Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Goran Malić

Analiza družbenih medijev

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: prof. dr. Marko Bajec

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Besedilo teme diplomskega dela študent prepiše iz študijskega informacijskega sistema, kamor ga je vnesel mentor. V nekaj stavkih bo opisal, kaj pričakuje od kandidatovega diplomskega dela. Kaj so cilji, kakšne metode uporabiti, morda bo zapisal tudi ključno literaturo.

Zahvala

Za pomoč in podporo se zahvaljujem dragi Andreji in sinu Davidu.

Posebna zahvala gre tudi mentorju prof. dr. Marku Bajcu.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
ACID	atomicity, consistency, isolation, durability)	atomarnost, konsistentnost, neodvisnost, trajnost
API	application programming interface	aplikacijski programski vmesnik
CRUD	create, read, update and delete	ustvari, beri, piši in briši
HTTPS	HyperText transfer protocol secure	zvarovana različica komunikacijskega protokola HTTP
JSON	JavaScript object notation file format	datotečni format JavaScript
NLP	natural language processing	procesiranje naravnega jezika
RDBMS	relational database management system	sistem za upravljanje relacijskih podatkovnih baz
SDK	software development kit	paket orodij za razvoj programske opreme
SQL	structured query language	strukturirani povpraševalni jezik za delo s podatkovnimi bazami
TF-IDF	term frequency and inverse document frequency	statistični podatek, ki nakazuje pomembnost besede v korpusu
TLS	transport layer security	kriptografski protokol za varno komunikacijo na medmrežju
URL	uniform resource locator	enolični krajevnik vira
XML	extensible markup language	razširljivi označevalni jezik

Povzetek

Naslov: Analiza družbenih medijev

Avtor: Goran Malić

Družbeni mediji so vedno bolj priljubljen način za izmenjavo informacij in spremljanje aktualnega dogajanja, zato predstavljajo pomemben vir podatkov, ki imajo tudi veliko uporabno vrednost za izvajanje številnih analiz ter raziskav. Predpogoj za to pa je, da jih znamo sistematično zbirati. Ker se v praksi pogosto večino časa ukvarjamo s procesom zbiranja podatkov in njhovim preoblikovanjem v primerno obliko, želimo v tej nalogi preveriti, kakšne so možnosti za celovit prevzem vsebin na različnih socalnih medijih. Na konkretnem primeru bomo implementirali možnost paralelnega zajema večje količine podatkov in preverili, kako jih lahko smiselno skladiščimo in po njih poizvedujemo. Na koncu bomo omogočili vpogled v podatkovno zbirko tudi z vizualizacijskim orodjem, ki omogoča spremljanje vsebin z minimalnim časovnim zamikom glede na čas nastanka objave na družbenem mediju. V zaključku bomo povzeli izkušnje pri delu z različnimi tehnologijami in navedli možnosti za nadaljno izboljšavo rešitve.

Ključne besede: družbeni mediji, Nifi, Solr, YouTube, API.

Abstract

Title: Social media analysis

Author: Goran Malić

Social media is an increasingly popular way of exchanging information and staying informed about current events, and therefore represent an important source of data, which is also valuable for research and analysis. The prerequisite for this is that we know how to systematically collect data. Since in practice, most of the time we are often dealing with the process of data collection and transformation, we will research the possibilities for comprehensive content acquisition of various social media. In this case study we will implement a solution for parallel data extraction of large data sets and find a way for efficient data storing and querying. In the end, we will access data with a visualization tool that allows data monitoring with minimal delay regarding to the time when content is published on the social media. In conclusion, we will summarize our experience with different technologies and outline the possibilities for further process improvement.

Keywords: social media, Nifi, Solr, YouTube, API.

Poglavje 1

Uvod

Družbeni mediji v sodobnem času hitro pridobivajo na razširjenosti in vedno bolj izpodrivajo druge oblike medijev. Z razmahom mobilnih naprav so danes dostopni vedno in povsod, uporabnikom pa omogočajo aktiven pristop za spremljanje dogajanja. V primerjavi z nekaterimi standardnimi oblikami medijev, kot sta časopis ali televizija, spletni družbeni mediji omogočajo interaktivnost, svobodo, odzivnost in ponujajo možnost dostopa do velikega števila različnih virov informacij. Najbolj priljubljeni družbeni mediji, kot so Facebook, Google, Twitter, LinkedIn ali Instagram, imajo registriranih na stotine milijonov uporabnikov in vsak dan pridobivajo nove. Posledica tega je, da se v ozadju generirajo velike količine podatkov, ki imajo seveda lahko tudi visoko uporabno vrednost.

Možnosti iskanja dodane vrednosti v podatkih, ki nastajajo z uporabo družbenih medijev, so brezmejne in presegajo namen te naloge. Če izpostavimo samo nekatere, so to iskanje vzorcev v podatkih, klasifikacija uporabnikov, analiza sentimenta, pridobivanje mnenj, jezikovna analiza ali preprosto spremljanje priljubljenosti lastnih vsebin. Podatki, ki jih pridobimo iz družbenih medijev, se pogosto uporabljajo v študijah podatkovnega rudarjenja in zahtevajo tehnologijo, ki podpira delo z velikimi podatkovnimi zbirkami, ki jih poznamo pod pojmom „Big Data“.

V praksi se pri tovrstnih podatkovnih analizah večino časa ukvarjamo s samim procesom zbiranja in priprave podatkov, lahko tudi 80 do 90 odstotkov časa, zato se v nalogi želimo dotakniti tega problema. Cilj naloge je preveriti, kakšne so možnosti za dostop do podatkov različnih družbenih medijev in na konkretnem primeru izvesti poskus celovitega zajema vsebine. V nadaljevanju bomo ugotavljali, do kakšnih vsebin in pod kakšnimi pogoji nam različni družbeni mediji sploh omogočajo dostop ter na praktičnem primeru preverili ali je možno celovito pridobivati podatke tudi na tistih vsebinah, ki jih nismo sami ustvarili. Zanimalo nas bo, ali lahko nove objave na družbenem mediju z minimalnim zamikom spremljamo tudi v našem analitčnem okolju. Poseben poudarek pa bo namenjen optimizaciji procesa zajema podatkov na način, ki zagotavlja maksimalen izkoristek aplikacijskega programskega vmesnika (v nadaljevanju „API“) ter minimalno ponavljanje prenosov vsebine, ki jo že imamo.

Pridobljene podatke bomo potem preuredili v čim bolj berljivo obliko, dostopno poljubnim analitičnim orodjem za nadaljno analizo. Na koncu bomo v izbranem orodju predstavili še praktični primer poročila z neko uporabno vrednostjo.

V zaključku naloge bomo povzeli izkušnje pri procesu prevzema podatkov iz družbenih medijev in navedli možnosti za izboljšavo procesa.

Poglavje 2

Pregled družbenih medijev

Raznolikost in neprestan razvoj ter prepletanje storitev družbenih medijev povzroča nekoliko težav pri definiciji, kaj vse lahko uvrstimo v kategorijo družbenih medijev. Literatura navaja nekatere skupne lastnosti. Aplikacije družbenih medijev temeljijo na t.i. Spletu 2.0, ki podpira izmenjavo informacij in sodelovanje med uporabniki spleta. Osnova družbenih medijev je uporabniško generirana vsebina, za sodelovanje na posameznem mediju pa je potreben uporabniški profil, ki ga definira in vzdružje ponudnik spletne storitve. Z medsebojnim povezovanjem profilov ter skupin profilov se vzpodbuja oblikovanje socialnih omrežij.

Vidimo, da pojma „družbeni medij“ ter „socialno omrežje“ nista povsem enaka. Družbeni oziroma socialni medij (angl. „social media“) je platforma oziroma skupek tehnologij, ki posameznikom omogoča mreženje ter deljenje vsebin. Družbeni medij temelji na vsebini, ki jo objavlja posameznik in je dostopna širšemu krogu ljudi, naj bo to blog, slika, video, forumska objava ali kaj podobnega. Socialno omrežje je rezultat uporabe družbenih medijev in predstavlja skupnost posameznikov, ki s povezovanjem v socialna omrežja ustvarjajo neko dodano vrednost. V tem kontekstu lahko govorimo tudi o dejavnosti socialnega mreženja (angl. „social networking“).

Skladno z navedenimi smernicami se strokovnjaki tega področja v grobem strinjajo, da lahko družbene medije klasificiramo v trinajst različnih kategorij. Te so: blogi, profesionalna omrežja, sodelovalno pisanje (Wiki), poslovna omrežja, forumi, t.i. mikroblogi, deljenje slik, recenzije produktov oziroma storitev, označevanje vsebine (angl. „tagging“), družabno igranje, socialna omrežja, deljenje video objav ter virtualni svetovi.

Slika 2.1 prikazuje primer klasifikacije družbenih medijev in pomen posameznih kategorij za operativne procese v podjetju.

	Corporate function					
Type of social media	R&D	Marketing	Customer service	Sales	HR	Organisation
Blogs	◔	◑	◔			
Business networks					●	◔
Collaborative projects	●					
Enterprise social networks	◕				◑	●
Forums	◑	◔	●			
Microblogs		◕	◔		◔	
Photo sharing		◑				
Products/services review	◔	◑		●		
Social bookmarking		◑				
Social gaming		◑				
Social networks	◔	●	◑		◔	◔
Video sharing		●	◔			
Virtual worlds	◔	◕		◔		

Importance: (empty) none or almost none; ◔ low; ◑ medium; ◕ high; ● very high

Slika 2.1: Klasifikacija družbenih medijev in vloga v podjetju [14].

2.1 Zbiranje podatkov

Poleg same vsebine objave lahko zbiramo podatke, kot so naslov, opis, deljenje objav (angl. „shares“), všečki, komentarji, omembe, vtisi, število klikov

oz. ogledov, število sledilcev, ključni pojmi, metapodatki objave ipd. Nekatere podatke lahko pri tem uporabimo neposredno, za druge pa moramo izvesti dodatne analitične procese. Primer je področje tekstovnega rudarjenja, v okviru katerega preoblikujemo podatke v strukturirano obliko in iščemo vzorce v njih. Poleg tega lahko izvajamo analizo sentimenta, pridobivanje mnenj (angl. „opinion mining“), procesiranje naravnega jezika NLP, izračuni metrik, kot je indeks TF-IDF in podobno [18].

Da bi bili analitični modeli čimbolj natančni, želimo iz družbenih medijev prenesti čm več različnih podatkov, ki lahko koristijo v procesu analize. Pomembno je, da je proces prenosa avtomatiziran in da uporabniku rešitve omogoča poljubno izbiro vsebine, ki jo želi analizirati.

V ta namen izkoriščamo spletni aplikacijski programski vmesnik (angl. „Web API“), ki ga ponujajo takorekoč vsi ponudniki družbenih medijev. Njihova dostopnost in učinkovitost se razlikujeta od primera do primera, v splošnem pa velja, da javno dostopno vsebino lahko prevzemamo na vseh družbenih medijih. Vse platforme zahtevajo neko obliko avtentikacije in ob dostopu na API vračajo odgovor v enem izmed standardnih formatov, ponavadi je to datoteka oblike JSON. Pri prevzemu podatkov smo omejeni s kvotami in pravili, ki določajo, v kakšni meri lahko obremenimo API v določenem časovnem intervalu. Te omejitve lahko ponavadi zaobidemo z zakupom dodatnih mesečnih kvot. Z vidika naloge nas zanimajo predsvem platforme, ki ponujajo kar največ vsebine, brez potrebe po dodatnih mesečnih izdatkih.

2.2 Varnost in zasebnost podatkov

Predpisi Evropske unije o varstvu podatkov zagotavljajo zaščito naših osebnih podatkov, kadarkoli se ti zbirajo. Predpisi veljajo za podjetja in organizacije, ki ponujajo svoje storitve v EU, četudi njihov sedež ni v EU, denimo Facebook ali Amazon. Pravico posameznika do varstva osebnih podatkov je

treba spoštovati pri vsakem shranjevanju ali obdelavi podatkov, ki neposredno ali posredno omogočajo njegovo identificiranje, in sicer ne glede na način zbiranja podatkov. Podjetje ali organizacija mora pridobiti uporabnikovo soglasje preden lahko obdela ali ponovno uporabi njegove osebne podatke, razen v posebnih primerih. [10].

