Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

Grega Vrbančič

# RAZVOJ SPLETNE APLIKACIJE ZA ANALITIKO PODATKOV V REALNEM ČASU S SPRING XD

Diplomsko delo

Maribor, september 2015

# RAZVOJ SPLETNE APLIKACIJE ZA ANALITIKO PODATKOV V REALNEM ČASU S SPRING XD

## Diplomsko delo


| | |
|---|---|
| Študent: | Grega Vrbančič |
| Študijski program: | Univerzitetni študijski program<br>Informatika in tehnologije komuniciranja |
| Smer: | Informacijski sistemi |
| Mentor: | red. prof. dr. Vili Podgorelec, univ. dipl. inž. rač. in inf. |
| Lektorica: | Eva Božič, diplomirana slovenistka in umetnostna zgodovinarka |

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

Številka: E1042540
Datum in kraj: 02. 06. 2015, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 46/2012)
izdajam

**SKLEP O DIPLOMSKEM DELU**

1. **Gregi Vrbančiču,** študentu univerzitetnega študijskega programa INFORMATIKA IN TEHNOLOGIJE KOMUNICIRANJA, smer Informacijski sistemi, se dovoljuje izdelati diplomsko delo.

2. **MENTOR:** **red. prof. dr. Vili Podgorelec**

3. **Naslov diplomskega dela:**
**RAZVOJ SPLETNE APLIKACIJE ZA ANALITIKO PODATKOV V REALNEM ČASU S SPRING XD**

4. **Naslov diplomskega dela v angleškem jeziku:**
**DEVELOPMENT OF A WEB APPLICATION FOR REAL-TIME DATA ANALYTICS WITH SPRING XD**

5. Diplomsko delo je potrebno izdelati skladno z "Navodili za izdelavo diplomskega dela" in ga oddati v treh izvodih (dva trdo vezana izvoda in en v spiralo vezan izvod) ter en izvod elektronske verzije do 30. 09. 2015 v referatu za študentske zadeve.

Pravni pouk: Zoper ta sklep je možna pritožba na senat članice v roku 3 delovnih dni.

Dekan:
red. prof. dr. Borut Žalik


Obvestiti:
- kandidata,
- mentorja,
- odložiti v arhiv.

## Zahvala

*Zahvaljujem se mentorju dr. Viliju Podgorelcu za prijazen odnos in strokovno pomoč pri izdelavi diplomskega dela.*

*Posebno zahvalo namenjam staršema, ki sta mi omogočila študij.*

*Zahvala gre tudi dekletu in vsem, ki so me na kakršenkoli način podpirali pri študiju.*

# Razvoj spletne aplikacije za analitiko podatkov v realnem času s Spring XD

**Ključne besede:** analitika podatkov, spletna aplikacija, Spring XD, masovni podatki

**UDK:** 004.5:004.775(043.2)

**Povzetek**

*Živimo v sodobni informacijski družbi, katera nevedoč, na vsakem koraku z uporabo trenutnih razpoložljivih tehnologij ter spleta, dnevno ustvari več eksabajtov podatkov v različnih oblikah. Želja po pridobivanju dodane vrednosti – znanja, iz na prvi pogled nepomembnih podatkov, je v gospodarstvu prisotna že dalj časa, kot so tudi že dalj časa prisotne tehnike, metode za analitiko podatkov. Te so se razvile do te mere, da nam lahko v sprejemljivem času ponudijo rezultate naših povpraševanj. Kljub temu pa ostaja želja po instantnem pridobivanju rezultatov, analitiki podatkov praktično v trenutku, ko se ti ustvarijo. V diplomskem delu smo se posvetili uporabi platforme Spring XD ter v povezavi z njo razvili spletno aplikacijo za analitiko podatkov v realnem času.*

# Development of a web application for real-time data analytics with Spring XD

**Key words:** data analytics, web application, Spring XD, big data

**UDK:** 004.5:004.775(043.2)

**Abstract**

*We live in a modern information society, which unknowingly at any time using current available technologies and web generate exabytes of data in various forms daily. The desire to aquire added value – knowledge, of at first sight insignificant data is present in the economy for a long time, as well as are for a long time present techniques and methods for data analytics. These have been developed to such an extent that they can provide us with the results of our queries within a reasonable time. Nevertheless, it remains a desire for obtaining instant results, desire for data analytics practically at the time of being generated. In this thesis we focused on the use of Spring XD platform and in conjunction with it we developed a web application for real-time data analytics.*

# KAZALO

# KAZALO SLIK



# KAZALO TABEL



# KAZALO PROGRAMSKIH KOD

# SEZNAM UPORABLJENIH KRATIC

DSL – Domain-specific language
HTTP – Hypetext transfer protocol
REST – Representational state transfer
TCP – Transmission control protocol
PMML – Predictive model markup language
POJO – Plain old java object
JSON – JavaScript object notation
XML – Extensible markup language
API – Application programming interface
HDFS – Hadoop Distributed File System
ANSI – American National Standards Institute
JMS – Java messaging service
MQTT – Message Queuing TelemetryTransport
JDBC – Java database connectivity
SpEL – Spring Expression Language

# 1 UVOD

Živimo v sodobni informacijski družbi, katera ne vedoč na vsakem koraku z uporabo trenutno razpoložljivih tehnologij ter spleta, dnevno ustvari več eksabajtov podatkov v različnih oblikah. Želja po pridobivanju dodane vrednosti znanja, iz na prvi pogled nepomembnih podatkov, je v gospodarstvu prisotna že dalj časa, kot so tudi že dalj časa prisotne tehnike, metode za analitiko podatkov. Te so se razvile do te mere, da nam lahko v sprejemljivem času ponudijo rezultate naših povpraševanj. Kljub temu pa ostaja želja po instantnem pridobivanju rezultatov – analitiki podatkov praktično v trenutku, ko se ustvarijo. Iz vidika določenih gospodarskih panog je posedovanje znanja, pridobljenega iz novo ustvarjenih podatkov, v trenutku, ko so ti ustvarjeni, lahko ključnega pomena in lahko pripomore k boljšim strateškim odločitvam podjetja ter posledično doprinese k boljšim poslovnim rezultatom. V diplomskem delu smo tako predelali teorijo o analitiki podatkov v realnem času, preučili delovanje in funkcionalnosti platforme za analitiko podatkov v realnem času *Spring XD* ter v povezavi s slednjo razvili aplikacijo za analitiko podatkov v realnem času.

Cilj diplomskega dela je bil v prvem delu seznaniti se s teorijo o analitiki podatkov v realnem času, raziskati in analizirati platformo *Spring XD*, pregledati njeno arhitekturo ter možnosti uporabe, razširitve in integracije platforme.

V drugem delu smo se osredotočili na praktični primer uporabe platforme. S pomočjo platforme *Spring XD* smo razvili spletno aplikacijo za analitiko podatkov v realnem času, katera podatke črpa iz socialnega omrežja *Twitter*, rezultate pa razumljivo in grafično predstavi uporabniku.

# 2 ANALITIKA PODATKOV V REALNEM ČASU

Velika količina podatkov ustvarjenih v današnjih dneh ustvarja nove pomembne poslovne priložnosti za organizacije, ki posedujejo masovne podatke, da iz njih izpeljejo in ustvarijo pomembno konkurenčno prednost. V podjetjih masovni podatki pomagajo spodbujati učinkovitost, kvaliteto in ustvarjanje personaliziranih produktov ter storitev, s katerimi dosegajo visoko stopnjo zadovoljstva strank. Medtem pa v akademskih sferah analitika masovnih podatkov ponuja nove možnosti raziskovanja s potencialno bogatejšimi rezultati in globljim vpogledom kot kadarkoli prej. V veliko primerih analitika masovnih podatkov združuje strukturirane in nestrukturirane podatke z viri podatkov v realnem času in poizvedbami nad njimi ter tako odpira nove poti za inovacije in uporabo. [1]

Pomen besedne zveze *»realni čas«* je različen glede na kontekst v katerem je uporabljena. Če govorimo o analitiki podatkov v realnem času in podobnih sistemih, imamo v mislih arhitekturo, ki nam omogoča reagirati na podatek v trenutku, ko smo ga prejeli, brez da bi ga morali pred tem hraniti v podatkovno bazo. Torej nam sistemi za analitiko podatkov omogočajo obdelavo podatkov v trenutku, ko podatki prispejo, za razliko od klasičnih sistemov, ki delujejo po principu hranjenja podatkov v bazo in obdelovanja teh v prihodnosti.

Joseph M. Hellerstein, profesor na univerzi Berkeley, je povedal: *»V kolikor imate v procesu analitike vključene ljudi, potem to ni v realnem času. Večina ljudi potrebuje sekundo ali dve, da reagira in to je ogromno časa že za tradicionalne transakcijske sisteme da sprocesirajo vhod in izhod.«* [2]

Zmožnost hitrega hranjena podatkov ni več nobena novost. Novost je zmožnost hitro narediti nekaj koristnega s podatki z majhno porabo virov. V zadnjem času smo priča porastu novih tehnologij za analitiko ogromnih tokov podatkov, kot tudi novih tehnologij ustvarjenih za ravnanje s kompleksnimi strukturami teh podatkov. V preteklosti so morali podatki biti urejeni v tabelah, v današnjem svetu analitike podatkov pa je oblika nepomembna, česar so se že tudi navadili moderni podatkovni znanstveniki, saj jim prebijanje skozi gruče neurejenih podatkov iz različnih virov v različnih oblikah ni več tuje. [3]

Michael Minelli, soavtor knjige *Big Data, Big Analytics pravi: »Masovni podatki v realnem času niso samo proces hranjenja petabatov ali eksabajtov podatkov v podatkovna skladišča. Gre se za zmožnost boljšega odločanja in sprejemanja smiselnih ukrepov v pravem trenutku. Gre za zaznavanje prevare v trenutku, ko nekdo povleče kreditno kartico ali pa za proženje ponudbe kupcu, med tem ko stoji v vrsti za nakup. Ali za prikaz reklame na spletno stran v trenutku medtem ko nekdo bere nek članek. Združuje in analizira podatke zato, da lahko sprejmemo pravilen ukrep, ob pravem času, na prvem mestu.«* [2]

## 2.1 Faze analitike podatkov v realnem času

Analitika podatkov v realnem času je iterativni proces, ki vključuje različna orodja in sisteme. Po besedah Davida Smitha, strokovnjaka na področju masovnih podatkov in enega izmed desetih najvplivnejših ljudi *Forbesove* lestvice, je na temo masovnih podatkov priporočljivo ločiti proces analitike podatkov v realnem času na pet faz: destilacija podatkov, razvoj modela, potrditev in namestitev, vrednotenje v realnem času in osvežitev modela. Petfazni model je oblikovan kot ogrodje za napovedno analitiko, prav tako pa deluje odlično kot splošno ogrodje za analitiko podatkov v realnem času. [2]

### 2.1.1 Destilacija podatkov

Podatki na podatkovnem nivoju so neobdelani in neurejeni, s pomanjkanjem strukture potrebne za izdelavo modela in izvajanje analitike. Faza vključuje pridobivanje značilnosti nestrukturiranega besedila, združevanje neskladnih podatkovnih virov, filtriranje po ciljni domeni, izbiro pomembnih značilnosti in rezultatov za modeliranje ter izvoz sklopov destiliranih podatkov.

### 2.1.2 Razvoj modela

Proces v tej fazi vključuje izbiro značilnosti, vzorčenje, preoblikovanje spremenljivk, ocenjevanje modela, izpopolnitev modela in merjenje uspešnosti modela. Cilj te faze je ustvariti napovedni model, ki je robusten, močan, celovit in izvedljiv. Ključne zahteve v tej fazi so hitrost, fleksibilnost, produktivnost in ponovljivost, katere so pomembne v kontekstu masovnih podatkov. [2]

### 2.1.3 Potrditev in namestitev

Cilj te faze je testiranje modela z namenom, da se prepričamo, da deluje v realnem okolju. Faza vključuje ponovno pridobivanje značilnosti iz novih podatkov, s katerimi testiramo model in primerjamo rezultate z obstoječimi. Če model deluje, je pripravljen za namestitev v produkcijsko okolje. [2]

### 2.1.4 Vrednotenje v realnem času

V realno-časovnih sistemih je vrednotenje sproženo s strani definirane akcije. Preko podatkovnega nivoja dobimo nove podatke, kateri so nato ovrednoteni s pomočjo modela, ustvarjenega v prejšnjih fazah.

