practical Real-Time Flight Delay Prediction System using Big Data Technology.” 2022 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT) (2022): 160-167.

[12] Meng, Xiangrui et al. “MLlib: Machine Learning in Apache Spark.” J. Mach. Learn. Res. 17 (2015): 34:1-34:7.

[13] Sax, M. (2019). Apache Kafka. Encyclopedia of Big Data Technologies.

[14] D'silva, Godson Michael et al. “Real-time processing of IoT events with historic data using Apache Kafka and Apache Spark with dashing framework.” 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT) (2017): 1804-1809.

[15] Corral-Plaza, David et al. “A stream processing architecture for heterogeneous data sources in the Internet of Things.” Comput. Stand. Interfaces 70 (2020): 103426.

[16] Lakshman, A., & Malik, P. (2010). Cassandra: a decentralized structured storage system. ACM SIGOPS Oper. Syst. Rev., 44, 35-40.

[17] Hunt, P., Konar, M., Junqueira, F.P., & Reed, B.C. (2010). ZooKeeper: Wait-free Coordination for Internet-scale Systems. USENIX Annual Technical Conference.

[18] Wilkinson, A., & Frederick, S. (2020). Spring Boot Maven Plugin Documentation.