Pogoje uporabe internetnih vsebin dodatno ureja uredba GDPR (angl. „General Data Protection Regulation“), ki je stopila v veljavo 25. 5. 2018. Velja za članice EU in za vse, ki kakorkoli poslujejo z njimi. Uredba določa, da morajo biti vsi osebni podatki zbrani in uporabljeni na zakonit način in z omejenim namenom. To pomeni, da se zbrani osebni podatki ne smejo obdelovati v druge namene, za katere niso bili pridobljeni, to je najpogosteje za namene trženja. Poleg tega so določena številna načela ter pravice posameznikov, med katere spada tudi pravica do dostopa, ki posamezniku omogoča, da potrdi vsako obdelavo osebnih podatkov, pri čemer mora biti jasno, kje in s kakšnim namenom se ta izvaja. Vse našteto ima seveda pomemben vpliv tudi za možnost dostopa do osebnih podatkov na družbenih medijih. Uredba je splošno veljavna, uporabljiva in zavezujoča neposredno za vse članice EU, kršitelje pa lahko doletijo tudi zelo visoke denarne kazni [5].

Družbeni mediji so torej obvezani slediti uredbam ter direktivam, zato uporabnikom ponujajo možnost zaščite osebnih podatkov, ki so lahko dostopni le ožjemu krogu izbranih ljudi. Do takih vsebin posledično nimamo dostopa niti z uporabo API-jev, razen seveda če bi se v sistem prijavili kot eden izmed uporabnikov, ki imajo dostop. Pri masovnem zajemu podatkov iz družbenih medijev zato praviloma dostopamo le do javnih profilov oziroma javno dostopnih vsebin. V nekaterih primerih kot je Facebook, so tudi dostopi do javnih vsebin možni šele po njihovem predhodnem pregledu in potrditvi ustreznosti aplikacije. Spet drugje je dostop do javnih vsebin možen že s strinjanjem s sicer izčrpnimi pogoji uporabe.

V splošnem velja, da moramo skrbno paziti na dovoljene pogoje uporabe podatkov tudi pri prevzemu podatkov iz javnih strani. Če podatkovne zbirke uporabimo za analize, katerih rezultat je javno dostopen, morajo biti rezultati predstavljeni v agregirani, anonimizirani obliki, ki ohranja zasebnost uporabnikov.

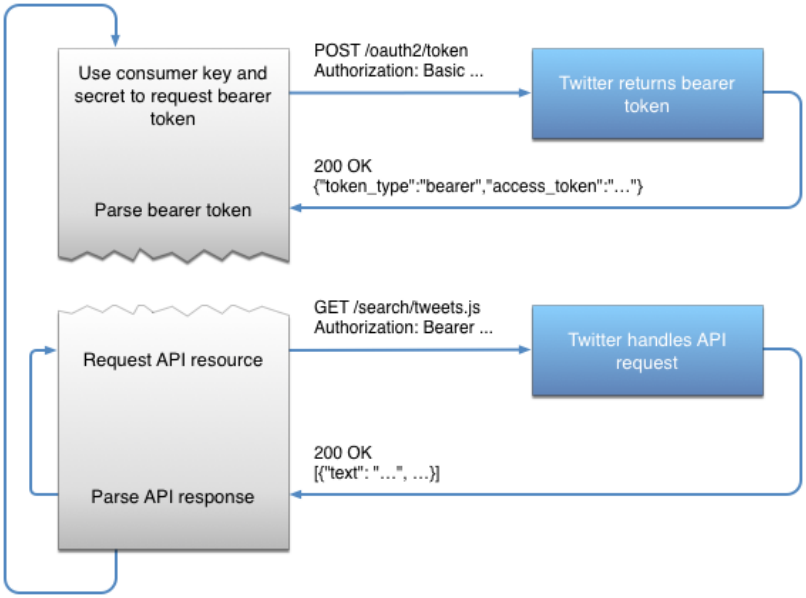
Ker bi lahko nepooblaščen dostop do API-ja kršil tovrstne smernice, je nujno zagotoviti ustrezno stopnjo varnosti pri njihovi uporabi. Tako kot za osebne uporabnike družbenih medijev je tudi za sisteme, ki želijo dostopati do API-ja, predpogoj za uporabo, da opravijo proces avtentikacije in avtorizacije.

Avtentikacija je proces validacije, pri katerem preverjamo, ali je oseba oziroma sistem dejansko tisti, za katerega se predstavlja. Avtorizacija je proces ugotavljanja, katere akcije lahko oseba ali sistem izvaja potem, ko je uspešno opravljen proces avtentikacije. Z namenom zagotavljanja visoke stopnje varnosti pri izvedbi omenjenega procesa večina družbenih medijev podpira uporabo protokola OAuth 2.0. [20].

2.2.1 OAuth 2.0

Protokol OAuth 2.0 je danes eden izmed najpomembnejših protokolov na področju varnosti pri komunikaciji med odjemalci ter spletnimi API-ji. Glavna prednost v primerjavi s starejšimi rešitvami je, da aplikacija od uporabnika nikoli ne zahteva podatkov o uporabniškem imenu ter geslu. Namesto tega se uporabnika preusmeri neposredno na prijavni vmesnik spletnega medija. Uporabnik se s svojimi poverilnicami prijavi neposredno na spletnem mediju, ki ga potem "vpraša", ali uporabljeni aplikaciji tudi v resnici želi dovoliti dostop do izbranih vsebin. Po uporabnikovi potrditvi lahko aplikacija nemoteno izvaja akcije, za katere je dobila pravice. Primer uporabe protokola na družbenem mediju Twitter je prikazan na sliki 2.2.

Prednosti takšnega pristopa so očitne, saj uporabniku svojih poverilnic nikoli



Slika 2.2: Primer procesa avtentikacije po protokolu Oauth 2.0 v okolju Twitter.

ni potrebno zaupati aplikacijam, s čimer so zmanjšane možnosti za zlorabo. Dodatna prednost je, da v primeru izgube zaupanja v aplikacijo, ji uporabnik lahko v svojem profilu odvzame pravico za dostop.

Za komunikacijo med aplikacijo in spletnim servisom je na voljo več načinov odobritev dostopa. Najbolj pogosto sta uporabljena implicitna odobritev ter odobritev dostopa z avtorizacijsko kodo. Prva spada med manj varne opcije in je namenjena uporabi spletnim ter mobilnim aplikacijam, druga pa je namenjena neposredni komunikaciji med strežnikoma in je zato opredeljena kot varna. Spletni servis v tem primeru avtorizacijski ključ pošlje neposredno na zaledni strežnik, ki omogoča varen prenos podatkov ter shranjevanje poverilnic. To je tudi edini način za pridobitev dolgoročnega ključa, ki nam omogoča dostop v daljšem časovnem obdobju [20]. Omenjeni način prijave uporabljamo tudi pri naši rešitvi.

2.3 Primeri družbenih medijev

V nadaljevanju bomo pogledali nekatere najbolj priljublje družbene medije iz različnih kategorij, za katere smo izvedli oceno zahtevnosti implementacije dostopa do podatkov.

2.3.1 Twitter

Twitter je namenjen objavam tako imenovanih mikro blogov, ki so se izkazali kot eden najbolj priljubljenih tipov objav na družbenih medijih. Platforma uporabnikom omogoča branje in objavo kratkih sporočil oziroma „tweetov“, ki vsebujejo do 280 znakov. Objave lahko prihajajo praktično od kogarkoli, uporabniki pa sami izberejo, katerim vsebinam želijo slediti. Sporočila je možno objavljati celotni Twitter skupnosti, zasebno izbranemu uporabniku ali kombinirano, pri čemer je objava vidna samo tistim uporabnikom, ki jim to dovolimo [25].

Ravno kratkost objav je tisto, kar dela okolje Twitter unikatno in privlačno. V poplavi informacij na internetu platforma na nek način prisili uporabnike, da podajajo informacije v strnjeni, faktografski obliki. Namesto podrobnih opisov ali razprav skozi Twitter informacije podajamo v obliki trditve, ideje, misli, dejstva ali vprašanja.

Platforma ni zanimiva samo za zasebne uporabnike ampak tudi za organizacije in podjetja, ki okolje izkoriščajo za mreženje s strankami in pridobivanje njihovega mnenja, za odgovarjanje na vprašanja, pridobivanje novih strank, oblikovanje blagovne znamke, marketing ter oglaševanje. Društva in organizacije platformo izrabljajo za promocijo delovanja čim širšemu krogu ljudi in za izmenjavo mnenj med člani. [25].

Spletni aplikacijski programski vmesnik platforme Twitter je bil eden prvih odprtih ter širše dostopnih na področju družbenih medijev. API omogoča

enostaven prevzem celotne vsebine podatkov, tudi tistih starejših, ki segajo v čas njegovega nastanka leta 2006. Posledica je bila izjemen razmah uporabe podatkov ne samo v poslovne ampak tudi v raziskovalne ter študijske namene. Tako kot pri večini sorodnih priljubljenih rešitvah je to skozi čas botrovalo novim pogojem uporabe in povzročilo številne omejitve dostopov na API.

Kot primer navajamo obdobje med aprilom ter junijem 2018, kjer so skrbniki platforme samo v tem obdobju odstranili 143.000 aplikacij, ki so bile domnevni kršitelj pogojev uporabe. V prvi polovci leta so poostrili postopek registracije in vse obstoječe razvijalce pozvali, naj dopolnijo registracijske podatke, če želijo ohraniti možnost dostopa do podatkov. Poleg tega so uvedli nove omejitve pri številu dovoljenih aplikacij ter možnosti pošiljanja podatkov na omrežje, uporabnikom pa so omogočili, da kar sami prijavijo aplikacije, ki so morebitni kršitelji pravil [30].

Trenutno so ponujeni trije različni dostopi do API-ja, to so Standard, Premium ter Enterprise. Osnovna verzija ponuja dostop do podatkov mlajših od 7 dni in služi učenju ali preverjanju koncepta zajema podatkov. Srednja še vedno ne zahteva plačila in ponuja celovit dostop do zgodovine podatkov. Premijska vključuje tehnično podporo in omogoča uporabo nekaterih naprednejših procesov pri zajemu podatkov, kot so kombinacija različnih filtrov, vzorčenje podatkov ter boljša podpora pri iskanju po zgodovinskih podatkih. [12]

Zastonjska različica določa omejitve prevzema podatkov na osnovi petnajst minutnih časovnih intervalov. V tem časovnem oknu je minimalno število dovoljenih branj 15, pri čemer API dovoljuje večje število cenejših poizvedb in manjše število dražjih. Za izvedbo vsake poizvedbe je zahtevana avtentikacija po protokolu Oauth 2.0 [15].

Twitter je torej odlična platforma za študijo zajema podatkov družbenih medijev, vendar se zanj nismo odločili zaradi izjemne razširjenosti, ki se pozna tudi na način, da ima veliko orodij že implementirano podporo za neposredni zajem podatkov iz Twitterja. To velja tudi za Apache Nifi, ki smo ga izbrali kot primarno integracijsko okolje in bo podrobneje predstavljeno v nadaljevanju naloge.

2.3.2 Facebook

Značilnosti platforme Facebook ni potrebno posebej predstavljati, saj so možnosti za uporabo njenih vsebin brezmejne. Facebook ponuja takorekoč vse tipe vsebin, ki so sicer domena različnih družbenih medijev. Uporabljamo lahko objave v obliki sporočil, slik, video posnetkov, blogov, t.i. podcastov, kvizov, večigralskih iger, prenosov v živo, izobraževalnih vsebin, časopisov, cenikov, promocijskega materiala, vodičev in podobno. Če k temu dodamo največje število mesečno aktivnih uporabnikov, ki v tem hipu presega dve milijardi [3], dobimo neprecenljivo zbirko podatkov, katerih scenarijev uporabe ne bomo posebej navajali, ker bi s tem presegli okvirje te naloge. Namesto tega bomo preverili možnosti izvedbe avtomatiziranega zajema podatkov.

Prevzemu vsebin iz okolja Facebook je namenjen spletni aplikacijski programski vmesnik poimenovan „Graph API“. Arhitektura vmesnika je sestavljena iz osnovnih treh tipov elementov:

- **Vozlišča:** Individualni objekti, kot so uporabnik, skupina, slika, video, dogodek, stran, komentar, povezava, album in podobno.
- **Povezave:** Relacija med posameznim objektom in skupino objektov, na primer skupina uporabnikov, komentarji objave, uporabnikove slike in podobno.
- **Polja:** Podrobne informacije o posameznem objektu. Uporabnik ima na primer ime, starost ter rojstni datum. Stran ima opis, ime, kategorijo in podobno.

Vsak tip elementa ima dodeljen tudi unikatni identifikator, s katerim lahko poizvedujemo po specifičnem objektu [17].

Žetoni

Za dostop do vsebin je potrebno imeti ustrezno veljaven žeton (angl. „access token“), ki za avtorizacijo prav tako izrablja protokol OAuth 2.0. Prijavi in avtorizaciji v sistem so namenjeni štirje tipi žetonov. Uporabniški žeton se uporabi vsakič, ko klient izvede bralni ali pisalni dostop do izbranih vsebin v imenu uporabnika. So najbolj pogosta oblika žetonov, pridobimo jih ob prijavi s potrditvijo uporabnika.