### 2.1.5 Osvežitev modela

Podatki se vedno spreminjajo, zato mora obstajati možnost osvežitve podatkov in modela zgrajenega nad njimi. Obstoječe skripte in programi uporabljeni za obdelavo podatkov in izgradnjo modela so lahko uporabljeni v procesu osvežitve modela. Priporočljiva je tudi raziskovalna analiza podatkov vključno s periodičnimi osvežitvami modela. Proces osvežitve kot tudi proces potrditve in namestitve modela lahko avtomatiziramo s pomočjo različnih spletno orientiranih storitev. [2]

## 2.2 Analitika besedil

Analitika besedil (angl. text analytics) se nanaša na obdelavo in procesiranje tekstovnih podatkov z namenom pridobitve uporabnega globljega vpogleda. Ključna komponenta analitike besedil je besedilno rudarjenje (angl. text mining). To je proces odkrivanja povezav in zanimivih vzorcev v velikih zbirkah podatkov.

Problem analitike besedil sestoji iz treh korakov: sintaktična analiza ali razčlenjevanje (angl. parsing), iskanje in pridobivanje (angl. search and retrieval) ter besedilno rudarjenje. Sintaktična analiza je proces, kateri na vhodu dobi nestrukturirano besedilo in ga razčleni v strukturirano obliko za nadaljnje potrebe analitike. Pod nestrukturirana besedila

lahko štejemo navadne tekstovne datoteke, dnevniške zapise, XML datoteke, HTML datoteke, ipd. Korak iskanja in pridobivanja je proces identifikacije dokumentov v korpusu, ki vsebujejo iskane elemente kot so besede, fraze, teme… Ti iskalni elementi so navadno poimenovani kot ključni elementi (angl. key terms). Besedilno rudarjenje uporablja izraze in indekse, ki so bili producirani v prejšnjih dveh korakih z namenom odkritja smiselnih spoznanj na določenem področju. [1]

### 2.2.1 Analiza čustev iz besedila

Analiziranje čustev iz besedila (angl. sentiment analysis) je proces zaznavanja polaritete besedila, ali z drugimi besedami razpoznavanje, ali je besedilo naravnano pozitivno, negativno ali nevtralno.

Pristope za klasifikacijo čustev na podlagi besedila lahko razdelimo v skupine: klasifikacija s strojnim učenjem, klasifikacija bazirana na leksikonih ter hibridna klasifikacija. Pristop za klasifikacijo s strojnim učenjem uporablja znane algoritme za strojno učenje ter izrablja jezikovne značilnosti. Klasifikacija, bazirana na leksikonih, se zanaša na zbirke znanih, v naprej ocenjenih čustvenih izrazov. Slednja je razdeljena na klasifikacijo, bazirano na slovarjih, ter klasifikacijo, bazirano na korpusih. Slednji uporablja statistične ali semantične metode za vrednotenje polaritete besedila. Hibridna klasifikacija združuje oba od prej naštetih. Zelo pogosto imajo leksikoni glavno vlogo v večini algoritmov za razpoznavanje polaritete besedil. [3]

Natančnost analize čustev iz besedila lahko merimo na različne načine. Najbolj pogost način merjenja je s primerjavo natančnosti med računalniškim algoritmom ter človekom. Študija univerze iz *Pittsburgha* govori, da se različni ljudje o polariteti enakega besedila strinjajo v 80 %. [5] Iz tega lahko sklepamo, da je odličen računalniški algoritem vsak, ki ima natančnost ovrednotenja polaritete okrog 80 %. Pri računalniškem analiziranju besedila se srečamo s kar nekaj izzivi. Eden največjih je zaznavanje ironije in sarkazma v besedilu, s čemer ima težavo tudi kar velik delež ljudi. Slednje je tudi eden najbolj pogostih vzrokov za napačno ovrednotenje polaritete besedil s strani algoritmov. Prav tako je eden večjih problemov pri analiziranju besedila večpomenskost besed. [4] Na primer, če nekdo govori o formuli, se nam lahko v primeru, da ne poznamo konteksta pogovora, poraja vprašanje ali govori o matematični formuli ali o dirkalniku formule 1.

## 2.3 Primeri uporabe

Uporaba sistemov za analitiko podatkov v realnem času je vsesplošno uporabna. Z rastjo količine generiranih podatkov s strani socialnih omrežij ter podatkov generiranih s strani pametnih mobilnih telefonov je analitika podatkov v realnem času vedno bolj ključnega pomena za usmerjeno oglaševanje (angl. targeted advertising). Tudi lokacijsko bazirano usmerjeno oglaševanje bi pridobilo ogromno z uporabo takšnega pristopa analitike podatkov, če si predstavljamo, da bi se nam na primer na telefonu, namesto naključno generiranih oglasov, prikazovali oglasi izbrani s strani algoritma, upoštevajoč našo trenutno lokacijo, vreme, trenuten čas, dan v tednu ipd. Možna uporabnost se pojavlja tudi na področju finančnega trgovanja. Družbe vlagajo ogromno sredstev v realno časovne algoritme, kateri imajo zmožnost trgovanja z ekstremno visoko hitrostjo. Takšni sistemi so sposobni izvršiti tisoče poslov v sekundi, upoštevajoč trenutno glavne svetovne novice, nekateri pa celo tudi objave iz socialnega omrežja Twitter, katere ovrednotijo ter jim določijo polariteto. Eno izmed področij, kjer bi bila uporabna analitika podatkov v realnem času, je tudi področje plačilnega prometa. V realnem času bi lahko analizirali transakcije in označevali sumljive za podrobnejši pregled s strani pooblaščene osebe. [5]

Če povzamemo, je analitika podatkov v realnem času predvsem primerna v scenarijih, kjer je nujna hipna interakcija s sistemom ali uporabnikom v trenutku nekega dogodka. V takšnih primerih uporabe je smiselno podatke hraniti zgolj v delovnem pomnilniku, saj s tem dodatno pohitrimo proces. Trajno hranjenje podatkov pa uporabimo za nadaljnjo strojno učenje in globljo analitiko podatkov.

# 3 PLATFORMA SPRING XD

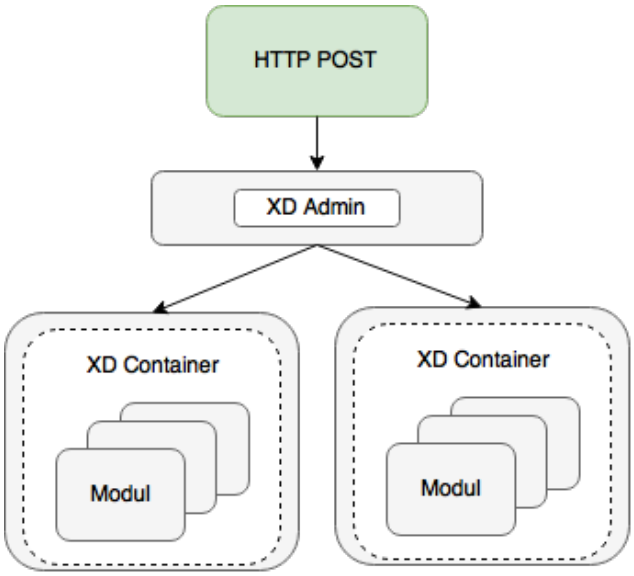
*Spring XD* (okrajšava za *Spring Extreme Data*) je odprtokodna razširljiva platforma, ki zagotavlja skalabilnost, toleranco do napak in delovanje v porazdeljenem okolju. Poenostavlja reševanje pogostih problemov pri delu z masovnimi podatki kot so obdelava in izvoz podatkov, izvajanje analitik v realnem času in orkestracijo paketnih delovnih tokov. Plaforma je prosto dostopna (pod licenco *Apache Licence 2.0*), zgrajena na zrelih, priznanih, odprtokodnih projektih *Spring* (*Spring Integration*, *Spring Data*, *Spring Batch*...) in napisana v programskem jeziku *Java*.

## 3.1 Arhitektura

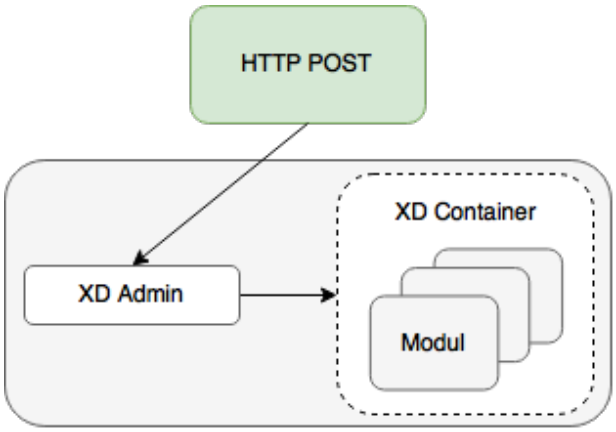
Plaforma lahko deluje v samostojnem načinu na enem računalniku ali v porazdeljenem integriranem načinu preko več računalniških vozlišč. V prvem primeru platforma teče kot en samostojen proces, ki je odgovoren za izvajanje vseh nalog, tako za procesiranje kot tudi za administracijo. Tak način je predvsem primeren za spoznavanje s platformo, hitro prototipiranje, razvoj ter testiranje. Porazdeljeno integrirano izvajanje (angl. distributed integrated runtime) pa je modularno razdeljeno na več računalniških vozlišč ter omogoča obsežnejše nastavitve ter prilagoditve. [7]

### 3.1.1 Arhitektura izvajalnega okolja

Glavni komponenti izvajalnega okolja platforme *Spring XD* sta *XD Admin* in *XD Container*. Z uporabo visoko nivojskega domensko specifičnega jezika (angl. domain specific language) pošljemo opis zahtevane naloge v izvajanje preko *HTTP* protokola, kjer jo *XD Admin* strežnik nato preslika v izvajalne module. Porazdeljeno okolje omogoča, da se izvajalni moduli izvajajo preko več *XD Container* strežnikov, vsak tak strežnik pa lahko izvaja sočasno več modulov (glej Slika 3.1). Pri uporabi platforme v samostojnem (angl. single node – Slika 3.2) načinu se vsi moduli izvajajo na enem *XD Container* strežniku, prav tako pa tudi *XD Admin* strežnik teče v enakem procesu. [7]




Slika 3.1: Delovanje v porazdeljenem integriranem izvajanju



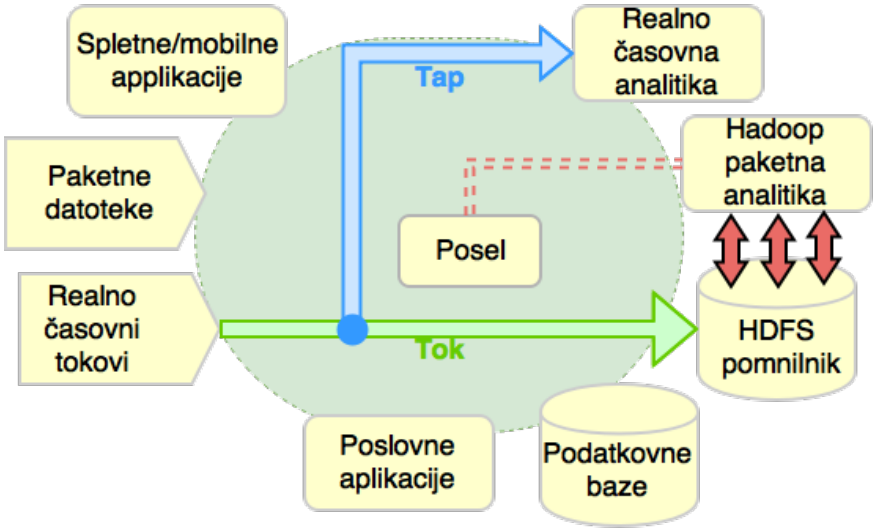

Slika 3.2: Delovanje v samostojnem izvajanju

Kadar je platforma pognana v porazdeljenem integriranem izvajanju, za dodelitev izvajanja modula posameznemu *XD Container* strežniku skrbi *ZooKeeper*. Moduli si v takšnem okolju delijo podatke s pošiljanjem sporočil, pri čemer uporabljajo sporočilno vmesno opremo (*Rabbit*, *Redis*).