Aplikacijski žeton služi dostopu do vsebin v imenu aplikacije namesto v imenu uporabnika. Za pridobitev aplikacijskega žetona je potrebno posredovati geslo oziroma „skrivnost“, kar predstavlja določeno ranljivost. Po priporočilu Facebooka geslo ne sme biti zakodirano nikjer v binarni kodi rešitve, uporaba tovrstnih žetonov pa je priporočljiva samo v primeru neposrednega klica na strežnik. Namizne aplikacije te vrste žetonov ne morejo uporabljati, ker so obravnavane kot rizične.

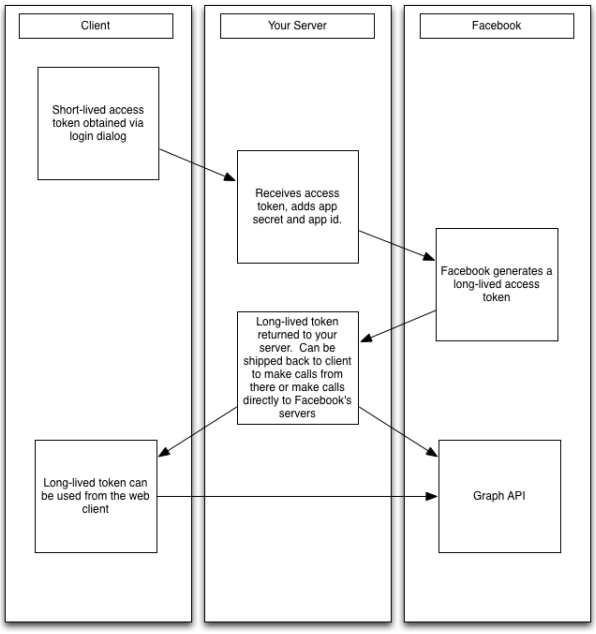
Žetoni za dostop do strani so podobni klasičnim uporabniškim žetonom, le da dovoljujejo dostop do vsebin, vezanih na posamezno stran namesto na posameznega uporabnika. Zadnjo opcijo predstavljajo žetoni klienta. Gre za identifikator, ki se ga vkodira v binarno kodo mobilne ali namizne aplikacije. Imajo omejen dostop do podatkov in se uporabljajo redko [11].

Proces prijave v sistem

Proces same prijave v Facebook je precej zapleten in vključuje veliko preusmeritev na različne naslove, potrjevanj in izmenjav žetonov. V ta namen je implementiran vmesnik za prijavo, poleg katerega Facebook ponuja bogat nabor paketov za razvoj programske opreme (angl. „SDK“), ki olajšajo delo z njim. V okviru diplomske naloge smo sicer uporabili RestFB, ki je

paket knjižnic imeplementiran v Javi in izdan pod licencno uporabe MIT. S knjižnico smo uspešno prevzeli nekatere podatke izbranih objav in z ukazom „feed“ izvedli iskanje objav pod določenimi pogoji.

Življenjska doba žetona je v večini primerov dve uri. To velja za tako imenovani „short-lived“ žeton, ki ga pridobimo ob prijavi v Facebook. Če želimo izvajati klice na API v daljšem obdobju, moramo takšne žetone zamenjati s tako imenovanim „long-lived“ žetonom. Dolgoročni žetoni trajajo 60 dni in omogočajo nemoten dostop do API-ja, mimo Facebookovega avtomatiziranega sistema za detekcijo vsiljivih aplikacij (angl. „spam“). Facebook priporoča dva načina prevzema dolgoročnega žetona, za nas pa je pravzaprav primeren katerikoli od obeh. Slika 2.3 prikazuje možnost neposredne avtentikacije z našega strežnika na Graph API.



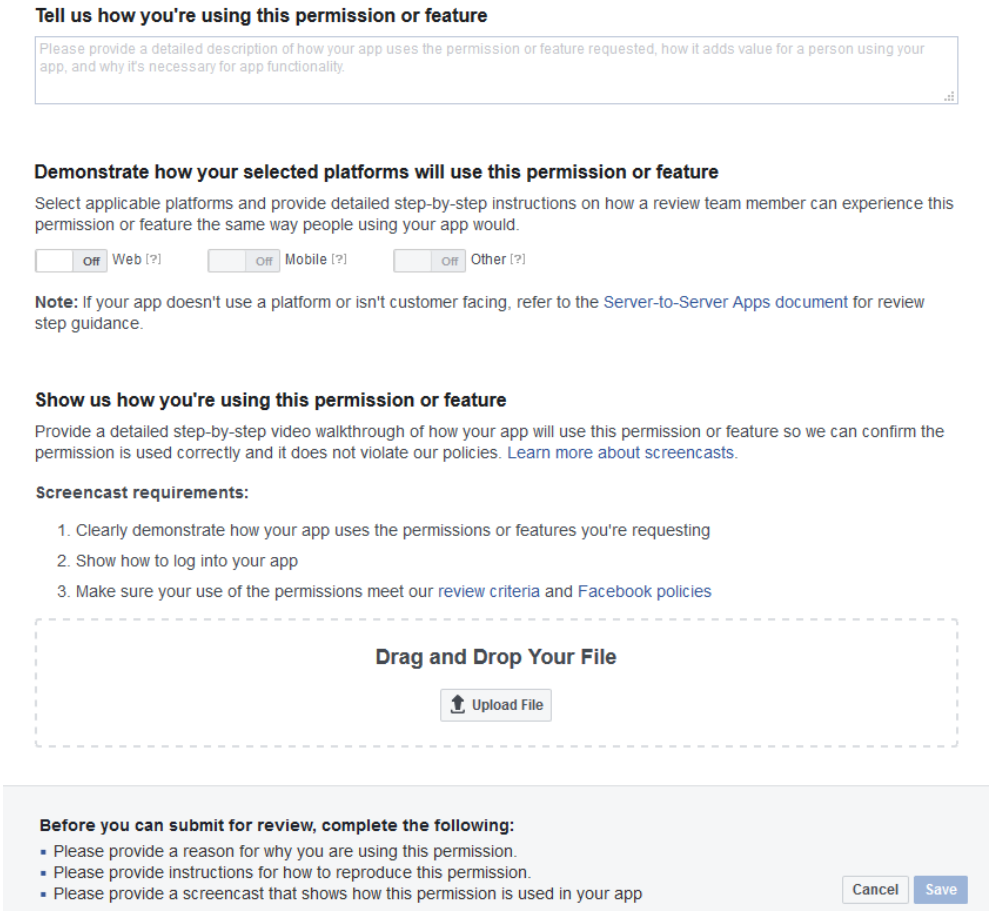
Slika 2.3: Primer postopka zamenjave kratkoročnega žetona z dolgoročnim v okolju Facebook. [16]

Postopek registracije

V bližnji preteklosti se je Facebook moral soočiti s številnimi škandali in očitki o netransparentnem ravnanju z osebnimi podatki. Najbolj odmeven se je zgodil v začetku leta 2018, ko je britansko politično analitično podjetje Cambridge Analitica z izkoriščanjem ranljivosti v Facebooku uporabilo podatke nekaj deset milijonov uporabnikov v politične namene brez njihove privolitve. [4]

Tudi omenjeni dogodek je botroval hitremu zaostrovanju pogojev uporabe družbenega medija. V želji, da bi povečali nadzor nad aplikacijami, ki dostopajo do podatkov, so spremenili proces registracije aplikacij in posledično tudi postopek pridobivanja trajnega žetona. V postopku registracije nove aplikacije je potrebno najprej navesti podatke, kot so URL povezava, na kateri je možno pregledati pogoje uporabe lastne aplikacije ter izjavo o varovanju zasebnosti. V naslednjem koraku se odločimo, do katerih področij platforme Facebook bomo zaprosili za dostop. Teh je kar osemindvajset in obsegajo tako dostop do javnih strani kot recimo modul za objavljanje video prenosov v živo. Za vsakega izmed modulov moramo nato navesti podrobno obrazložitev, zakaj in kako jo želimo uporabljati ter priložiti tudi video gradivo, ki dokazuje željeni namen uporabe. Primer ene izmed vnosnih mask pri postopku registracije nove aplikacije je viden na sliki 2.4.

Na koncu oddamo še poverilnice za testnega uporabnika, s katerim lahko nadzorniki platforme Facebook preverijo ustreznost rešitve in potrdijo, da aplikacija ne zlorabja pogojev uporabe ter omogoča prijavo preko predvidenega vmesnika. Opisani postopek velja predvsem za aplikacije, ki imajo uporabniški vmesnik in so na voljo širšemu krogu uporabnikov. V našem primeru, kjer komunikacija poteka naposredno iz strežnika na strežnik, je postopek verifikacije podoben, le da ni potrebno navajati testnih uporabnikov, saj nadzorniki ne bi mogli dostopati do našega zaprtega sistema. Kljub temu je potrebno priložiti video objavo, ki jasno nakazuje, na kakšen način se



Slika 2.4: Primer dokumentiranja uporabe posamezne funkcionalnosti pri postopku registracije nove aplikacije v okolju Facebook.

uporabljajo prevzeti podatki. Pri tem je potrebno še poudariti, da v našem primeru nismo našli možnost izbire platforme tipa „Server-to-Server“ ampak samo tip platforme kot spletno stran, kar seveda ni povsem enako.

Ker je takšen postopek validacije dolgotrajen in pretirano kompleksen, zagotovitev vsega potrebnega za izpolnjevanje pogojev pa bi nas oddaljilo od prvotnega namena naloge, smo se odločili, da bomo dali prednost drugemu družbenemu mediju.

2.3.3 Tumblr

Tumblr je še ena mikro blogerska platforma, ki ima v primerjavi s Twitterjem nekaj zanimivih specifik. Prva je ta, da dolžina objave ni tako striktno omejena in ponuja možnost vnosa 4.096 znakov. Druge lastnosti so možnost urejanja vsebin v formatu HTML, označevanje objav (angl. „tagging“) in možnost priprave zaporedja objav, ki se na portalu potem pojavljajo zaporedoma, na vsakih nekaj ur ali dni. Platforma trenutno gosti več kot 450 milijonov objav, kar jo po priljubljenosti uvršča v sam vrh družbenih medijev [9].

Trenutno veljavna verzija API-ja je poimenovana Tumblr API v2. Ta v ničemer posebno ne odstopa od preostalih medijev. Edina večja razlika je uporaba avtentikacijskega protokola OAuth 1.0a, namesto verzije 2.0. Ta je za razliko od naslednika nekoliko manj fleksibilen, temelji na kriptografiji in digitalnem podpisovanju, zato je v splošnem celo bolj varen od verzije 2.0. Posledica sta visoka stopnja kompleksnosti in manj načinov uporabe, zaradi česar je verrzija 2.0 veliko bolj popularna [1].

Višje kompleksnosti protokola pri avtentikaciji pravzaprav niti nismo opazili zaradi dobre podpore knjižnic za delo s Tumblrjem. Uporabili smo javansko knjižnico „jumblr“, ki se je izkazala kot enostavna za uporabo in omogoča delo brez posebnih zapletov.

Kljub temu se za implementacijo zajema podatkov iz okolja Tumblr nismo odločili predvsem iz dveh razlogov. Prvi je precej skromna možnost iskanja po objavah, zlasti v primerjavi s preostalimi API-ji, ki smo jih imeli možnost preizkusiti. Na voljo je metoda „tagged“, ki omogoča iskanje objav po določeni oznaki z opcijo omejevanja po datumu ter navedbo dodatnega filtra. To je bolj ali manj vse. Drugi razlog pa je, da je kljub velikemu številu objav pravzaprav težko najti nek kanal, kjer bi se periodično pojavljalo večje število zapisov, kar ni ugodno z vidika testiranja delovanja rešitve.

2.3.4 LinkedIn

Platforma LinkedIn je poslovni medij, ki se osredotoča na profesionalno mreženje uporabnikov. Z več kot 590 milijonov registriranih uporabnikov [2] predstavlja pomemben člen na zemljevidu družbenih medijev.

Z vidika razvoja smo mediju LinkedIn namenili najmanj časa. V primerjavi z ostalimi mediji je API nekoliko slabše dokumentiran, s poudarkom na načinu prijave. Ta sicer temelji na protokolu OAuth 2.0, vendar LinkedIn ne ponuja nobenih lastnih knjižnic, zato je potrebno celotno izmenjavo žetonov sprogramirati ročno. Pri tem je v veliko pomoč javanska knjižnica „scribe“, ki je namenjena splošni prijavi na katerikoli servis, ki uporablja protokol OAuth. V praksi se je izkazalo, da je prijavni postopek v glavnem namenjen aplikacijam in niti ne toliko neposredni povezavi na strežnik, v primerjavi z ostalimi omrežji pa ni posebno praktičen. Ker primarni fokus te naloge ni na sami aventikaciji, smo se odločili, da priložnost raje damo drugemu mediju.

Trenutno veljavna verzija API-ja je sicer v2, dostop do podatkov pa je razdeljen po domenah: ljudje, organizacije, deljenje objav, komunikacija, oglasi, skladnost s predpisi, taksonomije, službe, skupine. Pomanjkanje knjižnic niti ni ovira za dostop do vsebin, saj lahko izvajamo klasične zahtevke tipa GET po protokolu HTTPS/TLS, pri čemer v odgovoru prejmemo vsebino v tekstovnem formatu JSON. Zaradi tega je sicer nekoliko več dela v primerjavi z ostalimi API-ji, ker moramo natančno sestaviti URL parametre za vsako posamezno poizvedbo. Vključeni morajo bii tako podatki, vezani na paginacijo, kot tudi seznami vgnezdenih polj, filtri in podobno.

2.3.5 Google+

Google+ je platforma z zanimivo zgodovino. Po nastanku, ki sega v leto 2011, je hitro pridobivala veliko število uporabnikov, ki pa je nikoli niso sprejeli za svojo. Kljub velikemu številu registriranih uporabnikov ti namreč

v povprečju zelo malo časa namenijo uporabi družbenega medija. V preteklih letih je priljubljenost še dodatno upadla, pomagale niso niti varnostne težave, pri čemer je nedavno odkrita napaka na API-ju omogočila potencialno zlorabo osebnih podatkov več kot 50 milijonov uporabnikov [13]. Ravno v času nastanka tega dokumenta je Google javno najavil, da v aprilu 2019 ukinjajo storitev.

Tehnično gledano na API sicer nimamo nobenih pripomb. Omogoča enostavno pridobitev žetonov ter prijavo s protokolom OAuth 2.0, na voljo so tudi dobre knjižnice, ki jih je enostavno uporabljati. Žal pa so tudi naši testi potrdili, da se na platformi odvija premalo aktivnosti, da bi bila zanimiva za nadaljno analizo.

2.3.6 YouTube

Za razliko od Google+ se je YouTube izkazal za njegovo pravo nasprotje glede dejavnosti uporabnikov. Platforma ponuja veliko različnih vsebin, na nekaterih kanalih se nove vsebine pojavljajo na vsakih nekaj minut, veliko je tudi komentarjev na posamezno objavo.

Ker si API deli večino pozitivnih lastnosti z Google+, smo se odločili, da platformo YouTube izkoristimo za študijo primera celovitega zajema podatkov. Čeprav je platforma primarno namenjena objavi video posnetkov, to pravzaprav ne predstavlja posebne težave, ker ima vsaka video objava še vedno veliko spremnih podatkov, ki so podani v tekstovnem formatu in jih je posledično možno obravnavati na enak način kot vsebino drugih družbenih medijev.

Poglavje 3

Izbira orodij

Odločitev o izbiri primernih orodij za izvedbo zajemov podatkov je bila pogojena z naslednjimi lastnostmi:

- Porazdeljena, skalabilna platforma, ki omogoča paralelno procesiranje velikega števila zajemov in ima opcijo enostavnega dodajanja resursov.
- Možnost avtomatiziranega prenosa podatkov med različnimi sistemi.
- Podpora procesu transformacije, čiščenja ali plemenitenja podatkov.
- Tekstovna podatkovna baza s podporo tehniki iskanja „full-text search“. Ker bo rešitev prevzemala veliko količino besedila v ne nujno strukturirani obliki, želimo imeti čim bolj fleksibilno možnost poizvedovanja po podatkovni zbirki.
- Analitično orodje za predstavitev ter vizualizacijo rezultatov poizvedb.
- Odprtokodna rešitev z možnostjo razvoja v enem bolj razširjenih jezikov, s čimer želimo povečati nabor potencialnih družbenih medijev.
- Enostavna uporaba ter konfiguracija končne rešitve.

Kot dobra izbira orodij, ki takorekoč v celoti pokriva željene zahteve, se je izkazala kombinacija platforme za procesiranje in distribucijo podatkov

Apache Nifi, tekstovna NoSQL podatkovna baza Apache Solr ter orodje za vizualizacijo vsebin poimenovano Kibana Banana.

3.1 Apache Nifi

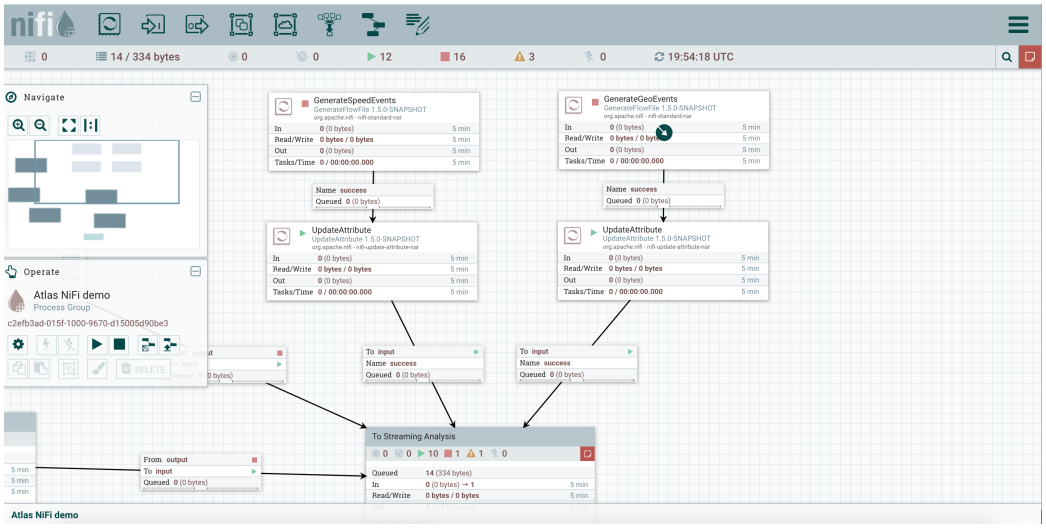
V okoljih „big-data“ se poleg osnovnega problema hranjenja velike količine podatkov srečujemo s problemom premikanja podatkov med različnimi sistemi. Apache nifi je javanska odprtokodna rešitev, ki omogoča povezovanje podatkovnih baz, datotečnih sistemov, gruč okolja Hadoop, naprav, sporočilnih vrst in veliko več. Pri tem lahko različne podatkovne tokove urejamo na eni sami nadzorni plošči, ki je pregledna ter enostavna za uporabo.

Ena izmed glavnih prednosti okolja Nifi je, da omogča strokovnjakom iz posameznih poslovnih področij oblikovanje procesov in kolaboracijo brez potrebe po poglabljanju v kodo rešitve. Dizajn procesov na nadzorni plošči je predstavljen s posameznimi koraki, katerih sosledje določajo usmerjene povezave med njimi. Rešitev tako pogosto spominja na proces, ki je bil izrisan na tablo. Samo okolje Nifi še posebej izstopa v primerih, ko so procesi osnovani na inkrementalnih medsebojno neodvisnih korakih [22]. Primer nadzorne plošče Nifi je viden na sliki 3.1.

3.1.1 Osnovni koncepti

Nifi je distribuiran procesno orientiran sistem, ki je bil zasnovan z namenom upravljanja tokov podatkov. Vsak proces v Nifiju se prične z nekim dogodkom ali podatkom, ki nato potuje skozi več stopenj transformacij v željeno končno obliko. Nifi je sestavljen iz glavnih treh komponent:

- **Tokovne datoteke:** Originalno so poimenovane kot „flowfiles“, obsegajo pa vsebino in množico atributov. Atribute lahko v procesu dodajamo in transformiramo z izraznim jezikom Nifi.
- **Procesorji:** Neodvisne osnovne logične enote, ki vsebujejo svojo kodo



Slika 3.1: Primer nadzorne plošče z definiranim procesom v okolju Apache Nifi [26].

in imajo definiran vhod ali izhod, lahko tudi oboje. Procesorji so tisti, ki generirajo tokovne datoteke oziroma z njimi nekaj počnejo.

- **Povezave:** Določajo, kako tokovne datoteke potujejo med posameznimi procesorji. Ločujejo pravilno obdelane zapise od napačnih, omogočajo ponavljanje izvedbe logike in podobno.

Procese na nadzorni plošči urejamo preprosto z operacijo tipa „povleci in spusti“. Vsak procesor prikazuje svoj števec datotek na vhodu ter izhodu, zato lahko tok podatkov spremljamo v realnem času in hitro ugotovimo, če se kje pojavijo ozka grla. Za potrebe razhroščevanja so na voljo statusna vrstica, okno s povzetkom izvajanja ter meni z zgodovino statusov [22].

Fleksibilnost Nifija nam omogoča prevzem podatkov na različne načine. Najbolj pogosti primeri so periodično izpraševanje (angl. „polling“) spletnih API-jev po protokolu „restful“ ter prevzem vsebine v formatu JSON ali prevzem podatkov iz datotečnih sistemov in podatkovnih baz. Poleg tega je možna tudi vzpostavitev procesa, ki posluša na izbranih vratih, pri čemer

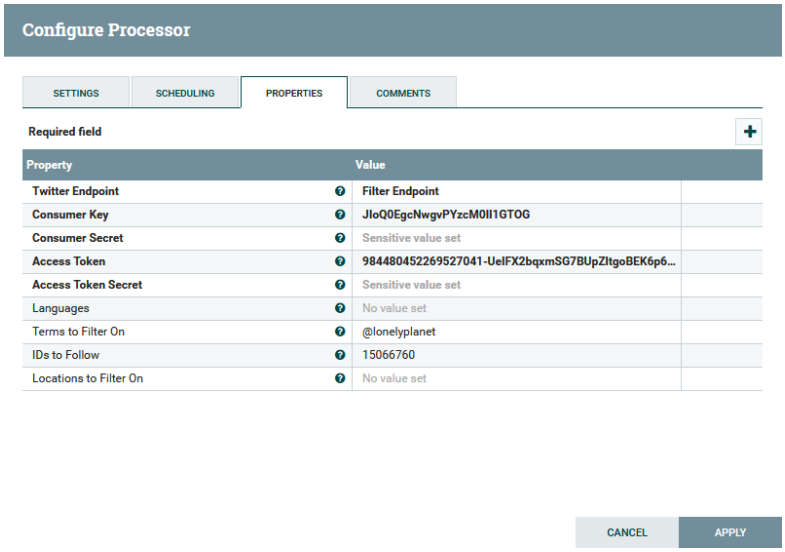
zunanji sistemi oddajajo podatke na Nifi. Možni so torej tako periodični zajemi, ki jih krmilimo s časovnikom ali vmesnikom „cron“, kot tudi prevzem podatkov v realnem času. V tej nalogi nas zanimajo predvsem procesi iz prve skupine.

3.1.2 Procesorji

Trenutno je ob namestitvi na voljo 286 procesorjev, ki služijo različnim namenom in jih lahko razdelimo v naslednje kategorije: transformacija podatkov, usmerjanje, dostop do podatkovnih baz, ekstrakcija atributov, interakcija z operacijskim sistemom, sprejem ter oddaja podatkov, agregacija, HTTP komponente in spletni servisi Amazon. Ker gre za odprtokodno platformo, se s časom pojavljajo vedno novi procesorji, prototip novega procesorja pa je bil izdelan tudi v okviru te naloge.

Vmesnik za konfiguracijo in spremljanje izvajanja procesorja je vselej enak, razlikujejo se le izhodi, vhodi in množica atributov ter nastavitev. Že nekaj let je na voljo tudi procesor, ki omogoča zajem podatkov iz družbenega medija Twitter. Za uporabo procesorja je priporočljivo pridobiti trajni žeton, kar lahko enostavno storimo z registracijo nove aplikacije v uporabniškem profilu Twitter. Postopek pridobivanja žetona je vsaj zaenkrat enostaven, potrebno je le potrditi strinjanje s pogoji uporabe. Ko imamo pripravljene vse potrebne ključe, jih lahko vnesemo v nastavitvenem meniju procesorja, tako kot to prikazuje slika 3.2.

Procesor omogoča iskanje objav po različnih iskalnih geslih ter sledenje kanalom. Delovanje je zanesljivo, čeprav ne omogoča posebno naprednih nastavitev, ki bi omogočale celovit zajem podatkov za nazaj ali podrobnejšo izbiro vsebin, ki bi jih želeli spremljati. To so nekatere izmed lastnosti, ki jih želimo z našim procesorjem izboljšati.



Slika 3.2: Konfiguracija procesorja Nifi za sprejem podatkov iz družbenega medija Twitter. [19]

3.2 Apache Solr

Apache Solr je hitra in skalabilna platforma za shranjevanje ter iskanje podatkov v besedilih in je bazirana na knjižnici Apache Lucene [28]. Lucene je odprtokodna rešitev, namenjena visoko performančnem iskanju po besedilih. Knjižnica ima močno spletno skupnost, kar je omogočilo njen razvoj in priljubljenost, zato je danes to najbolj uveljavljena iskalna rešitev. Pomembno je poudariti, da je Lucene zares samo knjižnica in nima niti lastnih konfiguracijskih datotek. [29].