*XD Admin* strežnik je namenjen administraciji ter prejemanju ukazov. Uporablja vgrajen vsebnik servletov, preko katerega izpostavlja *REST* vmesnik za ustvarjanje, zagon, zaustavitev ter uničenje tokov in računalniških poslov (angl. job) ter podobno. Razvit je z uporabo odprtokodnega ogrodja *Spring MVC* in knjižnico *Spring HATEOAS*. [7]

### 3.1.2 Arhitektura XD Container strežnika

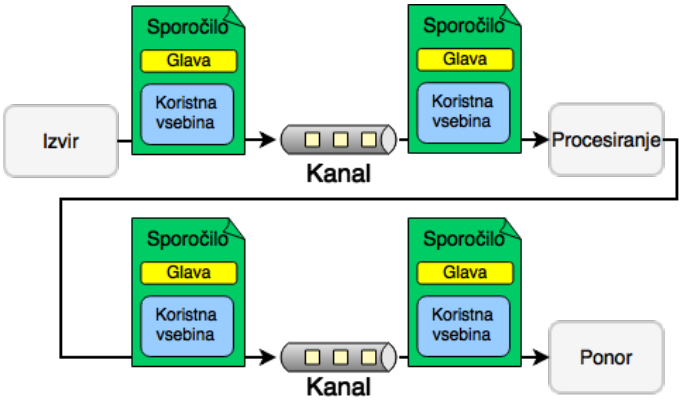
Komponente, ključne pri obdelovanju podatkov v platformi *Spring XD,* so tokovi, računalniški posli ter tap-i. Na sliki 3.3 je prikazano delovanje obdelave podatkov preko platforme *Spring XD*.




Slika 3.3: Prikaz delovanja obdelave podatkov

Tokovi, katerih programski model bazira na podjetniških integracijskih vzorcih EIP (angl. Enterprise integration Patterns), definirajo, kako se dogodkovni podatki zbirajo, obdelujejo, shranjujejo in posredujejo oziroma shranijo. Vsak tok je sestavljen iz modula za vhoden vir, modula za obdelovanje, ki je opcijski, ter modula za ponor. Modul za vhoden vir ustvarja sporočila iz zunanjega vira (TCP, HTTP,..), katera vsebujejo koristno vsebino ter zbirko glav sporočila. Tok sporočil nato potuje od izvora skozi kanal (opcijsko do modulov za obdelavo podatkov ter poslednje skozi kanal do končnega ponora toka (glej Slika 3.4).

Tap-i podvajajo tokove takoj za modulom za vhodni vir in nam s tem omogočajo vzporedno obdelavo enakih podatkov na različne načine, brez poseganja v izvoren tok.




Slika 3.4: Tok sporočil

Tok lahko ustvarimo z naslednjim zaporedjem ukazov (glej Programska koda 3.1), ki jih vpišemo v Spring XD ukazno lupino.

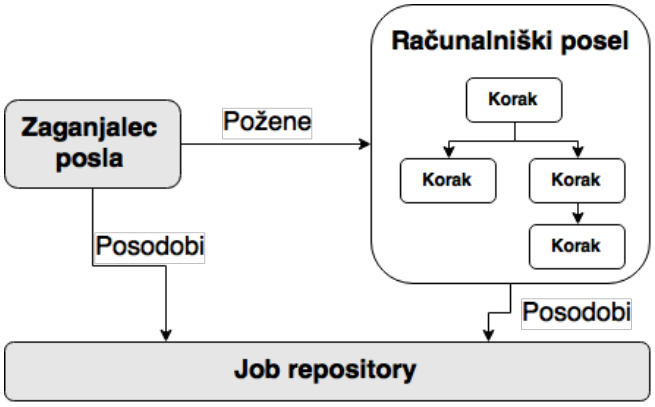
```
stream create --name toktvitov --definition "twittersearch
    --consumerKey=ključ --consumerSecret=skrivnost
    --query=SpringXD | log" --deploy
```

Programska koda 3.1: Ustvarjanje toka

Za besedno zvezo *stream create*, ki je ukaz za ustvarjanje novega toka, sledi parameter *name,* s katerim določimo toku unikatno ime, za njim sledi parameter *definition,* kateri služi za definiranje toka, ki ga je potrebno napisati v skladu s *Spring DSL* sintakso. V našem primeru smo najprej uporabili klic modula *twittersearch*, ki služi kot vir podatkov, nato pa podali vrednosti za obvezna parametra *consumerKey* ter *consumerSecret* in na koncu podali parametru *query* željen iskalni niz, po katerem želimo pridobivati tvite. Zatem smo dodali cevovod (simbol |) in ponor toka speljali v dnevniško datoteko z ukazom *log,* s čimer smo zaključili definicijo toka. Kot zadnji ukaz smo dodali še zastavico *deploy*, ki je sicer opcijska in služi za takojšen zagon toka.

Računalniški posli v *Spring XD* bazirajo na ogrodju *Spring Batch* z namenom poenostavitve ustvarjanja delovnih tokov računalniških poslov (glej Slika 3.5). V *Spring XD* je računalniški posel v bistvu usmerjen graf, katerega vsako vozlišče je en korak pri izvajanju. Koraki so lahko izvedeni zaporedno ali vzporedno, odvisno od nastavitev posameznega računalniškega posla. Te lahko poženemo, ustavimo in ponovno zaženemo pri čemer je

slednje mogoče, ker je napredek izvedenih korakov računalniškega posla hranjen preko vmesnika *Job repository* v podatkovni bazi. [7]




Slika 3.5: Princip delovanja delovnih tokov računalniškega posla

Tap omogoča ”prestrezanje” podatkov, dokler se ti obdelujejo v izvornem toku, ter obdelovanje prestreženih podatkov v ločenem toku. Izvorni tok podatkov tako ostane nedotaknjen s strani »tap-a« in se ga ta tudi ne zaveda.
Tap ustvarimo z naslednjim ukazom (glej Programska koda 3.2), ki ga vpišemo v *Spring XD* ukazno lupino.

```
stream create --name toktvitovtap
   --definition "tap:stream:toktvitov > counter --name=stevectvitov"
   --deploy
```

Programska koda 3.2: Ustvarjanje tap-a

## 3.2 Moduli

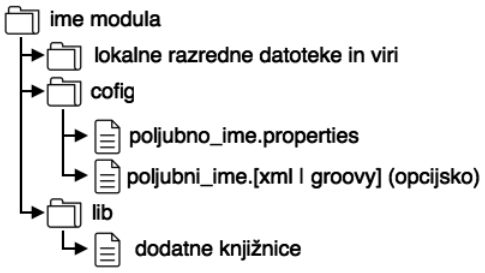
Tokovi so sestavljeni iz modulov, kateri enkapsulirajo enoto dela v ponovno uporabljivo komponento. Module razvrščamo v skupine glede na vlogo oziroma funkcijo, ki jo modul opravlja. Trenutne skupine modulov *Spring XD* so *»viri«*, *»ponorji«*, *»procesorji«* ter *»računalniški posli«*. Skupina ali tip v katero modul spada, določa v kakšen zaporedju so moduli lahko uporabljeni v celotnem toku oziroma uporabljeni v paketnem delovnem toku. Tako morajo biti moduli iz skupine virov prvi moduli v vsakem toku. Moduli iz skupine procesorjev opravljajo definirano nalogo, pri čemer uporabijo sporočilo kot vhodni parameter in po opravljeni nalogi producirajo novo izhodno sporočilo. Ponorji prejmejo

vhodna sporočila in podatke izvozijo v določen zunanji vir ter s tem končajo tok. Skupina računalniških poslov je posebna vrsta modulov, le-ti implementirajo *Spring Batch* računalniške posle, katerih delovanje je omogočeno na *Spring XD*. Platforma kot taka vsebuje vnaprej pripravljene module, katere lahko povežemo v tokove za potrebe razvoja kompleksnih aplikacij, ki temeljijo na masovnih podatkih.

### 3.2.1 Razvoj modula

Moduli, ki spadajo v skupino virov, ponorjev ali procesorjev, so razviti z uporabo ogrodja *Spring Integration*. Tipično takšni moduli izvajajo samo eno nalogo in jih zlahka ponovno uporabimo v različnih tokovih. Pri razvijanju modulov se je priporočeno držati uporabljenih konvencij glede poimenovanja vhodnih in izhodnih kanalov, kateri so v skladu z načelom *KISS* vedno poimenovani *»input«* in *»output«*. [7]

Modul je zapakirana komponenta, vsebujoča artefakte, kateri so uporabljeni za ustvarjanje *Spring* aplikacijskega konteksta. Modul se kot tak ne zaveda svojega izvajalnega okolja. Vsak aplikacijski kontekst posameznega modula je konfiguriran in povezan z ostalimi moduli preko vtičnikov zaradi namena ohranjanja podpore za porazdeljeno delovanje. Fizično gledano je modul podoben *war* datoteki v vsebniku servletov. *XD Container* konfigurira in zažene modul v trenutku, ko je ta nameščen. Struktura zapakiranega modula se je skozi čas opazno razvijala predvsem zaradi dodajanja novih funkcionalnosti za podporo razvoju modulov po meri. Ta evolucija pa je doprinesla povečano fleksibilnost posameznih artefaktov. Tako je sedaj struktura zapakiranega modula definirana, kot je prikazano na sliki 3.6:




Slika 3.6: Struktura modula

Moduli tipično vsebujejo konfiguracijsko datoteko aplikacijskega konteksta v *XML* ali *Groovy* formatu. Obstaja lahko izključno samo ena datoteka. V primeru uporabe konfiguracijskega razreda z anotacijo @*Configuration*, nobena konfiguracijska datoteka ne sme biti prisotna. V datoteki o lastnostih modula s končnico *properties* lahko zapišemo pot do opcijskega razreda, v katerem so definirane vse omogočene funkcionalnosti modula. Kot alternativo opcijskemu razredu lahko omogočene funkcionalnosti navedemo v datoteki o lastnostih modula. V primeru, da uporabimo konfiguracijski razred, mora datoteka vsebovati *base_packages* atribut, kjer je navedena pot do osnovnega paketka izvorne programske kode. [7]

Za uporabo razvitega modula je potrebno modul registrirati v *Spring XD* registru modulov. Ko je modul pravilno zapakiran s pravilno strukturo, ga z ukazom *module upload* registriramo v platformo.

```
module upload --file pot_do_modula/mojmodul-1.0.0.BUILD-SNAPSHOT.jar
```

Programska koda 3.3: Ukaz za namestitev modula

### 3.2.2 Osnovni moduli

*Spring XD* ima v osnovi vnaprej naložene osnovne module za takojšen začetek uporabe platform. Ti so v spodnji tabeli opisani ter kategorizirani glede na skupino modulov, v katero spadajo. [6]

Tabela 3.1: Seznam osnovnih modulov

| Ime modula | Opis modula | Tip modula |
|---|---|---|
| **HTTP** | Bere vhodne podatke iz HTTP zahtevka | Izvor |
| **TCP** | Bere ali piše podatke iz/v TCP vtičnice | Izvor ali ponor |
| **Mail** | Bere ali pošilja pošto preko IMAP strežnika | Izvor ali ponor |
| **JMS** | Prejema sporočila JMS | Izvor |
| **RabbitMQ** | Prejema ali pošilja sporočila na RabbitMQ strežnik | Izvor ali ponor |
| **Twitter stream** | Uporablja Twitter streaming API za branje tvitov | Izvor |
| **Twitter search** | Uporablja Twitter search API za branje tvitov glede na iskalno geslo | Izvor |
| **File** | Bere ali zapisuje v datoteko | Izvor ali ponor |
| **Tail** | Bere tok iz dane datoteke | Izvor |
| **MQTT** | Prejema ali pošilja sporočila telemetrije | Izvor ali ponor |

| | | |
|---|---|---|
| **Time** | Na podan interval odda časovni žig | Izvor |
| **Gemfire** | Posluša morebitne akcije ali poizvedbe na Gemfire strežniku | Izvor |
| **Log** | Zapisuje v dnevnik | Ponor |
| **JDBC** | Shrani izhodne podatke v relacijsko podatkovno bazo | Ponor |
| **HDFS** | Shrani izhodne podatke HDFS | Ponor |
| **Splunk server** | Spremeni izhodne podatke v Splunk dogodek in ga pošlje preko TCP na Splunk strežnik | Ponor |
| **Gemfire server** | Izhodne podatke zapiše v Gemfire Cache strežnik | Ponor |
| **Filters** | Izvaja filtriranje toka s SpEL izrazi ali Groovy skriptami | Procesor |
| **JSON Field Value** | V primeru ujemajočih se vrednosti JSON polj posreduje sporočilo naprej | Procesor |
| **JSON Field Extractor** | Razširi vrednost JSON polja in odda vrednost na izhod | Procesor |
| **Transform** | Preoblikuje vhodno sporočilo in ga pošlje na izhod | Procesor |
| **Split** | Prejme sporočilo in ga razdeli na več delov glede na podan izraz | Procesor |
| **Aggregator** | Združi vsebino sporočil in ustvari novo agregirano sporočilo | Procesor |

## 3.3 Analitika

Platforma omogoča izvajanje različnih ocenjevalnih algoritmov za strojno učenje, kot tudi podpira enostavnejše analitike z uporabo različnih tipov števcev. Funkcionalnost je omogočena preko modulov, kateri so lahko dodani v tok. Mogoče je tok podatkov primarno uporabiti za izvajanje analitik, vendar je bolj pogosta uporaba tap-ov v ta namen.