Poleg prednosti same knjižnice Lucene, z uporabo okolja Solr pridobimo še nekatere pomembne funkcionalnosti, kot so na primer:

- Strežniško okolje, ki komunicira v več formatih, vključno z XML ter JSON.
- Uporaba nastavitvenih datotek zlasti za podatkovno shemo indeksa, ki definira polja ter konfiguracijo tekstovnega iskanja.

- Nekaj tipov predpomnenja, namenjenih hitrejšemu iskanju
- Nadzorna plošča, ki jo lahko urejamo v spletnem vmesniku.
- Izvedba operacije „faceting“ nad rezultatom iskanja.
- Široka paleta razčlenjevalnikov poizvedb, kot je na primer vmesnik „eDisMax“, ki je bolj priročen od razčlenjevalnika poizvedb knjižnice Lucene.
- Delovanje v gruči, ki jo koordinira Zookeeper.
- Vmesnik za uvoz podatkov ter razhroščevalnik.

3.2.1 Osnovni koncepti

Osnovna podatkovna enota v okolju Solr oziroma Lucene je dokument, ki si lasti nek unikatni identifikator. S pomočjo identifikatorja lahko nad dokumentom izvajamo poljubno operacijo CRUD. V primerjavi z nekaterimi drugimi dokumentno orientiranimi sistemi, kot je na primer MongoDB, Lucene ne predvideva enostavne uporabe vgnezdenih struktur znotraj dokumenta. Ta mora biti posledično enodimenzionalna, s čimer pridobimo strukturo, ki je podobna tabelam v relacijskih podatkovnih bazah. Možna je sicer uporaba atributov tipa „multi-values“, ki dovoljujejo hranjenje seznama vrednosti, vendar to ni povsem enako. Še ena pomembna lastnost v primerjavi z relacijskimi podatkovnimi bazami je omejena zmožnost povezovalnih poizvedb, ker bi ta preveč vplivala na skalabilnost platforme. Relacijske baze so prav tako bolj učinkovite pri zagotavljanju konsistence transakcij po načelu ACID. Seveda pa v zameno za nekatere omejitve pridobimo številne prednosti, med katerimi je v ospredju veliko večja fleksibilnost pri iskanju podatkov v velikih podatkovnih zbirkah. Poizvedovanje po vsebinah je bolj podobno naravnemu jeziku in je v nekaterih primerih možno, tudi če ne poznamo natančne arhitekture podatkovnega modela. [29].

3.2.2 Podatkovna shema

Definicija podatkovne sheme je eden izmed osnovnih korakov pri izgradnji učinkovitega iskalnega sistema. Gre za procesa definicije strukture podatkov in analize besedila, s katerima spremljamo uspešnost delovanja. Pred samo definicijo sheme je dobro razumeti problem, ki ga rešujemo, in predvideti tiste vsebine, po katerih bomo želeli poizvedovati [28].

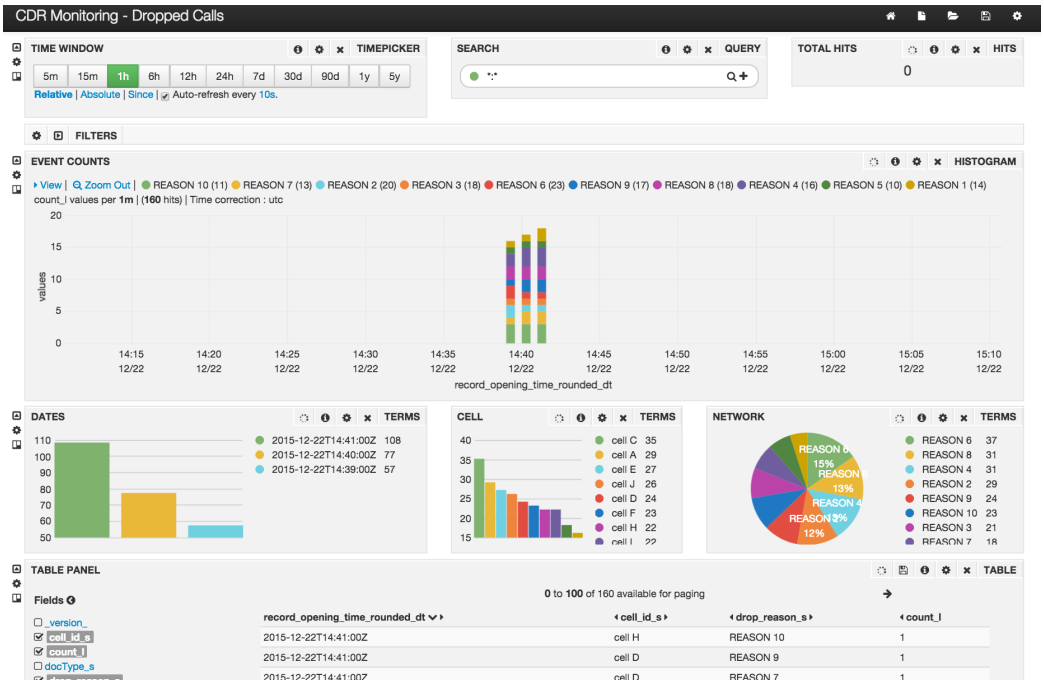
Osnovna enota, po kateri poizvedujemo v okolju Solr, je dokument in je ekvivalent vrstici oziroma posameznemu zapisu v relacijskih podatkovnih bazah. V datoteki „schema.xml“ definiramo strukturo dokumenta in njegovih polj, pri čimer imamo na voljo veliko možnosti konfiguracije. Definicije posameznih polj Solr nato uporablja kot osnovo pri dodajanju novih vsebin in pri poizvedovanju po podatkovni zbirki. Dodatne možnosti konfiguracije sistema so na voljo v datoteki „solrconfig.xml“.

3.3 Kibana Banana

Banana je vizualizacijsko orodje, ki je prilagojeno platformi Solr in temelji na Kibani. Kibana je odprto kodna rešitev, ki omogoča vizualizacijo strukturiranih ali nestrukturiranih podatkov, do katerih dostopamo z indeksi tipa „elasticsearch“. Rešitev je napisana v jezikih HTML ter JavaScript in je namenjena iskanju ter prikazu podatkov tako v standardni obliki, namenjeni za poslovno inteligenco, kot tudi spremljanju podatkov v realnem času. Ponuja številne načine vizualizacije in nam pomaga olajšati razumevanje velikih količin podatkov. Do poročil lahko dostopamo s poljubnim spletnim brskalnikom, posamezne dele poročila pa lahko vkomponiramo tudi v druge sisteme. Omogočeno je deljenje posnetkov stanja pri rezultatih iskanja z drugimi uporabniki [21].

Večino naštetegs omogoča tudi Banana, prilagojena pa je posebej za delo z okoljem Solr. Posledično jo namestimo kar znotraj inštalacije za Solr, v

direktorij *../server/solr-webapp/webapp*. Po namestitvi je Banana privzeto dostopna na istem URL naslovu kot Solr, le da na koncu navedemo še pot do naslovne strani *../solr/banana/src/index.html*. S tem je že omogočeno poizvedovanje po podatkovnih zbirkah sistema Solr. Primer prikaza vizualizacije vidimo na sliki 3.3.



Slika 3.3: Primer poročila v vizualizacijskem orodju Banana.

Nadzorno ploščo v Banani poljubno prilagajamo željam in potrebam z dodajanjem ali odvzemanjem posameznih elementov. Posameznim komponentam določimo tudi željeno frekvenco osveževanja podatkov. V veliko pomoč pri prikazovanju podatkov je tehnika poimenovana „faceting“, ki jo ponuja okolje Solr. Pod tem pojmom razumemo operacijo razporejanja iskalnih rezultatov v skupine na osnovi indeksiranih pojmov. Operacija je koristna pri postopku analize in omogoča hitro kataloško prikazovanje vsebin pod različnimi pogoji.

Poglavje 4

Razvoj procesorjev v okolju Nifi

Za pripravo procesorjev Nifi v osnovi potrebujemo dve komponenti, to sta Java ter Apache Maven. Apache Maven je javansko okolje, ki omogoča avtomatizirano prevajanje kode in upravljanje projektov ter enostavno souporabo knjižnic in vtičnikov. Maven olajša proces grajenja rešitve pri čemer uporabnika vzpodbuja k uporabi dobrih praks. Metapodatki projekta so zabeleženi v konfiguracijski datoteki „pom.xml“ oziroma v projektnem objektnem modelu. Definirana je enotna direktorijska struktura, v samem projektu pa je predviden prostor za specifikacijo, izvorno kodo ter izvedbo testnih scenarijev (angl. „unit testing“).

Pri razvoju smo si pomagali še z nekaterimi razvojnimi orodji, predvsem z orodjem IntelliJ IDEA, ki olajša razvoj javanskih rešitev in ima dobro podporo tudi za Maven. Za preizkus delovanja spletnih servisov sta bila uporabljena cURL ter Postman. Enak komplet razvojnih orodij je bil sicer uporabljen tudi pri implementaciji rešitev na ostalih družbenih medijih.

Pred pričetkom implementacije procesorja je potrebno najprej zagotoviti poverilnice za dostop do vsebin platforme YouTube. To lahko storimo v nad-

zorni plošči „Google APIs“, kjer najprej ustvarimo nov projekt, v nadzorni plošči zavihka „APIs & Services“ pa izberemo in omogočimo opcijo „YouTube Data API v3“. Nato na zavihku „Credentials“ dodamo nov ključ tipa „OAuth 2.0 client ID“. Ko imamo na izbranem projektu generiran ključ, nas klik nanj preusmeri na masko za prevzem poverilnic.
S klikom na prenos v JSON formatu dobimo datoteko z avtorizacijskimi podatki, s katerimi lahko dostopamo do vmesnika API. Pri tem sta ključna podatka „client_id“ ter „client_secret“, primer vsebine datoteke pa je viden na sliki 4.1. Pot do datoteke s poverilnicami je tudi ena izmed ključnih nastavitev pri konfiguraciji procesorja.

Slika 4.1: Primer datoteke JSON z avtorizacijskimi podatki.

V okviru diplomske naloge so bili razviti trije procesorji za namen demonstracije celovitega prevzema nekaterih vsebin iz družbenega medija YouTube:

- Procesor za iskanje novih objav in prevzem njihove vsebine.
- Procesor za osveževanje objav podatkovne zbirke, pri čemer posamezne objave posodabljamo v celoti ali delno.
- Procesor za zajem podatkov komentarjev objav, ki jih imamo v podatkovni zbirki.

Opomba: Potrebno je poudariti, da pri vseh oblikah prenosov prenašamo izključno tekstovne vsebine. Samih video posnetkov ali slik ne prenašamo v nobenem primeru.

Predvideni način uporabe rešitve je tak, da eden ali več procesorjev iščejo nove objave video objav skladno z nastavljenimi iskalnimi pogoji. Ti procesorji na izhod pošiljajo celovito vsebino objav v tekstovni obliki, ki je lahko zanimiva za nadaljno analizo, hkrati pa v interni zbirki vodijo evidenco prevzetih objav. Naslednja dva procesorja nato za vsako objavo iz podatkovne zbirke periodično osvežujeta vsebine, ki se sčasoma spreminjajo. Primer so uporabniški komentarji na posamezno objavo, ki nastajajo sproti ali statistike ogledov video objav, ki se spreminjajo z vsakim klikom na posamezno objavo.

Na tak način je zagotovljeno, da vsebine posamezne objave ne prenesemo samo prvič, ampak redno izvajamo proces posodabljanja vsebin, ki se lahko dogajajo še dolgo po datumu prve objave posnetka, včasih tudi po več let.

Vsi trije procesorji ponujajo možnost natančne konfiguracije frekvence ter intenzitete klicev, s čimer lahko enostavno porezdelimo obremenitev API-ja skozi čas in na tak način sledimo dobrim praksam za implementacijo tovrstnih rešitev. Posameznim procesorjem lahko dodelimo tudi omejitev kvote porabe in na tak način poskrbimo, da imajo določeni zajemi prednost pred drugimi, ker se ne more zgoditi, da bi manj pomembni procesori porabili kvoto, ki je rezervirana za pomembnejše zajeme. Logika za izvajanje zajemov je sicer zbrana v skupni kodi, ki je ločena od samih procesorjev, zato je samo delovanje neodvisno od okolja Nifi in ga je z minimalno modifikacijami možno preseliti na drugo okolje. Koncept zajema, ki je demonstiran na družbenem mediju YouTube, pa je dovolj generičen, da ga je z nekaj prilagoditvami možno implementirati tudi na drugih družbenih medijih.