### 3.3.1 Napovedna analitika

Za namene izvajanja napovedne analitike nam platforma nudi *PMML* modul, ki je integriran z *JPMML-Evaluator* knjižnico. Le-ta nam omogoča podporo za širok nabor različnih tipov modelov ter je interoperabilna z modeli izvoženimi iz *R*, *Rattle*, *KNIME* ter *RapidMiner*. *Spring XD* platforma tako podpira ključne abstrakcije za izvajanje analitičnih modelov v procesu obdelave toka. [7]

*PMML* modul deluje na način, s katerim preko vhodnega parametra prejme pot do analitičnega modela. Analitični modeli so ponavadi definirani s strani strokovnjaka za podatkovno rudarjenje ali generirani s pomočjo orodja za analitiko podatkov. Trenutno modul podpira samo modele definirane s *PMML*. *PMML* je standardizirana datoteka, bazirana na XML-ju, ki omogoča opis in izmenjavo analitičnih modelov. Tak model je nato naložen v modul, kateri v toku nad prejetimi sporočili izvaja ocenjevanje. [7]

### 3.3.2 Števci in merilniki

Števci in merilniki so analitične podatkovne strukture, pogosto imenovane kar metrike. *Spring XD* omogoča uporabo števcev in merilnikov na mestu ponora toka, uporabimo pa jih lahko tudi s pomočjo tap-ov.

Platforma nam omogoča izbiro med petimi različnimi metrikami:

- števec,
- skupni števec,
- števec vrednosti polja,
- merilnik,
- bogat merilnik.

Primarni namen števca je štetje dogodkov, sproženih s strani vhodnega sporočila. Vsak števec mora imeti unikatno ime, opcijsko lahko ima tudi začetno vrednost. Najbolj tipična uporaba števca bi bila štetje sporočil ciljnega toka podatkov. V tem primeru se, pri vsakem vhodnem sporočilu, števec poveča za ena (1). Takšen enostaven števec ustvarimo po predhodno ustvarjenem toku (glej Programska koda 3.1) z ukazom v Spring XD ukazno lupino (glej Programska koda 3.2).

Skupni števec se razlikuje od števca po tem, da ne hrani samo zadnjega stanja števca, ampak hrani tudi vrednost števca za vsako minuto, uro, dan in mesec za celoten interval njegovega delovanja. Podatke pridobimo na način, da podamo začeten in končen datum ter interval v kakšnem želimo rezultate. Če bi želeli ustvariti skupni števec za že ustvarjen tok (glej Programska koda 3.1), lahko to storimo z naslednjim ukazom:

```
stream create --name toktvitovtap2
   --definition "tap:stream:toktvitov > counter
```

Programska koda 3.4: Ustvarjanje skupnega števca

Števec vrednosti polja je metrika, ki se uporablja za štetje pojavitev unikatnih vrednosti za določeno polje v koristni vsebini sporočila. *Spring XD* v osnovi podpira tri tipe koristnih vsebin sporočil in sicer *POJO*, *Tuple* ter *JSON String*. V spodnjem primeru (glej Programska koda 3.5) tako ustvarimo števec vrednosti polja, s katerim štejemo pojavitve posameznih oznak (angl. hashtag) v tvitih.

```
stream create --name toktvitovtap3
   --definition "tap:stream:toktvitov > aggregate-counter
```

Programska koda 3.5: Ustvarjanje števca vrednosti polja

Merilnik je podobna metrika kot števec, saj podobno kot slednji, hrani le eno vrednost povezano z unikatnim imenom. V primeru merilnika vrednost lahko predstavlja katerokoli numerično vrednost definirano s strani aplikacije. V primeru, da merilnik uporabimo kot ponor toka, je pričakovana numerična vrednost vsebovana v koristni vsebini sporočila. Ponavadi bi to bil v niz oblikovan decimalni zapis. Primer preproste uporabe merilnika (glej Programska koda 3.6):

```
stream create --name tok --definition "http --port=9090 | log" --deploy
stream create --name merilnik --definition "tap:stream:tok > gauge"
```

Programska koda 3.6: Uporaba merilnika

Bogat merilnik (angl. Rich gauge) ima enake lastnosti kot merilnik, poleg tega pa hrani tudi trenutno povprečje, minimalno ter maksimalno vrednost. Primer preproste uporabe bogatega merilnika:

```
stream create --name tok --definition "http --port=9090 | log" --deploy
stream create --name bogatmerilnik --definition "tap:stream:tok
   > rich-gauge" --deploy
```

Programska koda 3.7: Uporaba bogatega merilnika

## 3.4 Upravljanje

Platforma nam ponuja upravljanje preko namenskega spletnega grafičnega vmesnika poimenovanega *Admin UI* ter preko posebej razvite ukazne lupine imenovane *Spring XD Shell*. Za programsko upravljanje pa imamo na voljo za uporabo REST spletni aplikacijski vmesnik.

### 3.4.1 Admin UI

*Admin UI* je spletni grafični vmesnik za upravljanje s platformo *Spring XD*. Kot privzeto je dostopen na *URL* naslovu *http://<naslov-streznika>:9393/admin-ui.* Trenutno je razdeljen na štiri sklope:

- Sklop poimenovan *Containers*: omogoča nam pregled nad trenutno instanciranimi *Spring XD* vsebniki ter njihovimi lastnostmi. Omogočena je tudi zaustavitev posameznih vsebnikov.
- Sklop poimenovan *Streams*: vsebuje pregled obstoječih tokov in njihovih definicij. Možen je tudi zagon, zaustavitev ter uničenje posameznega toka.
- Sklop poimenovan *Jobs*: vsebuje štiri dodatne zavihke preko katerih lahko nadzorujemo in upravljamo module ustvarjenih računalniških poslov, pregledujemo njihove definicije ter upravljamo z njihovim delovanjem.
- Sklop poimenovan *Analytics*: omogoča pregled trenutno delujočih metrik, ki delujejo nad tokovi.

### 3.4.2 Spring XD Shell

Platforma *Spring XD* vključuje interaktivno ukazno lupino, katero lahko uporabimo za ustvarjanje, namestitev, uničenje in poizvedovanje tokov in računalniških poslov. Prav tako vključuje tudi ukaze za pomoč pri pogostih operacijah, med drugim tudi za upravljanje z *HDFS*, *UNIX*[1] ukazno lupino in pošiljanje *HTTP* zahtev. *Spring XD Shell* ukazna lupina je razvita iz odprtokodnega projekta *Spring Shell*, ki služi kot enostavno razširljivo ogrodje z ukazi, baziranimi na *Spring* programskem modelu, kar poenostavi razvoj interaktivnih ukaznih lupin s funkcionalnostmi kot so *ANSI* obarvanje, dopolnjevanje ukazov, brskanje po zgodovini uporabljenih ukazov itd.

[1] UNIX je večopravilni večuporabniški operacijski sistem, razvit leta 1969 v Bellovih laboratorijih.

Ukazi so v grobem razdeljeni v skupine glede na področje upravljanja. Tako imamo skupino ukazov za upravljanje modulov, skupino za upravljanje tokov, skupino za upravljanje računalniških poslov, skupino za analitiko  ter skupino za upravljanje *HDFS*.

V veliko pomoč pri vnašanju ukazov nam je funkcionalnost dopolnjevanja ukazov. Ta nam ob pritisku na tipko *TAB,* poleg dopolnjevanja delno vpisanih ukazov, ponudi tudi seznam mogočih nadaljnjih ukazov ter opcijskih parametrov glede na trenutno vpisan ukaz.

### 3.4.3 Aplikacijski programski vmesnik

V kolikor želimo platformo *Spring XD* povezati z lastno aplikacijo, lahko to storimo preko *REST* aplikacijskega programskega vmesnika oziroma, če razvijamo v programskem jeziku *Java,* lahko to storimo z namensko knjižnico *spring-xd-rest-client*, ki pa je zgolj posrednik do *REST* aplikacijskega vmesnika platforme, kateri je privzeto dosegljiv na vratih 9393. Preko slednjega je mogoča uporaba vseh ponujenih funkcionalnosti platforme, aplikacijski programski vmesnik pa nam rezultate povpraševanj vrača zgolj v *JSON* formatu. Na ta način je zagotovljena neodvisna integracija platforme v različne aplikacije, saj je proženje poizvedb preko *HTTP* protokola in obdelovanje sporočil v *JSON* formatu enostavno, ne glede na programski jezik ter arhitekturo obstoječe aplikacije.

# 4 IZDELAVA PRIMERA UPORABE

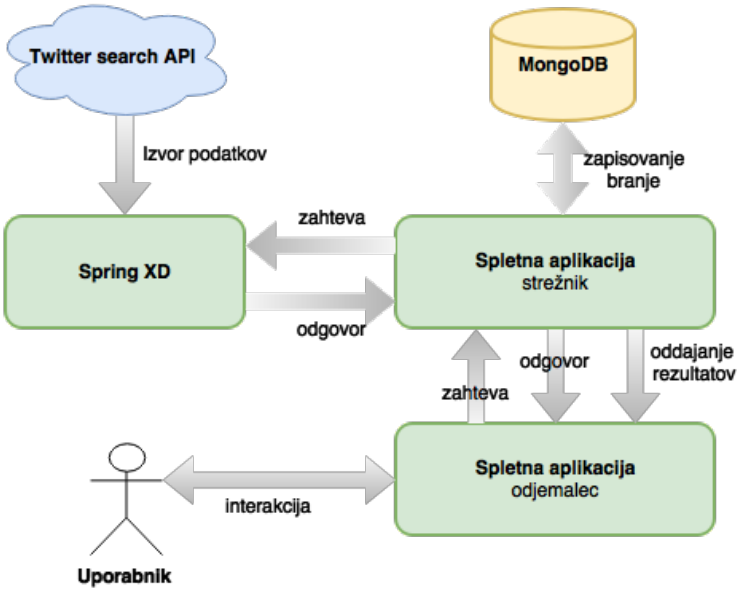
Cilj diplomskega dela je med drugim bil tudi razvoj spletne aplikacije za analitiko podatkov v realnem času v povezavi s platformo *Spring XD*. Z razvojem pa smo pridobili še praktične izkušnje pri uporabi platforme ter jo bolje razumeli.

## 4.1 Analiza problema

Cilj razvoja praktičnega primera uporabe platforme *Spring XD* je razvoj spletne aplikacije, ki uporabniku omogoča ustvarjanje podatkovnih tokov, kateri črpajo podatke iz *Twitter search* spletnega programskega vmesnika in so filtrirani na podlagi vnesenih ključnih besed. Uporabnik ima na voljo izbiro, katere izmed analitičnih metrik se bodo izvajale nad ustvarjenim podatkovnim tokom. Rezultati slednjih pa so uporabniku predstavljeni v obliki grafov.

## 4.2 Načrtovanje

S postavljenimi cilji, poznavanjem platforme ter analitike podatkov v realnem času, smo pridobili dovolj širok pogled na problem, da smo si lahko zastavili osnovno arhitekturo spletne aplikacije (glej Slika 4.1).




Slika 4.1: Arhitektura spletne aplikacije

Platforma *Spring XD* bo delovala v samostojnem načinu ter iz *Twitter search* spletnega aplikacijskega vmesnika pridobivala podatkovni tok. Strežniški del spletne aplikacije bo razvit s pomočjo odprtokodnega ogrodja *Spring Boot*, ki bo prav tako deloval kot samostojni proces znotraj v spletno aplikacijo integriranega spletnega strežnika *Undertow*. Komunikacija med spletno aplikacijo in platformo bo potekala preko *REST* protokola. Vrednosti metrik bo spletna aplikacija pridobivala iz platforme na intervalu 5 sekund. Za trajno hranjenje podatkov bo spletna aplikacija uporabljala dokumentno orientirano podatkovno bazo *MongoDB*. V sklopu spletne aplikacije bo razvit tudi spletni aplikacijski vmesnik, ki bo izpostavljen za potrebe odjemalčevega dela spletne aplikacije. Tudi ta komunikacija bo potekala po protokolu *REST*. Za pošiljanje rezultatov analitičnih metrik iz strežnika na odjemalec pa bo vzpostavljena *websocket* povezava. Prejeti podatki bodo v odjemalčevem spletnem brskalniku predstavljeni v obliki grafov, katere bomo izrisovali s pomočjo *JavaScript* knjižnice *d3.js*, celotni odjemalčev del spletne aplikacije pa bo razvit s pomočjo ogrodja *AngularJS*.

Kot stranski produkt pri razvoju spletne aplikacije za analitiko podatkov v realnem času s *Spring XD*, bo razvit tudi dodaten modul. Modul bo izvajal analizo čustev v besedilu ter tvitom določal polariteto.

## 4.3 Vzpostavitev delovnega okolja

Za potrebe razvoja smo si na osebnem računalniku namestili ter pognali zastonjsko podatkovno bazo *MongoDB* ter platformo *Spring XD*. Prav tako smo si prenesli in namestili akademsko različico integriranega razvojnega okolja *IntelliJ IDEA*. Znotraj *IntelliJ IDEA* smo s pomočjo vgrajenega čarovnika za razvoj aplikacij baziranih na *Spring Boot* ogrodju ustvarili nov projekt. Poskrbeli smo tudi za ustrezen nadzor različic programske kode, za kar smo uporabili *Git* repozitorij. Po navodilih za razvoj modulov za platformo *Spring XD*, katera so zapisana v uradni dokumentaciji platforme, smo znotraj *IntelliJ IDEA* ustvarili še projekt za razvoj modula, pri katerem smo prav tako poskrbeli za nadzor različic programske kode. S tem imamo vzpostavljeno delovno okolje za razvoj spletne aplikacije za analitiko podatkov v realnem času s *Spring XD*.

## 4.4 Implementacija

### 4.4.1 Spletna aplikacija

Večji del priprave projekta za razvoj spletne aplikacije je namesto nas opravil čarovnik za ustvarjanje projektov, integriran v razvojno okolje *IntelliJ IDEA,* kateri je ustvaril projekt z orodjem *Gradle*. V konfiguracijsko datoteko *build.gradle* smo dodali še odvisnosti za knjižnice, potrebne za razvoj, ter namesto v projekt vgrajenega aplikacijskega strežnika *Tomcat* uporabili aplikacijski strežnik *Undertow (glej Programska koda 4.1)*.

```
buildscript {
    …
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")
    }
    configurations {
        compile.exclude module: "spring-boot-starter-tomcat"
        compile.exclude module: "slf4j-log4j12"
    }
}
…
dependencies {
    compile("org.springframework.boot:spring-boot-starter-undertow")
    compile("org.springframework.boot:spring-boot-starter-actuator")
    compile("org.springframework.boot:spring-boot-starter-data-mongodb")
    compile("org.springframework.boot:spring-boot-starter-web")
    compile("org.springframework.boot:spring-boot-starter-security")
    compile("org.springframework.xd:spring-xd-rest-client:1.0.4.RELEASE")
    compile("org.springframework.boot:spring-boot-starter-data-jpa")
    compile("org.hsqldb:hsqldb:2.3.2")
    compile("org.springframework.boot:spring-boot-starter-websocket")
    compile("org.springframework:spring-messaging")
    testCompile("org.springframework.boot:spring-boot-starter-test")
    testCompile("org.apache.commons:commons-lang3:3.0")
}
…
```

Programska koda 4.1: Izpostavljeni spremenjeni deli konfiguracijske datoteke build.gradle

Konfiguracija spletne aplikacije zajema tri konfiguracijske razrede: *DevMongoDBFactoryConfig*, *WebConfig* in *WebSocketConfig*. Prvi služi za konfiguracijo vtičnika za povezavo s podatkovno bazo *MongoDB* (glej Programska koda 4.2), drugi služi za nastavitev, katero HTML datoteko naj aplikacija servira ob prihodu na stran ( glej Programska koda 4.3), tretji pa je namenjen za konfiguracijo uporabe *websocket* povezave (glej Programska koda 4.4).

```
@Configuration
@Profile("dev")
@EnableMongoAuditing
public class DevMongoDBFactoryConfig {

   @Bean
   public MongoDbFactory mongoDbFactory() throws Exception {
      return new SimpleMongoDbFactory(new MongoClient(), "realtimeanalytics");
   }

   @Bean
   MongoTemplate mongoTemplate() throws Exception {
      return new MongoTemplate(mongoDbFactory());
   }
}
```

Programska koda 4.2: Konfiguracijski razred DevMongoDBFactoryConfig

```
@Configuration
public class WebConfig {

   @Bean
   public WebMvcConfigurerAdapter serveIndex() {
      return new WebMvcConfigurerAdapter() {

         @Override
         public void addViewControllers(ViewControllerRegistry registry)
            registry
                  .addViewController("/app")
                  .setViewName("forward:/templates/app/index.html");

         }
      };
   }
```

Programska koda 4.3: Konfiguracijski razred WebConfig

```
@Configuration
@EnableWebSocket
@EnableWebSocketMessageBroker
public class WebSocketConfig extends AbstractWebSocketMessageBrokerConfigurer {

   @Override
   public void configureMessageBroker(MessageBrokerRegistry registry) {
      registry.enableSimpleBroker("/stream");
      registry.setApplicationDestinationPrefixes("/api");
   }

   @Override
   public void registerStompEndpoints(StompEndpointRegistry registry) {
      registry.addEndpoint("/ws"). withSockJS();
   }
}
```

Programska koda 4.4: Konfiguracijski razred WebSocketConfig

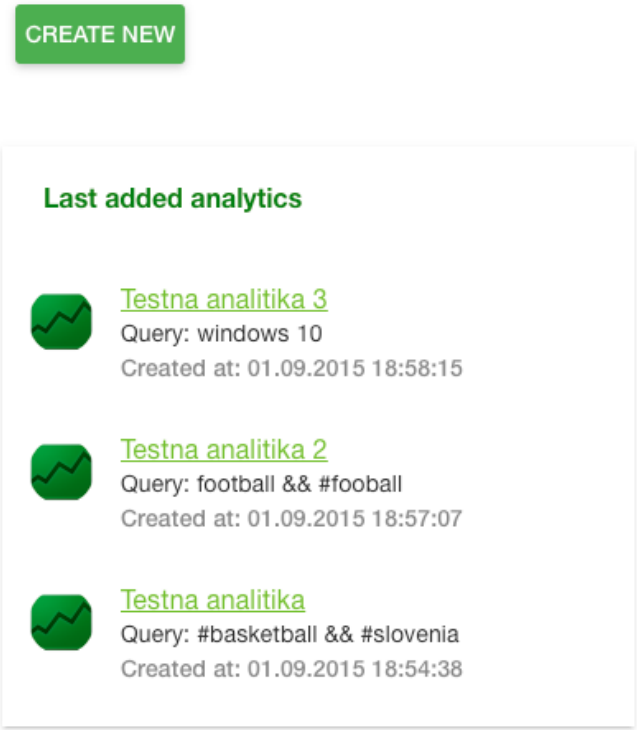
Spletni aplikacijski programski vmesnik, na katerega se bo povezoval odjemalčev del spletne aplikacije, smo implementirali v razredu *StreamController* (glej Priloga A). Vmesnik omogoča ustvarjanje novih tokov (metoda *createStream*) ter pridobivanje seznama vseh obstoječih tokov (metoda *getAllStreams)*. Metoda *createStream* kot vsebino zahtevka prejme *JSON* objekt, katerega na vhodu pretvori v *Java* objekt ter ga s pomočjo razreda *SpringXDService* (glej Priloga B) najprej trajno hrani v podatkovno bazo, nato pa ga z uporabo razredov iz knjižnice *Spring XD rest client* ustvari še na platformi *Spring XD*. Zatem se kliče še metoda *schedule*, katera doda ustvarjen tok na seznam že obstoječih tokov, obenem pa zažene novo nalogo (angl. task), ki teče v svoji niti in periodično prebira vrednosti analitičnih metrik s poizvedbami na *Spring XD platformo*. Pridobljene vrednosti pošlje preko *websocket* povezave v uporabnikov brskalnik (glej Programska koda 4.5).

```
public void run() {
      messageTemplate.convertAndSend("/stream/"+stream.getId()+"/lac",
springXDService.fetchLanguageCounterData(stream));
      messageTemplate.convertAndSend("/stream/"+stream.getId()+"/twc",
springXDService.fetchTweetsCounterData(stream));
      messageTemplate.convertAndSend("/stream/"+stream.getId()+"/tac",
springXDService.fetchTagCounterData(stream));
      messageTemplate.convertAndSend("/stream/"+stream.getId()+"/sea",
springXDService.fetchSentimentAnalyticsCounterData(stream));
   }
```

Programska koda 4.5: Metoda, ki pošilja vrednosti pridobljene iz Spring XD

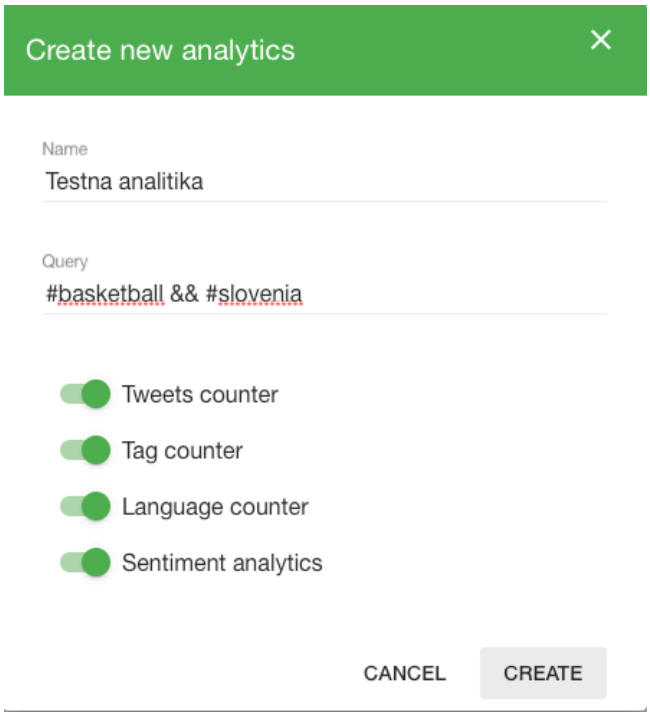
Odjemalčeva stran spletne aplikacije je razvita s pomočjo ogrodja *AngularJS*, za lažje grafično oblikovanje in lepšo prezentacijo pa smo uporabili modul *Angular Material*. Datoteka *app.js* skrbi za inicializacijo ter zagon odjemalčeve aplikacije v trenutku, ko se vsi potrebni viri prenesejo iz strežnika na odjemalca. V njej smo konfigurirali tudi osnovno barvno shemo grafične predloge ter vključili module, katere bomo potrebovali za razvoj aplikacije.