4.1 Procesor za iskanje ter prevzem podatkov video objav

Iskanju video objav na aplikacijskem programskem vmesniku YouTube je namenjena funkcija „**youtube.search.list**“. Ta kot vhod sprejme kopico parametrov, s katerimi natančno določimo iskalne pogoje, vrača pa odgovor v JSON formatu, ki ima lahko poleg primarnih opisnih podatkov video objave še nekatere dodatne vsebine.

Osnovni problem, ki se pojavi pri zajemu podatkov družbenih medijev, je kako poiskati vse objave, ki pripadajo določenemu kanalu ali iskalnemu pojmu. Aplikacijski programski vmesniki ponavadi vrnejo prvih nekaj rezultatov, ki ustrezajo iskalnemu pogoju, število vseh razpoložljivih objav pa je lahko včasih tudi sedem mestno. Za sprehajanje po celotni zbirki zadetkov večina API-jev tako pozna koncept žetonov za dostop do naslednjih strani. Poizvedovanje po podatkih vrne prvo stran z nekaj zadetki in žeton, s katerim lahko pošljemo novo poizvedbo za naslednjo stran. Na tak način se lahko sprehajamo po celotnem naboru objav, dokler ne prevzamemo vseh.

Posledica takšnega načina zajema je, da želimo zapise pred prevzemom urediti po določenem vrstnem redu. Za najboljšo opcijo se pri tem izkaže časovna značka oziroma čas nastanka objave, po kateri uredimo zapise v padajočem vrstnem redu. Če bi zapise uredili recimo po pomembnosti zadetka, imamo lahko težavo pri celovitosti zajema. Recimo da bi prevzeli prve tri strani najpomembnejših zadetkov in bi se vmes pojavila nova objava, ki bi jo algoritem po pomembnosti razvrstil na drugo stran. V tem primeru bi nam takšna objava izpadla iz zajema, ker je ne bi obdelali ne v trenutni, niti v prihodnji sekvenci poizvedb.

Način zajema z žetoni je na voljo tudi na vmesniku YouTube, v splošnem pa ima svoje prednosti in slabosti. Če je malo objav za prevzem, je tak

način zelo učinkovit in zanesljiv, saj lahko s peščico zaporednih klicev hitro prevzamemo celotno vsebino. Težave se lahko pojavijo, kadar prenašamo veliko število zadetkov, ali kadar želimo prevzemati podatke novih objav z visoko frekvenco osveževanja. V primeru YouTube bi to recimo pomenilo, da z vsako poizvedbo vedno pridobimo 50 zadnjih objav, četudi se od naše zadnje poizvedbe ni pojavila nobena nova. Število 50 je namreč maksimalno število zadetkov, ki ga vrača API klic za iskanje objav na YouTube. Količina petdesetih objav v tem primeru sicer ne vpliva na ceno poizvedbe, je pa nepraktična tako z vidika obremenitve API-ja kot tudi ustvarjanja nepotrebnega prometa po mreži in našem okolju. Iz omenjenih razlogov se je zato v praksi bolje izkazalo prevzemanje podatkov z uporabo datumskih parametrov in ne z žetoni.

Z uporabo datumskih parametrov namreč lahko pri vsaki poizvedbi natančno zamejimo začetno ter končno časovno točko zajema. Na tak način se nam nikoli ne zgodi, da bi posamezno video objavo nenamenoma prevzeli dvakrat. Če se med časom zadnje poizvedbe in trenutnim časom ni pojavila nobena nova objava, bo klic na API s tema časovnima točkama vrnil prazen rezultat, kar pomeni, da ne ustvarjamo odvečnega prometa in se niti ni potrebno spreševati, kateri del vsebine v odgovoru je že bil prenešen. Logika zajemov se obnese tudi pri prevzemu starejših objav, ker se nikoli ne more zgoditi, da bi se pojavila nova video objava, ki bi imela čas objave postavljen v obdobje, ki smo ga že obdelali.

YouTube API omogoča takšen način delovanja. Parametra, ki ju uporabimo, se imenujeta „publishedAfter“ ter „publihsedBefore“. Implementacija rešitve je zasnovana na način, da se poleg omenjenih parametrov uporablja še en dodatni, s katerim označujemo širino časovnega okna.

4.1.1 Osnovni principi delovanja

S prevzemom zadetkov v enem zaporedju klicev torej nimamo posebnih težav. Ker so družbeni mediji dober generator velike količine podatkov, pa se lahko zgodi, da prevzem celotne vsebine z eno samo iteracijo ni smiseln. Za prevzem vseh objav, ki ustrezajo posameznim iskalnim pogojem, bi lahko potrebovali nekaj tisoč klicev, pri čemer vsak klic lahko traja nekaj sekund. Tudi če ponudnik družbenega medija to dovoljuje, lahko takšen zajem traja nekaj ur. Z vzporednim zajemom podatkov za nekaj različnih kanalov bi hitro lahko ustvarili nesorazmerno veliko količino prometa.

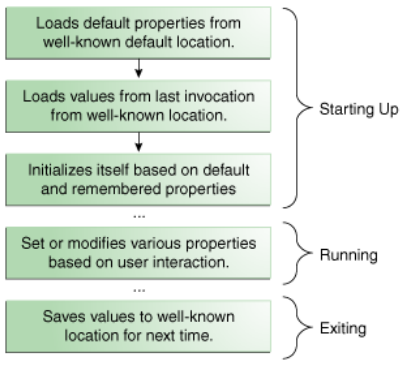
Večina ponudnikov družbenih medijev poskuša z raznimi omejitvami preprečiti takšen način delovanja, pri čemer so jasno defnirane kvote, ki določajo, koliko klicev je dovoljenih v določenem časovnem oknu. Primer je recimo Twitter, ki omejitve kvot definira na osnovi petnajstminutnih intervalov [7]. Tudi če teh omejitev nimamo, dobre prakse narekujejo, da je zajeme smiselno enakomerno razporediti na daljšem časovnem obdobju, pri čemer klice izvajamo periodično in z omejeno intenziteto. Porazdeljevanje klicev ter minimizacijo večkratnega prevzemanja enake vsebine so smernice, ki jih priporoča tudi Facebook [8].

Prednosti takšne zasnove so še druge. Omogoča učinkovito paralelizacijo večih vzporednih zajemov in lažje prilagajanje izhodnim sistemom. Sistemi, ki sprejemajo izhodne podatke procesorjev, lahko izvajajo zahtevne sprotne operacije čiščenja ter transformacije podatkov ali procese osveževanja iskalnih indeksov, zato včasih ne dohajajo količine podatkov, ki se pojavlja na vhodu. Poleg tega je lažje obvladovati morebitne izpade delovanja, enostavneje je tudi napovedati trajanje posameznih ciklov, kar predstavlja nezanemarljivo prednost.

Skladno z zapisanimi ugotovitvami je rešitev implementirana na način, ki omogoča enostavno konfiguracijo porazdelitve zajemov, pri čemer za posa-

mezen časovni interval določimo maksimalno količino prevzetih objav z enim klicem ter dovoljeno število klicev, ki jih lahko izvedemo v enem časovnem intervalu. Za enakomerno proženje zajemov skrbi Nifi procesor sam z vgrajenim razporejevalnikom (angl. „scheduler“) procesorskih klicev.

Ker narava rešitve predvideva, da se posamezna serija zajemov ne zaključi v enem časovnem intervalu, je potrebno zagotoviti, da je stanje obdelav persistentno in se prenaša iz enega intervala v drugega. Poleg tega moramo persistenco zagotoviti v primeru izklopa ali izpada procesorja, zato da ta zna nadaljevati z delom od zadnje uspešno obdelane točke naprej. V ta namen smo si pomagali z javanskim razredom „Java.Properties“, ki omogoča hranjenje parov ključev in vrednosti ter persistenco stanja [6]. Izkazal se je kot dobra izbira in pomemben element pri implementaciji delovanja procesorjev. Primer uporabe omenjenga razreda je viden na sliki 4.2.



Slika 4.2: Tipičen primer uporabe razreda Java Properties med izvajanjem neke aplikacije.

Nifi sicer ponuja tudi vmesnik StateManager, ki je enostaven za uporabo, vendar ima določene omejitve. Ena takšnih je, da pri serializaciji dovoljuje maksimalno velikost datoteke 1MB. Ker so lahko nekateri seznami veliki, na primer seznam prevzetih video objav, se ta omejitev izkaže za pretirano. Da

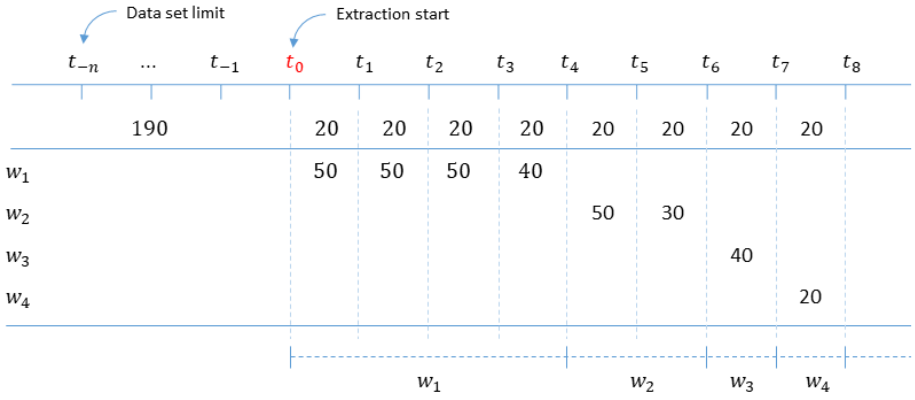
bi se izognili razdeljevanju seznama v več ločenih objektov, smo zato raje uporabili razred Java Properties.

4.1.2 Algoritem za prevzem novih objav

Za izvedbo celovitega zajema podatkov glede na iskalni pojem ali identifikator kanala torej izvajamo periodične zajeme od časovne točke vklopa procesorja proti časovnim točkam starejših objav. Pri tem je potrebno nasloviti še nekatere pomembne lastnosti procesa. Prva je ta, da pri iskanju mogoče vseeno ne želimo prevzeti celotne zgodovine objav, ampak nas zanimajo podatki recimo le za zadnje leto. To uredimo enostavno, z dodatnim parametrom, ki ga nastavimo na konfiguraciji procesorja in s katerim zamejimo časovno točko podatkov za nazaj. Procesor bo tako ob vklopu prevzemal objave v smeri od mlajših k starejšim, ko pride do konca, pa bo z zajemom zaključil in začel ponovno od trenutne časovne točke.

Iz opisanega delovanja vidimo, da se je med prevzemanjem starejših objav pojavilo novo časovno okno, ki sega od trenutka vklopa procesorja do trenutka, ko je procesor prevzel vse starejše objave. V tem časovnem oknu so se pojavile nove objave, zato mora procesor prevzeti tudi te. Po zaključeni drugi iteraciji prevzemov so se vmes ponovno nabrale nove objave, ki jih je potrebno obdelati. Razvidno je torej, da se zajemi vedno odvijajo v serijah, ki pripadajo določenemu časovnemu oknu. Če je količina prevzema podatkov v enem ciklu dovolj visoka, so ta okna z vsako naslednjo serijo manjša, dokler ne pridemo do stanja, kjer je časovno okno manjše od enega cikla, kar pomeni, da lahko en cikel proženja procesorja prevzame vse nove objave. Opisano logiko delovanja ponazarja slika 4.3.

Na ilustrativnem primeru predpostavljamo, da je vklopljeno delovanje procesorja, ki v povprečju vrača 20 novih objav na periodo. Omejitev zajemov za nazaj smo postavili na časovno točko, ki v danem časovnem intervalu glede na vklop procesorja obsega 190 zapisov. Te zapise torej moramo prevzeti



Slika 4.3: Primer 1: logika delovanja procesorja pri nemotenem zajemu podatkov.

za nazaj, predno lahko nadaljujemo s sprotnimi zajemi. Procesor je nastavljen tako, da je maksimalno dovoljeno število prevzetih zapisov v eni periodi enako 50.

Sosledje dogodkov v primeru 1 (slika 4.3) je sledeče:

1. V časovni točki t_0 vklopimo delovanje procesorja. S tem se prične prvo časovno okno w_1 v katerem želimo prevzeti 190 zadnjih objav. Točka t_{-n} predstavlja omejitev „From date“, ki smo jo določili v konfiguracji procesorja in določa starost podatkov, ki jih še želimo prevzeti.

2. Ker je omejitev števila prejetih zapisov 50, v prvih treh periodah t_0 - t_3 prevzamemo trikrat po 50 zapisov, torej maksimum, kot ga določa konfiguracija procesorja.