Razvili smo dva kontrolerja (angl. controller) izmed katerih eden skrbi za prikazovanje in osvežitev stranskega prikaza (glej Slika) – *SideCtrl*, drugi – *MainCtrl* pa skrbi za vzpostavitev *websocket* povezave s strežnikom ter prejemanjem podatkov.




Slika 4.2: Stranski prikaz

V stranskem prikazu se v obliki seznama izpisujejo zadnje ustvarjeni tokovi oziroma metrike ter osnovni podatki o njih. Ob kliku na ime se nam v osrednjem delu pričnejo izrisovati grafi metrik izbranega toka. Nad seznamom je umeščen gumb za ustvarjanje novega toka. Ob kliku na slednjega se odpre pojavno okno z vnosno formo, kot je prikazano na sliki 4.3. Od uporabnika je zahtevan vnos poljubnega imena za tok, iskalni niz po katerem želi uporabnik filtrirati tvite ter označba, katere vse analitične metrike bi želeli, da se izvajajo. Po kliku na gumb *Create* se izvede programska koda za pošiljanje zahtevka na spletni aplikacijski programski vmesnik.




Slika 4.3: Pojavno okno za ustvarjanje nove analitike

Za prikaz pojavnega okna za ustvarjanje novega toka skrbi funkcija spisana znotraj kontrolerja *SideCtrl,* katera za prikaz dialoga uporabi modul *ngDialog*. Logika modula nato poskrbi, da se v pojavno okno naloži definirana *HTML* predloga in pojavno okno poveže s kontrolno funkcijo (glej Programska koda 4.6). V kontrolni funkciji smo z uporabo *AngularJS* objekta *$http* spisali rutino za  pošiljanje podatkov na *REST* spletni aplikacijski vmesnik v namen ustvarjanja novega toka.

```
//show dialog
    $scope.showDialog = function(ev) {

      $mdDialog.show({
        controller: DialogController,
        templateUrl: 'views/createanalytics.html',
        parent: angular.element(document.body),
        targetEvent: ev,
        clickOutsideToClose:true
      })
    };
```

Programska koda 4.6: Funkcija za prikaz pojavnega okna

V *MainCtrl* kontrolerju je spisana programska koda, ki s pomočjo knjižnic *Sock.js* ter *STOMP.js* vzpostavi povezavo s strežniškim delom spletne aplikacije (glej Programska koda 4.7).

```
var socket = new SockJS('http://localhost:8080/ws');
var stompClient = Stomp.over(socket);
stompClient.connect({}, function(frame) {});
```

Programska koda 4.7: Vzpostavitev websocket povezave

V tem kontrolerju smo prav tako spisali funkcijo za prijavo na websocket kanale, preko katerih nam strežniški del spletne aplikacije oddaja rezultate metrik (glej Programska koda 4.8).

```
//function for subscribing to stream of analytics results
    var subscribe = function (stream) {
        if(typeof socket !== 'undefined' && typeof stompClient !== 'undefined') {
          if(typeof subsriptionLac !== 'undefined') {
            subsriptionLac.unsubscribe();
          }
          subsriptionLac = stompClient.subscribe('/stream/'+stream.id+'/lac', callbackLac);
          if(typeof subsriptionTwc !== 'undefined') {
            subsriptionTwc.unsubscribe();
          }
          subsriptionTwc = stompClient.subscribe('/stream/'+stream.id+'/twc', callbackTwc);
          if(typeof subsriptionTac !== 'undefined') {
            subsriptionTac.unsubscribe();
          }
          subsriptionTac = stompClient.subscribe('/stream/'+stream.id+'/tac', callbackTac);
          if(typeof subscriptionSea !== 'undefined') {
            subscriptionSea.unsubscribe();
          }
          subscriptionSea = stompClient.subscribe('/stream/'+stream.id+'/sea', callbackSea);
        } else {
          subscribe();
        }
    };
```

Programska koda 4.8: Funkcija za prijavo na websocket kanale

Za potrebe preklapljanja med pregledi rezultatov različnih tokov in posledično prijavljanje in odjavljanje iz različnih kanalov, smo s pomočjo objekta *$watch* spisali še poslušalca na spremenljivko *currentAnalytic* in v njem klicali funkcijo za prijavo na *websocket* kanale (glej Programska koda 4.9).

```
//watch for change of current analytic. in case of change, unsubscribe from old one and subscribe
to new one
    $rootScope.$watch('currentAnalytic', function(newVal, oldVal) {
      if(oldVal==null && newVal!=null) {
        subscribe(newVal);
      } else if (newVal!=null) {
        stompClient.unsubscribe();
        subscribe(newVal);
      }
    });
```

Programska koda 4.9: Poslušalec spremenljivke *currentAnalytic*

Prikaz rezultatov v obliki grafov smo izvedli s pomočjo knjižnice *d3.js* ter uporabo *AngularJS* direktiv. Direktive so lastno ustvarjene *HTML* značke, katere prevajalnik ogrodja *AngularJS* prepozna in jih poveže s pripadajočo programsko logiko, *HTML* predlogo ipd. Z njihovo uporabo smo tako razvili neodvisne lastne elemente za različne tipe grafov (stolpast, mehurčni), kateri skrbijo za prikaz.

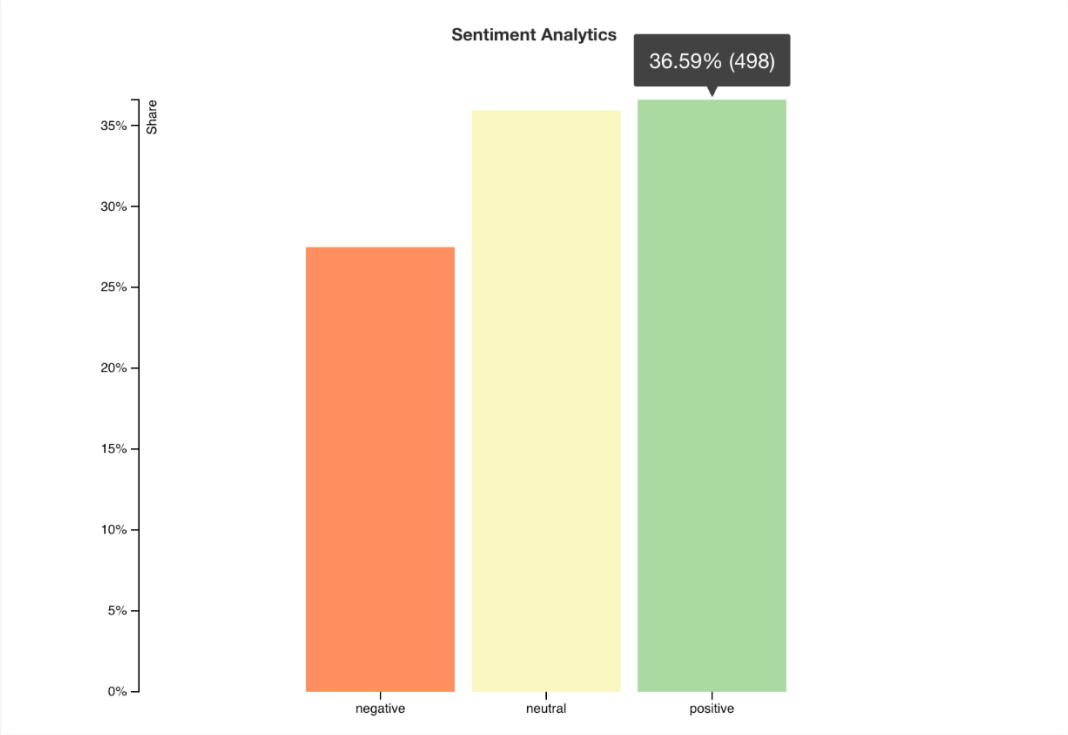
Direktive imajo posebno, ogrodju lastno definirano strukturo. V grobem je to funkcija, ki vrača objekt z naborom določenih atributov in njim pripadajočim vrednostim. Za naše potrebe smo spisali direktive s sledečimi parametri (glej Programska koda 4.10).

```
angular.module('appApp')
  .directive('barChart', function () {
    return {
      restrict: 'E',
      scope: {
        data: '=',
        type: '='
      },
      link: function postLink(scope, element, attrs) {
              …..
              …..
      }
    };
  });
```

Programska koda 4.10: Osnova direktive

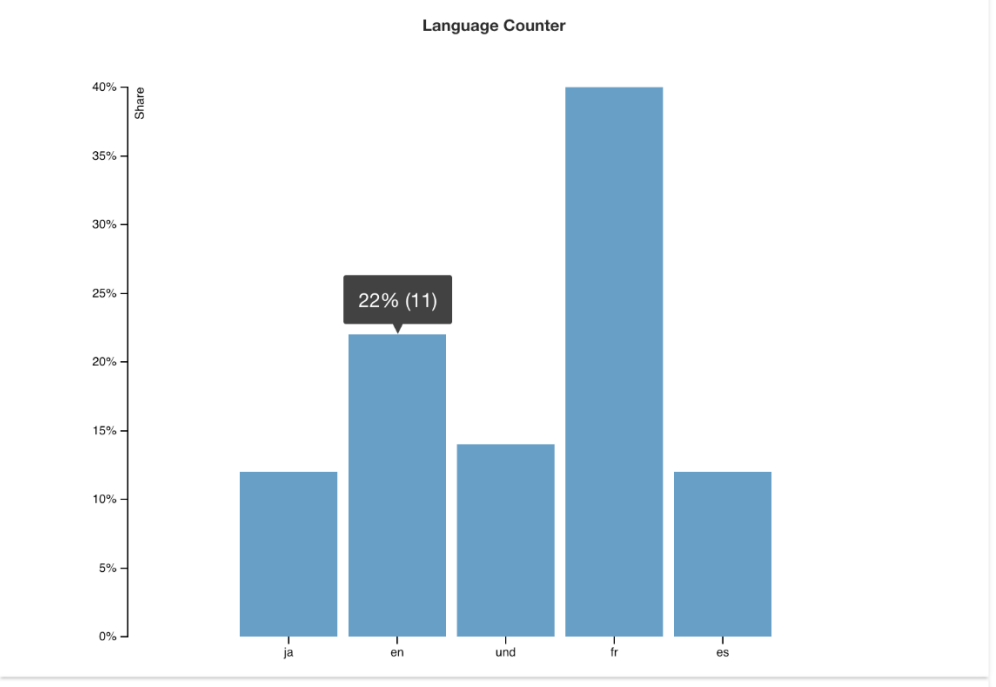
V našem primeru smo uporabili obvezni atribut *restrict,* s katerim določimo možnost uporabe direktive. V primeru, da je vrednost atributa enaka *E,* pomeni, da lahko direktivo uporabimo zgolj kot samostojno značko (<*bar-chart*></*bar-chart*>). Če je vrednost enaka *A,* lahko direktivo uporabimo samo kot atribut v kakšni drugi *HTML* znački. V kolikor je vrednost enaka *AE,* pa lahko direktivo uporabimo na kateregakoli izmed prej opisanih načinov. Uporabili smo tudi opcijska atributa *scope* ter *link*. Prvi omogoča, da direktivi omogočimo dodajanje lastnih parametrov, kateri v našem primeru služijo za povezovanje direktive s podatki. Drugi pa omogoča manipulacijo *DOM* drevesa preko *JavaScript* funkcije. Slednjo smo izkoristili ravno v ta namen, saj smo kot vrednost parametra vpisali funkcijo, ki s pomočjo knjižnice *d3.js* izriše graf na zaslon (glej Priloga C).

Končen rezultat je viden na slikah 4.4, 4.5 in 4.6.




Slika 4.4: Stolpast prikaz rezultatov analize čustev iz besedila


Slika 4.5: Mehurčni prikaz pogostosti uporabe oznak




Slika 4.6: Stolpast prikaz pogostosti različnih jezikov

### 4.4.2 Spring XD modul

Projekt za razvoj *Spring XD* modula smo ustvarili s pomočjo orodja *Maven*. V projektno *pom* datoteko (glej Programska koda 4.11) smo dodali *parent* element, od katerega naš modul deduje funkcionalnosti za pravilno delovanje. Vključili smo še definicije odvisnosti za knjižnice, potrebne za razvoj. Poleg za delovanje modula nujno potrebne knjižnice ogrodja *Spring Integration* smo vključili še *Stanford CoreNLP,* katere funkcionalnosti smo uporabili za napovedovanje polaritete tvitov.

```
<parent>
   <groupId>org.springframework.xd</groupId>
   <artifactId>spring-xd-module-parent</artifactId>
   <version>1.1.0.RELEASE</version>
</parent>
<dependencies>
   <dependency>
      <groupId>org.springframework.integration</groupId>
      <artifactId>spring-integration-java-dsl</artifactId>
      <version>1.0.0.RELEASE</version>
   </dependency>
   <dependency>
      <groupId>edu.stanford.nlp</groupId>
      <artifactId>stanford-corenlp</artifactId>
      <version>3.5.2</version>
   </dependency>
   <dependency>
      <groupId>edu.stanford.nlp</groupId>
      <artifactId>stanford-corenlp</artifactId>
      <version>3.5.2</version>
      <classifier>models</classifier>
   </dependency>
</dependencies>
```

Programska koda 4.11: Izsek konfiguracijske datoteke orodja Maven

V datoteko *Settings.properties* smo zapisali konfiguracijo knjižnice za napovedno analitiko, s katero definiramo na kak način naj bo vhodno besedilo tvita obdelano –prvi trije parametri ter kateri model naj bo apliciran nad temi podatki – četrti parameter (glej Programska koda 4.12).

```
annotators = tokenize, ssplit, parse, sentiment
```

Programska koda 4.12: Konfiguracija knjižnice Stanford CoreNLP

Sledila je konfiguracija samega modula. To smo storili v konfiguracijskem razredu ter v datoteko *spring-module.properties* zapisali samo pot do glavnega paketa izvorne kode modula. V našem primeru je to bilo:

```
base_packages=si.um.vrbancic
```

Programska koda 4.13: Konfiguracija modula

Vsa programska logika se nahaja v preostalih dveh razredih, poimenovanih *Prediction* ter *SentimentPredicton*. Prvi služi za prejem vsebine sporočila iz vira podatkov. V našem primeru prejme tvit v formatu *JSON,* katerega vsebino preoblikujemo v zbirko parov ključ – vrednost. Besedilo tvita nato pošljemo kot parameter metodi razreda, ki opravlja klasifikacijo, ta pa nam vrne številčno vrednost 1, 2 ali 3 (negativno, nevtralno, pozitivno). Glede na prejeto številčno vrednost nato zbirki parov podatkov dodamo nov par s ključem *sentiment* in vrednostjo *negative*, *neutral* ali *positive. Zbirko nato pretvorimo nazaj v JSON obliko in pošljemo na izhod modula. (glej Programska koda 4.14)*.

```
Map<String, Object> tweet = mapper.readValue(
                    payload,
                    new TypeReference<Map<String, Object>>() {}
        );
        prediction = SentimentPrediction.predictSentiment(tweet.get("text").toString());
            if(prediction == 1)
                tweet.put("sentiment", "negative");
            else if(prediction == 2)
                tweet.put("sentiment", "neutral");
            else
                tweet.put("sentiment", "positive");

         return mapper.writeValueAsString(tweet).toString();
```

Programska koda 4.14: Izsek programske kode iz razreda Prediction

Metoda *predictSentiment*, ki je prejela besedilo tvita z namenom klasifikacije njegovega besedila glede na čustva, se nahaja v razredu *SentimentPrediction.java*. Po inicializaciji objekta *StanfordCoreNLP* smo najprej preverili, če ima prejeto besedilo sploh kakšno vrednost. Zatem smo s pomočjo objekta *StanfordCoreNLP* obdelali vhodno besedilo ter s pomočjo razreda *RNNSentimentCoreAnnotations* dobili ocenjeno polariteto za podano besedilo (glej Programska koda 4.16).

```
Annotation annotation = coreNLP.process(tweet);
for(CoreMap sentence : annotation.get(CoreAnnotations.SentencesAnnotation.class)){
  Tree tree = sentence.get(SentimentCoreAnnotations.SentimentAnnotatedTree.class);
   int sentiment = RNNCoreAnnotations.getPredictedClass(tree);
       String partText = sentence.toString();
          if(partText.length() > longest) {
             mainSentiment = sentiment;
             longest = partText.length();
          }
}
```

Programska koda 4.15: Izsek programske kode iz razreda SentimentPrediction

Tako smo razvili modul, kateri bo vsakemu tvitu določil polariteto v realnem času. Na koncu moramo projekt še samo zapakirati v izvedljivo *jar* datoteko, kar storimo z *Maven* ukazoma *clean* in *package*. Zatem smo po postopku, opisanem v poglavju 3, namestili modul v platformo *Spring XD*.

## 4.5 Uporabljene tehnologije

Pri razvoju primera uporabe smo uporabili tehnologije bazirane na programskem jeziku *Java*. Namestili smo *Java Development Kit 8*, katerega smo potrebovali za razvoj spletne aplikacije in modula ter delovanje platforme *Spring XD* verzije *1.2.1.RELEASE*. Pri razvoju spletne aplikacije smo uporabili ogrodji *Spring Boot* verzije *1.2.3.RELEASE* ter *AngularJS* verzije *1.3.0*. Pri razvoju modula za platformo *Spring XD* pa smo uporabili ogrodje *Spring integration* verzije *1.0.0.RELEASE* ter knjižnico *Stanford CoreNLP* verzije *3.5.2*. Za potrebe trajnega hranjenja podatkov smo uporabili dokumentno orientirano podatkovno bazo *MongoDB* verzije *3.0.6*.

# 5 SKLEP

Skozi diplomsko delo smo se spoznali s teorijo analitike podatkov v realnem času, razjasnili pojem »realni čas«, predstavili faze analitike podatkov v realnem času, razložili analitiko besedil, se podrobneje ukvarjali z analizo čustev v besedilu ter poglavje zaključili s predstavitvijo primerov uporabe. V naslednjem poglavju smo podrobneje preučili izbrano platformo *Spring XD,* začenši s samo arhitekturo ter načini delovanja. Opisali in s primeri podprli smo proces ustvarjanja tokov in tap-ov ter razložili delovanje računalniških poslov. Predstavili smo vlogo modulov, se podrobneje posvetili razvoju le-teh ter v tabeli na kratko razložili funkcije osnovnih. Raziskali smo področje analitike v sklopu platforme ter s primeri prikazali možnost uporabe analitičnih metrik. Kot zadnje pa smo spoznali načine upravljanja platforme.

Tako na teoretičnem nivoju kot tudi praktičnem primeru uporabe smo izkusili kompleksnost analitike podatkov v realnem času, kot tudi potencial ki ga ta prinaša. Čeprav smo razvoj skrbno načrtovali in se ga lotili sistematično v skladu z dobrimi praksami, smo kljub temu naleteli na prepreke predvsem zaradi sicer razumljivo napisane, a na trenutke premalo natančne oz. podrobne dokumentacije. To lahko seveda pripišemo dejstvu, da je *Spring XD* nova, še neuveljavljena platforma.

Skozi razvoj smo se srečali tudi s sorodno tematiko in sicer z grafično predstavitvijo podatkov, katero bi si veljalo tudi podrobneje pogledati. Ta je tako kot sistemi in platforme za analitiko podatkov v vedno večjem in hitrejšem razvoju, saj nudi pogled na podatke iz nekoliko netradicionalne perspektive.

# VIRI

[1] EMC Education Services, Data Science and Big Data Analytics, Wiley, 2015.

[2] M. Barlow, Real-Time Big Data Analytics, O'Reilly, 2011.

[3] M. C. A. D. Michael Minelli, Big Data, Big Analytics: Emerging Business Intelligence ana Analytic Trends for Today's Businesses, Wiley, 2013.

[4] W. Medhat, „ScienceDirect,“ 8 9 2014. [Elektronski]. Available: http://www.sciencedirect.com/science/article/pii/S2090447914000550. [Poskus dostopa 12 08 2015].

[5] Lexalytics, „Semantria,“ Lexalytics, 2015. [Elektronski]. Available: https://semantria.com/sentiment-analysis. [Poskus dostopa 26 08 2015].

[6] J. Bell, Machine Learning Hands-On for Developers and Technical Professionals, Indianapolis, Ind.: Wilez, 2015.

[7] Pivotal inc., „Spring XD reference,“ 2015.

# Priloga A

## StreamController.java

```
package si.greg.app.controller.api;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.messaging.simp.SimpMessageSendingOperations;
import org.springframework.scheduling.TaskScheduler;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.scheduling.support.CronTrigger;
import org.springframework.web.bind.annotation.*;
import si.greg.app.domain.Stream;
import si.greg.app.service.SpringXDService;
import si.greg.app.service.StreamService;
import si.greg.app.task.FetchDataTask;
import si.greg.app.utils.StreamState;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ScheduledFuture;

@RestController
@RequestMapping(value = "/api/stream")
public class StreamController {

    private TaskScheduler taskScheduler;
    private SimpMessageSendingOperations messagingTemplate;
    private SpringXDService springXDService;
    private StreamService streamService;

    private Map<Stream, ScheduledFuture<?>> mapOfScheduledTasks;

    @Autowired
    StreamController(
            @Qualifier("taskScheduler") TaskScheduler taskScheduler,
            SimpMessageSendingOperations messagingTemplate,
            SpringXDService springXDService,
            StreamService streamService) {
        this.taskScheduler = (taskScheduler);
        this.messagingTemplate = messagingTemplate;
        this.springXDService = springXDService;
        this.streamService = streamService;
        this.mapOfScheduledTasks = new HashMap<>();
    }
```

```
void schedule(Stream stream) {
        mapOfScheduledTasks.put(stream, this.taskScheduler.schedule(new
FetchDataTask(stream, messagingTemplate, springXDService), new CronTrigger("0/5 * * * *
*")));
    }

    void unSchedule(Stream stream) {
        for(Map.Entry<Stream, ScheduledFuture<?>> entry : mapOfScheduledTasks.entrySet()) {
            if(entry.getKey().getId().equals(stream.getId())){
                entry.getValue().cancel(true);
                mapOfScheduledTasks.remove(entry.getKey());
            }
        }
    }

    @RequestMapping(method = RequestMethod.POST, consumes ="application/json",
produces = "application/json", headers="Accept=application/json")
    public Stream createStream(@RequestBody Stream stream) throws
UnsupportedEncodingException {
        Stream returnStream = springXDService.createStream(stream);
        schedule(returnStream);
        return returnStream;
    }

    @RequestMapping(value = "/get", method = RequestMethod.GET)
    public List<Stream> getAllStreams() {
        return streamService.getStreams();
    }
}
```