3. V časovnem ciklu t_4 prevzamemo še zadnjih 40 zapisov, ki jih je ostalo od prvotnih 190. S tem se zaključi obdelava prvega časovnega okna w_1.

4. Medtem ko smo v obdobju t_0 - t_4 prevzemali objave za nazaj, se je pojavilo 80 novih zapisov (v vsakem ciklu povprečno 20). Odpre se novo časovno okno w_2, v katerm prevzemamo nove objave

5. V periodah t_5 in t_6 prevzamemo še preostalih 80 objav. Ker ta iteracija obsega dva cikla, se je vmes pojavilo že 40 novih. Odpremo novo okno w_3, v katerem bomo prevzeli nove objave.

6. Število preostalih zapisov je sedaj že dovolj majhno, da jih lahko prevzamemo v enem ciklu, zato velja, da je širina časovnega okna w_3 enaka dolžini časovnega cikla t_7.

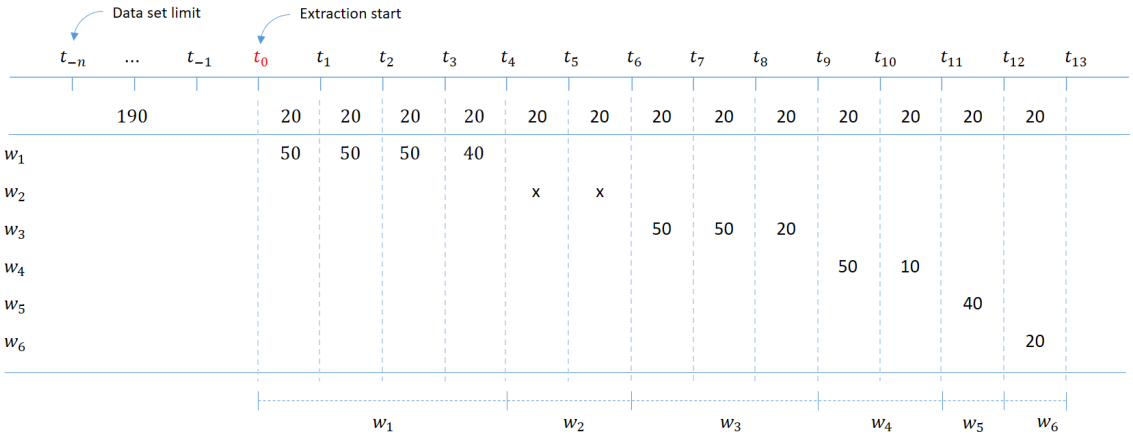
7. Od te točke naprej so časovna okna vedno dovolj kratka, da lahko z enim proženjem procesorja prevzamemo vse objave, ki se pojavljajo sproti.

Opisana logika delovanja se izkaže za zanesljivo in fleksibilno, ker garantira, da neglede na frekvenco ter intenziteto klicev v določenem časovnem obdobju vedno opravimo celovit zajem, pri čemer na izgubimo nobene objave. Pri tem sta tako dolžina časovnih ciklov kot tudi maksimalna obremenitev API-ja stvar konfiguracije nekaj parametrov procesorja, s čimer se lahko enostavno prilagajamo potrebam in omejitvam, ki jih predpisuje API. Edini pogoj za pravilno delovanje rešitve je, da izberemo takšno konfiguracijo, ki zagotavlja, da v enem časovnem ciklu lahko prevzamemo nekoliko več zapisov, kot je povprečno število novih zapisov na cikel.

Dodatna prednost je, da celovitost zajemov nikoli ni pogojena z dolžino obdobja, v katerem procesor ne deluje oziroma je izklopljen. Procesor si vedno zapomni, do katere točke je prišel in ob naslednjem proženju nadaljuje od zadnje uspešne izvedbe. Nedelovanje procesorja v primerih izpadov, napak ali izvedba rednih vzdrževalnih postopkov samo nekoliko podaljšajo obdobje, v katerem bodo prevzete vse objave, ki ustrezajo izbranemu iskalnemu pojmu. Poenostavljeni primer izpada delovanja prikazuje slika 4.4.

Začetni pogoji zajema in konfiguracija procesorja sta enaka kot pri prvem primeru. Sosledje dogodkov v ilustrativnem primeru 2 (slika 4.4) je sledeče:

1. V časovni točki t_0 vklopimo delovanje procesorja. S tem se prične prvo



Slika 4.4: Primer 2: logika delovanja procesorja v primeru začasnega izpada delovanja.

časovno okno w_1 v katerem želimo prevzeti 190 zadnjih objav. Točka t_{-n} predstavlja omejitev „From date“, ki smo jo določili v konfiguracji procesorja in določa starost podatkov, ki jih še želimo prevzeti.

2. Ker je omejitev prejetih zapisov 50, v prvih treh periodah t_0 - t_3 prevzamemo trikrat po 50 zapisov, torej maksimum, kot ga določa konfiguracija procesorja.

3. V časovnem ciklu t_4 prevzamemo še zadnjih 40 zapisov, ki jih je ostalo od prvotnih 190. S tem se zaključi obdelava prvega časovnega okna w_1.

4. Za razliko od prvega primera se nato zgodi izpad delovanja procesorja, ki traja dva časovna cikla, torej od t_5 do t_6. Dolžina izpada je določena s časovnim oknom w_2.

5. V periodi t_7 se ponovno vzpostavi delovanje. Število novih zapisov, ki so se pojavili, medtem ko smo prevzemali prvih 190 objav in je trajal izpad, je sedaj 120. Prične se torej novo časovno okno w_3, v katerem prevzemamo preostalih 120 objav (povprečno 20 novih na cikel).

6. V periodah t_7 - t_9 prevzamemo vseh 120 objav časovnega okna w_3. Ker je dolžina tega okna tri cikle, se je vmes pojavilo še 60 novih objav. Prične se okno w_4, v katerem nadaljujemo s prevzemi.

7. Dva cikla t_{10} - t_{11} sta v tem primeru dovolj, da prevzamemo objave okna w_4. Vmes se je pojavilo 40 novih, ki jih prevzemamo v oknu w_5.

8. Število novih zapisov je že dovolj nizko, da v enem časovnem ciklu prevzamemo vse nove zapise. V oknu w_6 enostavno prvzamemo vseh 20 novih zapisov.

9. Od te točke naprej so časovna okna vedno dovolj kratka, da lahko z enim proženjem procesorja prevzamemo vse objave, ki se pojavijo sproti.

Oba prikazana primera sta ilustrativne narave in prikazujeta primer, kjer je kapaciteta procesorja nastavljena relativno blizu povprečnemu številu novih objav. V praksi je to razhajanje ponavadi večje, okna se hitro ožajo, zlasti po tem, ko smo uspešno prevzeli vse starejše objave in se vračamo k prevzemu sprotnih zapisov.

Implementacija rešitve zagotavlja celovit zajem, ki uporabnikom omogoča, da se ne ukvarjajo z ročnim poganjanjem zajemov na manjkajočih ali izpadih časovnih intervalih in da se jim ni potrebno spraševati, ali smo pri zajemu izpustili pomembno objavo.

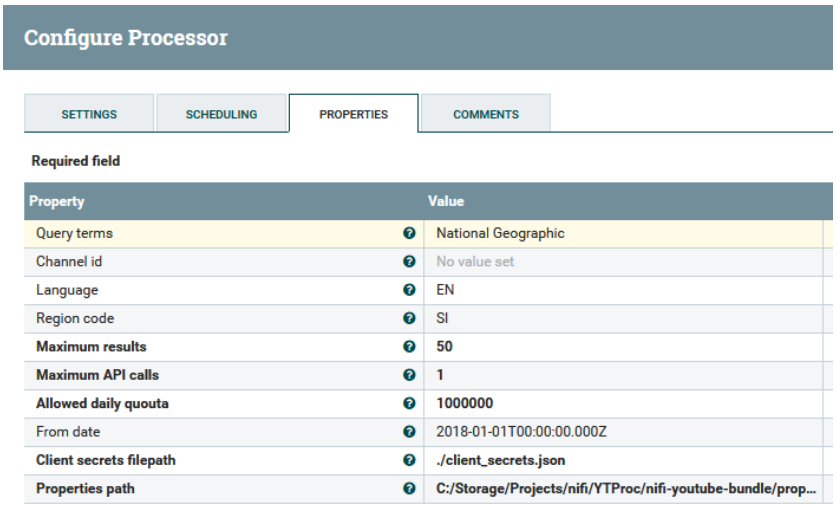
4.1.3 Konfiguracija procesorja

Pri konfiguraciji procesorjev Nifi je pomembno določiti predvsem nastavitve na zavihkih Scheduling ter Properties. Na prvem določimo urnik proženja procesorja, na drugem pa urejamo parametre, ki so specifični vsakemu procesorju. Pri konfiguraciji urnika Nifi ponuja dva načina:

- **Timer driven:** Periodično proženje v časovnih ciklih oziroma intervalih. Določimo lahko poljuben cikel proženja na urnem, minutnem ali sekundnem nivoju.

- **Cron driven:** Fleksibilna konfiguracija urnika na enak način kot pri razporejevalniku na okolju Linux.

Zaradi periodične narave zajema podatkov iz API-jev, je v našem primeru primernejša prva možnost. Ker pa v določenih primer potrebujemo bolj natančno določitev urnika, s katerim razmejimo posamezne serije oziroma iteracije, uporabljamo tudi konfiguracijo tipa Cron. Možni parametri za konfiguracijo procesorja za iskanje ter prevzem podatkov video objav je viden na sliki 4.5.



Slika 4.5: Konfiguracija procesorja za iskanje ter prevzem podatkov video objav.

V nadaljevanju je pojasnjen pomen posameznih parametrov:

- **Query terms:** Iskalni pojem, po katerem želimo iskati nove objave.
- **Channel id:** Identifikator kanala, za katerega želimo prevzemati objave. Deluje v kombinaciji z iskalnim pojmom.
- **Language:** Dvoznakovna šifra jezika po standardu ISO639-1, s katerim želimo zamejiti zadetke iskanja.

- **Region code:** Dvoznakovna šifra države po standardu ISO3166-1, s katero se želimo zamejiti na zadetke, ki so dostopni v izbrani državi.
- **Maximum results:** Največje število zadetkov, ki jih želimo prejeti z enim klicem na API.
- **Maximum API calls:** Največje število dovoljenih klicev na API, ki se izvedejo v enem časovnem ciklu, torej z enim proženjem izvedbe procesorja.
- **Allowed daily quota:** Dovoljena poraba dnevne kvote klicev na API za izbrani procesor.
- **From date:** Časovna točka, od katere želimo prevzemati podatke. Zapisana je v univerzalnem časovnem formatu UTC. Določa širino prvega časovnega okna ob vklopu procesorja.
- **Client secrets file path:** Pot do datoteke z avtentikacijskimi podatki, s katerimi se procesor lahko prijavi na API.
- **Properties directory:** Direktorij za odlaganje objektov tipa Java Properties za zagotavljanje persistentnega delovanja procesorja.

S kombinacijo nastavitev urnika izvajanja procesorja ter parametrov „Maximum results“ in „Maximum API calls“ lahko enostavno določimo razmerje med ažurnostjo podatkov in stopnjo obremenitve API-ja.

Primer: nastavitev proženja na vsakih 5 minut in vrednosti paramerov „Maximum results“ na 50 ter „Maximum API calls“ na 10 pomeni, da vsakih 5 minut na API pošljemo od ene do desetih poizvedb, pri čemer vsaka vrne 50 iskalnih zadetkov. Za vsako iskalno poizvedbo sledi še ena dodatna, s katero prevzamemo podrobnosti posameznih video objav, kar skupaj pomeni maksimalno število dvajsetih klicev, s katerimi prevzamemo podatke za 500 video objav. S takšno konfiguracijo procesorja torej lahko vsak dan teoretično prevzamemo od 0 do 144.000 novih objav.

4.1.4 Spremljanje porabe kvote

Vsak aplikacijski programski vmesnik predvideva določeno omejitev uporabe, ki je izražena s kvotami. Kvote definirajo dovoljeno uporabo API-ja v določenem časovnem intervalu, na primer uri, minuti ali dnevu, pri čemer ima vsak klic na API svojo ceno. Cene posameznih klicev se lahko med seboj občutno razlikujejo tudi po nekaj tisočkrat, glede na zahtevnost same operacije. Najnižjo ceno imajo ponavadi bralni dostopi, višjo pa pisalni. Cene in omejitve kvot se s časoma lahko spreminjajo.

Enaki principi veljajo tudi za aplikacijski programski vmesnik YouTube, ki pozna tri tipe operacij. To so bralna, ki je najcenejša, dražja pisalna in operacija nalaganja video objave, ki je najdražja. Trenutno so razmerja v ceni med omenjenimi operacijami 1 proti 50 proti 1600. Z vidika zajema podatkov nas zanimajo samo operacije prvega tipa. Ker te spadajo v najcenejšo kategorijo, lahko Nifi procesorji opravijo zajem velikih količin podatkov.