# Priloga B

## SpringXD.service.java

```
package si.greg.app.service;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.messaging.simp.SimpMessagingTemplate;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;
import org.springframework.xd.rest.client.AggregateCounterOperations;
import org.springframework.xd.rest.client.impl.SpringXDException;
import org.springframework.xd.rest.client.impl.SpringXDTemplate;
import org.springframework.xd.rest.domain.metrics.AggregateCountsResource;
import si.greg.app.domain.Stream;
import si.greg.app.repository.StreamRepository;
import si.greg.app.utils.*;

import java.io.UnsupportedEncodingException;
import java.util.Calendar;
import java.util.Date;
import java.util.Map;

@Service
public class SpringXDService {

   @Value("${rta.springxd}")
   private String springXd = "http://127.0.0.1:9393/";

   @Autowired
   private StreamRepository streamRepository;

   private SpringXDTemplate xdTemplate = XDClient.getInstance(springXd);

   @Autowired
   public SpringXDService(SimpMessagingTemplate messagingTemplate) {
      SimpMessagingTemplate messagingTemplate1 = messagingTemplate;
      ObjectMapper objectMapper = new ObjectMapper();
   }

public Stream createStream(Stream stream) throws SpringXDException,
UnsupportedEncodingException {
      stream = streamRepository.save(stream);
      //create stream
      xdTemplate.streamOperations().createStream(
                  "id" + stream.getId(), //stream name
                  StreamUtil.generateStreamDefinition(stream), //stream definition
                  true //deploy
            );
```

```
        //create and deploy Counters based on user selection
        for (int i = 0; i < stream.getAnalyticsTypes().length; i++) {
            switch (stream.getAnalyticsTypes()[i]) {
                case "TweetsCounter":
                    createTweetCounter(stream);
                    break;
                case "TagCounter":
                    createTagCounter(stream);
                    break;
                case "LanguageCounter":
                    createLanguageCounter(stream);
                    break;
            }
        }
        //set stream state to deployed
        stream.setState(StreamState.getStateValue(StreamState.SState.DEPLOYED));
        //save stream to db and return saved stream
        return streamRepository.save(stream);
    }

    public String fetchLanguageCounterData(Stream stream) {
        Map<String, Double> data = null;
        data = xdTemplate
                .fvcOperations()
                .retrieve("lacid" + stream.getId())
                .getFieldValueCounts();
        if(data == null) {
            return "";
        }
        return MapToJson.toJson(data);
    }

    public String fetchTweetsCounterData(Stream stream) {
        AggregateCountsResource acr = xdTemplate
                .aggrCounterOperations()
                .retrieve(
                        "twcid" + stream.getId(),
                        stream.getCreatedDate(),
                        new Date(),
                        AggregateCounterOperations.Resolution.minute);
        return MapToJson.toJson(acr.getValues());
    }

    public String fetchTagCounterData(Stream stream) {
        Map<String, Double> data = null;
        data = xdTemplate.fvcOperations().retrieve("tacid" + stream.getId())
.getFieldValueCounts();
        if (data == null) {
            return "";
        }
        return MapToJson.toJson(data);
    }
```

```
public String fetchSentimentAnalyticsCounterData(Stream stream) {
        Map<String, Double> data = null;
        data = xdTemplate
                .fvcOperations()
                .retrieve("seaid" + stream.getId())
                .getFieldValueCounts();
        if(data == null) {
            return "";
        }
        return MapToJson.toJson(data);
    }

    private long getCurrentTimeInMilis() {
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(new Date());
        return calendar.getTimeInMillis();
    }

    //methods for creating counters
    public Object createLanguageCounter(Stream stream) throws SpringXDException {
        xdTemplate
                .streamOperations()
                .createStream(
                        "lacid" + stream.getId(),
                        StreamUtil.generateTweetLanguageCounterDefinition(stream),
                        true);
        return "lac created and deployed";
    }

    public Object createTweetCounter(Stream stream) throws SpringXDException {
        xdTemplate
                .streamOperations()
                .createStream(
                        "twcid" + stream.getId(),
                        StreamUtil.generateTweetCounterDefinition(stream),
                        true);
       return "twc created and deployed";
    }

    public Object createTagCounter(Stream stream) throws SpringXDException {
        xdTemplate
                .streamOperations()
                .createStream(
                        "tacid" + stream.getId(),
                        StreamUtil.generateTweetTagCounterDefinition(stream),
                        true
                );
        return "tac created and deployed";
    }
```

```
public Object createSentimentAnalytics(Stream stream) throws SpringXDException {
        xdTemplate
                .streamOperations()
                .createStream(
                        "seaid" + stream.getId(),
                        StreamUtil.generateSentimentAnalyticsCounterDefinition(stream),
                        true
                );

        return "sea created and deployed";
    }

//    methods for undeploy and destroy counters
    public Object destroyTweetLangCounter(Stream stream) throws SpringXDException
{
        xdTemplate
                .streamOperations()
                .undeploy("tweetLangCounter" + stream.getName() + stream.getUserId());

        xdTemplate
                .fvcOperations()
                .delete("tweetLangCounter" + stream.getName() + stream.getUserId());

        return "tweetLangCounter destroyed";
    }

    public Object destroyTweetCounter(Stream stream) throws SpringXDException {
        xdTemplate
                .streamOperations()
                .undeploy("tweetCounter" +  stream.getName() + stream.getUserId());

        xdTemplate
                .aggrCounterOperations()
                .delete("tweetCounter" + stream.getName() + stream.getUserId());

        return "tweetCount destroyed";
    }

    public Object destroyTweetTagCount(Stream stream) throws SpringXDException {
        xdTemplate
                .streamOperations()
                .undeploy("tweetTagCounter" +  stream.getName() + stream.getUserId());

        xdTemplate
                .fvcOperations()
                .delete("tweetTagCounter" + stream.getName() + stream.getUserId());

        return "tweetTagCount destroyed";
    }
}
```

# Priloga C

## barchart.js

```
angular.module('appApp')
  .directive('barChart', function () {
    return {
      restrict: 'E',
      scope: {
        data: '=',
        type: '='
      },
      link: function postLink(scope, element, attrs) {

        var margin = {top: 20, right: 20, bottom: 30, left: 40},
          width = 680 - margin.left - margin.right,
          height = 500 - margin.top - margin.bottom;

        var formatPercent = d3.format('.0%');

        var x = d3.scale.ordinal().rangeRoundBands([0, width], .1, 1);
        var y = d3.scale.linear().range([height, 0]);

        var xAxis = d3.svg.axis().scale(x).orient('bottom');
        var yAxis = d3.svg.axis().scale(y).orient('left').tickFormat(formatPercent);

        //tooltip
        var tip = d3.tip()
          .attr('class', 'sentiment-tooltip')
          .offset([-10, 0])
          .html(function(d) {
          return "<span>" + d3.round(d.share*100, 2) + "%  (" + d.value + ")</span>";
        });

        var svg = d3.select(element[0])
          .append('svg')
          .attr('width', width + margin.left + margin.right)
          .attr('height', height + margin.top + margin.bottom)
          .append('g')
            .attr('transform', 'translate(' + margin.left + ',' + margin.top + ')');

        svg.call(tip);

        //Render graph based on 'data'
        scope.render = function(data) {

          //Set our scale's domains
          x.domain(data.map(function(d) { return d.label; }));
          y.domain([0, d3.max(data, function(d) { return d.share; })]);
```

```
//Redraw the axes
        svg.selectAll('g.axis').remove();

        //X axis
        svg.append('g')
            .attr('class', 'x axis')
            .attr('transform', 'translate(0,' + height + ')')
            .call(xAxis);

        //Y axis
        svg.append('g')
            .attr('class', 'y axis')
            .call(yAxis)
          .append('text')
            .attr('transform', 'rotate(-90)')
            .attr('y', 6)
            .attr('dy', '.71em')
            .style('text-anchor', 'end')
            .text('Share');

        var bars = svg.selectAll('.bar')
          .data(data, function(d) {return d.label});

        //new data
        bars.enter().append('g')
          .append('rect')
          .attr('class', function(d) {
            if(scope.type == 'sentiment') {
              if(d.label == 'negative') {
                return 'bar-negative';
              }
              if(d.label == 'neutral') {
                return 'bar-neutral';
              }
              if(d.label == 'positive') {
                return 'bar-positive'
              }
            } else {
                return 'bar';
            }
          })
          .attr('x', function(d) { return x(d.label); })
          .attr('y', function(d) { return y(d.share); })
          .attr('width', x.rangeBand())
          .attr('height', function(d) { return height - y(d.share); })
          .on('mouseover', tip.show)
          .on('mouseout', tip.hide);

      //removed data
              bars.exit().remove();
```

```
//updated data
        bars
            .transition()
            .duration(1000)
            .attr('height', function(d) { return height - y(d.share); })
            .attr('y', function(d) { return y(d.share); });
       };

       //Watch 'data' and run scope.render(newVal) whenever it changes
       //Use true for 'objectEquality' property so comparisons are done on equality and not
reference
        scope.$watch('data', function(newVal, oldVal){
          console.log('render now!!!');
          if(typeof scope.data !== 'undefined') {
            var sum = 0;
            for(var i=0; i<scope.data.length; i++) {
              sum += scope.data[i].value;
            }

            for(var i=0; i<scope.data.length; i++) {
              scope.data[i].share = scope.data[i].value/sum;
            }

            scope.render(scope.data);
          }
        }, true);
      }
   };
 });
```

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

# IZJAVA O AVTORSTVU

Spodaj podpisani/-a GREGA VRBANČIČ

z vpisno številko E1042540

sem avtor/-ica diplomskega dela z naslovom:

RAZVOJ SPLETNE APLIKACIJE ZA ANALITIKO PODATKOV V REALNEM ČASU S SPRING XD

*(naslov diplomskega dela)*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

red. prof. dr. VILI PODGORELEC

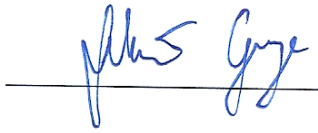
in somentorstvom (naziv, ime in priimek)

_____

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela.
- soglašam z javno objavo elektronske oblike diplomskega dela v DKUM.

V Mariboru, dne 9.9.2015

Podpis avtorja/-ice:



Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

# IZJAVA O USTREZNOSTI ZAKLJUČNEGA DELA

*Podpisani mentor :*

VILI PODGORELEC

*(ime in priimek mentorja)*

*in somentor (eden ali več, če obstajata):*

*(ime in priimek somentorja)*

*Izjavljam (-va), da je študent*

*Ime in priimek:* GREGA VRANČIČ

*Vpisna številka:* E1042540

*Na programu:* ITK UNI

*izdelal zaključno delo z naslovom:*

RAZVOJ SPLETNE APLIKACIJE ZA ANALITIKO PODATKOV V REALNEM ČASU S SPRING XD

*(naslov zaključnega dela v slovenskem in angleškem jeziku)*

DEVELOPMENT OF A WEB APPLICATION FOR REAL-TIME DATA ANALYTICS WITH SPRING XD

v skladu z odobreno temo zaključnega dela, Navodilih o pripravi zaključnih del in mojimi (najinimi oziroma našimi) navodili.

Preveril (-a, -i) in pregledal (-a, -i) sem (sva, smo) poročilo o plagiatorstvu.

Datum in kraj:
MARIBOR, 8.9.2015

Podpis mentorja:

Datum in kraj:

Podpis somentorja (če obstaja):

Priloga:
- Poročilo o preverjanju podobnosti z drugimi deli.«

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

**IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE ZAKLJUČNEGA DELA IN OBJAVI OSEBNIH PODATKOV DIPLOMANTOV**

Ime in priimek avtorja-ice: GREGA VRBANČIČ

Vpisna številka: E1042540

Študijski program: ITK UNI

Naslov zaključnega dela: RAZVOJ SPLETNE APLIKACIJE ZA ANALITIKO PODATKOV V REALNEM ČASU S SPRING XD

Mentor: red. prof. dr. VILI PODGORELEC

Somentor:

Podpisani-a GREGA VRBANČIČ izjavljam, da sem za potrebe arhiviranja oddal elektronsko verzijo zaključnega dela v Digitalno knjižnico Univerze v Mariboru. Zaključno delo sem izdelal-a sam-a ob pomoči mentorja. V skladu s 1. odstavkom 21. člena Zakona o avtorskih in sorodnih pravicah dovoljujem, da se zgoraj navedeno zaključno delo objavi na portalu Digitalne knjižnice Univerze v Mariboru.

Tiskana verzija zaključnega dela je istovetna z elektronsko verzijo elektronski verziji, ki sem jo oddal za objavo v Digitalno knjižnico Univerze v Mariboru.

---

Zaključno delo zaradi zagotavljanja konkurenčne prednosti, varstva industrijske lastnine ali tajnosti podatkov naročnika: _____ ne sme biti javno dostopno do _____(datum odloga javne objave ne sme biti daljši kot 3 leta od zagovora dela).

---

Podpisani izjavljam, da dovoljujem objavo osebnih podatkov, vezanih na zaključek študija (ime, priimek, leto in kraj rojstva, datum zaključka študija, naslov zaključnega dela), na spletnih straneh in v publikacijah UM.

Datum in kraj: MARIBOR, 9.9.2015 Podpis avtorja-ice: 

Podpis mentorja: _____
*(samo v primeru, če delo ne sme biti javno dostopno)*

Podpis odgovorne osebe naročnika in žig:
*(samo v primeru, če delo ne sme biti javno dostopno)* _____