Za boljši nadzor nad porabo kvote je v Nifi procesorju implementirana logika za spremljanje porabe kvote. Z vsakim klicem na API se namreč zabeleži cena posameznega klica po veljavnem ceniku. Če z izvedbo klicev presežemo dovoljeno dnevno kvoto, se klici ne izvajajo več, procesor pa zabeleži informacijo, da je presegel dovoljeno kvoto. Vsak procesor posebej vodi tudi lastno evidenco porabljenih klicev, zato je možno kvote prerazporejati med različnimi procesorji. Na tak način lahko zagotovimo, da bo prevzem pomembnejših vsebin vedno nemoten, prevzem manj pomembnih vsebin pa se lahko zaključit tudi predčasno.

Cena posameznega klica je odvisna od nekaterih parametrov, predvsem od tega, katere dele objave prenašamo (angl. „parts“). Vsak del predstavlja posamezen vsebinski sklop, ki ga lahko prenašamo ali ne. Primer je prenos vsebine video objave, ki ima trenutno na voljo 13 različnih delov, katerih skupna cena znaša 20 enot. Sam klic stane 1 enoto, k čemur prištejemo ceno

posameznega sklopa objave, ki ga prenašamo. Za osveževanje podatkov statistik na primer porabimo dve enoti, za podatke o formatu video posnetka pa dodatni dve. Če želimo prenesti vse, kar nas zanima, je torej cena klica enaka 5 enot.

Za racionalno izrabo kvot je zato pomembno, da maksimalno vnovčimo vrednost vsakega posameznega klica. Platforma YouTube ima to značilnost, da je cena poizvedbe vedno določena glede na sestavo klica in je neodvisna od števila rezultatov, ki jih lahko vrne posamezna poizvedba. Primer je klic metode „youtube.search.list“, ki stane 100 enot ne glede na to, ali smo kot maksimalno število vrnjenih zadetkov nastavili vrednost 1 ali 50. Drugi primer je klic metode „youtube.videos.list“, ki z izbranimi vsebinami „snippet“, „statistics“, „contentDetails“ ter „topicDetails“ stane 9 enot ne glede na to, ali smo se v poizvedbi omejili na eno ali 50 različnih video objav.

Skladno z zapisanimi ugotovitvami je Nifi procesor implementiran na način, da se posamezne objave vedno obdeluje v paketih, skladno z nastvitvijo parmetra „Maximum results“. Če je ta nastavljen na najvišjo dovoljeno vrednost, pri vsakem klicu vedno obdelamo 50 objav, s čimer optimiziramo število klicev na API ter porabo dnevne kvote. Nižanje vrednosti omenjenega parametra ima lahko svoje prednosti, vendar na samo porabo dnevne kvote ne vpliva.
Nastavitev parametra „Maximum API calls“ po drugi strani vpliva na porabo dnevne kvote, saj določa število dovoljenih klicev v posamezni periodi.

4.2 Procesor za osveževanje podatkov video objav

Za objave na družbenih medijih je značilno, da se del podatkov osvežuje še dolgo po tem, ko je bila prvič objavljena na družbenem mediju. Primer so statistični podatki o številu ogledov ali všečkov objave, ki se pravzaprav po-

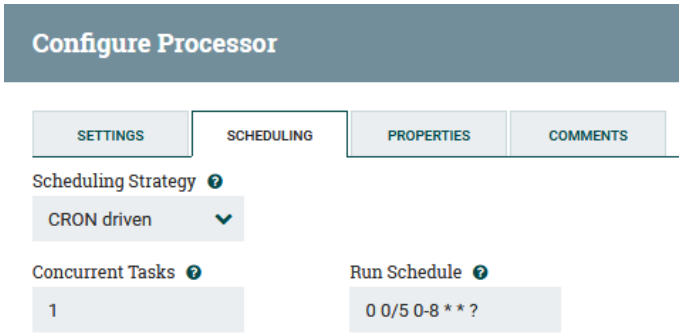
sodabljajo neprestano z vsakim obiskom strani. Naslednji tip spremenljive vsebine predstavljajo komentarji h posamezni objavi, ki lahko nastajajo še leta po tem, ko se je zapis pojavil na družbenem mediju. Nenazadnje se lahko spremeni tudi vsebina same objave ali komentarja, saj imajo uporabniki možnost urejanja vsebine za nazaj.

API-ji imajo redko zagotovljeno dobro podporo za ugotavljanje sprememb na vsebini, ki je enkrat že bila prenešena. Ponavadi sta v ta namen na voljo datum zadnje spremembe ali značka ETag, ki je del protokola HTTP in je namenjena detekciji sprememb na izbrani vsebini. Slednja je na voljo tudi na Google API-ju in v nekaterih primerih omogoča minimizacijo količine prenosa podatkov. Če pri poizvedbi navedemo značko ETag in vsebina ni bila spremenjena, bo API namesto vsebine vrnil napako tipa 304. Mehanizem je sicer izjemno uporaben pri aplikacijah, ki delajo s posameznimi zapisi, ne obnese pa se najbolje pri delu s seznami. Pri prevzemu daljšega seznama objav z enim samim klicem bi API vrnil napako 304 samo v primeru, da ni bil osvežen noben izmed zapisov, kar je v praksi redko. Podobno je s komentarji, kjer je vsebina razdeljena po posameznih straneh in lahko obsega na tisoče zapisov. Namesto uporabe značk je zato uporabniku procesorjev Nifi ponujena možnost izbire dela vsebine, ki jo želi osveževati.

4.2.1 Konfiguracija procesorja

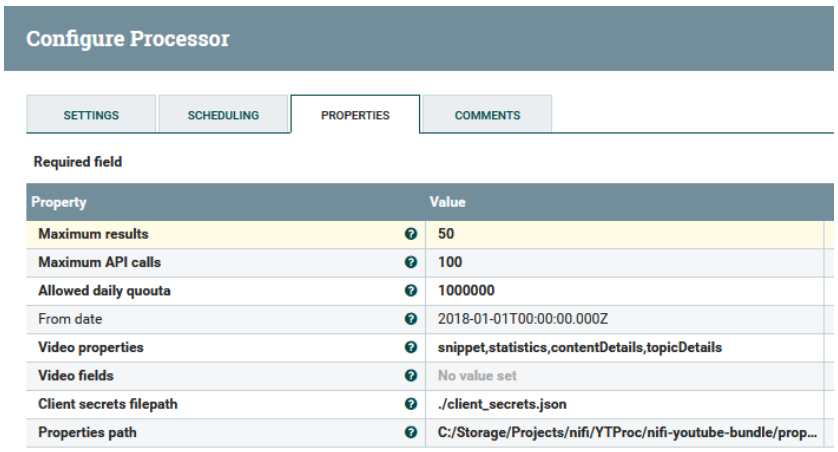
S časom lahko podatkovne zbirke postanejo velike, zato je potrebno upoštevati, da je tudi iteracija osveževanja podatkov potratna. Posledično procesa osveževanja podatkov ponavadi ne želimo izvajati ves čas, ampak je dovolj, če to izvedemo občasno. V ta namen nam pride prav fleksibilnost nastavitev, ki jo omogoča vmesnik Cron. Na sliki 4.6 vidimo primer konfiguracije urnika procesorja, ki cikel osveževanja izvaja na vsakih 5 minut samo med polnočjo in osmo uro zjutraj. Po tej uri se zajemi za izbrani dan končajo. Če bi želeli izvesti še en

cikel zajemov, bi lahko dodali nov procesor z drugače določenim urnikom.



Slika 4.6: Urnik proženja procesorja za osveževanje podatkov video objav.

Primer konfiguracije procesorja za osveževanje podatkov video objav je viden na sliki 4.7.



Slika 4.7: Konfiguracija procesorja za osveževanje podatkov video objav.

Večina možnih nastavitev je enaka tistim, ki jih ima procesor za iskanje ter inicialni zajem podatkov. Novi sta naslednji konfiguraciji:

- **Video properties:** Seznam delov video objave, ki jih želimo periodično osveževati. Cena posameznega dela je ponavadi dve enoti.

- **Video fields:** Seznam polj, ki jih želimo periodično osveževati. Število navedenih polj nima vpliva na ceno poizvedbe.

Parametra omogočata, da dodatno zamejimo vsebino, ki se osvežuje periodično. Z dodatnim filtrom poskrbimo, da je obremenitev sitemov čim manjša, hkrati pa nam zagotavlja, da osvežujemo samo tisti del vsebine, ki nas resnično zanima. Parameter „Maximum API calls“ je v tem primeru nastavljen nekoliko višje, ker želimo v vsaki periodi obdelati večjo količino podatkov kot pri sprotnih zajemih.

4.2.2 Način delovanja procesorja

Ko procesor zazna, da se je pričela nova iteracija osveževanja vsebine objav, v prvi fazi najprej izvede sortiranje podatkovne zbirke prenešenih objav od mlajše proti starejšim. Nato se izvede faza priprave množic identifikatorjev objav, ki jih bomo prevzeli z vsakim klicem na API. Velikost posamezne množice je odvisna od konfiguracije parametra „Maximum results“ in optimalno obsega 50 objav.

Ko imamo pripravljene množice posameznih objav, se te v obliki seznamov zapišejo v ustrezno datoteko tipa Java Properties. Nato po vrsti jemljemo enega za drugim in izvajamo klice na API, dokler ne zmanjka zapisov ali pa smo dosegli dovoljeno dnevno kvoto. Za prevzem vsebine se izvaja klic na metodo „**youtube.videos.list**“. Po uspešnem prevzemu vseh seznamov video objav se datoteka Java Properties pobriše, procesor pa je pripravljen na nov cikel osveževanja vsebine.

Primer: Če podatkovna zbirka obsega 200.000 video objav, bo procesor pripravil 4.000 množic velikosti 50 in periodično izvajal klice, 100 v vsakem ciklu. Glede na primer konfiguracije procesorja, ki je razviden iz slik 4.6 ter 4.7, bi procesor takšno podatkovno zbirko osvežil v štiridesetih ciklih in za to porabil tri ure in dvajset minut. Cena celotnega prevzema bi v tem primeru bila 3.600 enot.

V praksi se je izkazalo, da nas najbolj zanimajo podatki statistik, ki se skozi čas tudi največ spreminjajo. Ostali metapodatki video objave se spreminjajo le, če avtor objave izvede spremembo vsebine objave. V primeru, da bi osveževali samo podatke video objave, bi bila cena prevzema v tem primeru 1.200 enot. S povečanjem intenzitete klicev iz 100 na 1000 pa bi zajem opravili v dvajsetih minutah. Skladno z zapisanim bi lahko uporabili tudi drugačno konfiguracijo procesorja, na primer takšno, kjer statistike osvežujemo vsako uro, ostalo vsebino pa enkrat na teden ali pa sploh ne.

Če procesor pri osveževanju ugotovi, da je bila posamezna objava umaknjena, jo v svoji podatkovni zbirki označi kot neaktivno. Takšne objave v prihodnje ignorira in jih ne poskuša več osveževati.

4.3 Procesor za zajem podatkov komentarjev

Pri zajemu podatkov komentarjev se srečujemo s podobnimi težavami kot pri osveževanju vsebine video objav, le da imajo prenosi v tem primeru še občutno višjo ceno. Za namen prenašanja podatkov komentarjev API ponuja dve metodi:

- **youtube.commentThreads.list:** Prevzem komentarjev za posamezno video objavo ali kanal na krovnem nivoju (angl. „top level“).
- **youtube.comments.list:** Prevzem seznama odgovorov na komentarje (angl. „replies“).

Nobena izmed metod za prevzem žal ne ponuja možnosti iskanja po datumu spremembe ali vsaj po datumu objave komentarja, tako kot je možno poizvedovati pri iskanju video objav. Na voljo je sicer značka ETag, ki označuje spremembo posameznega zapisa, vendar je ta uporabna le, če komentarje prevzemamo posamezno, enega po enega. Ker prenose izvajamo v paketih,

nam značka ETag ne ponuja natančne informacije o tem, na katerem komentarju v množici je prišlo do spremembe. Tudi podatek o časovni znački posodobitve zapisa nam ne koristi, saj po njej ni mogoče poizvedovati.

Posledica tega je, da podatke komentarjev osvežujemo na način, da jih prevzemamo v celoti, pri čemer si pomagamo z žetoni za prevzem zaporednih strani (angl. „pageToken“). Zapise prevzemamo v paketih po 100, pri čemer z vsako poizvedbo dobimo žeton za prevzem novih 100 zapisov na naslednjih strani. Postopek ponavljamo, dokler ne obdelamo vseh strani.

4.3.1 Konfiguracija procesorja

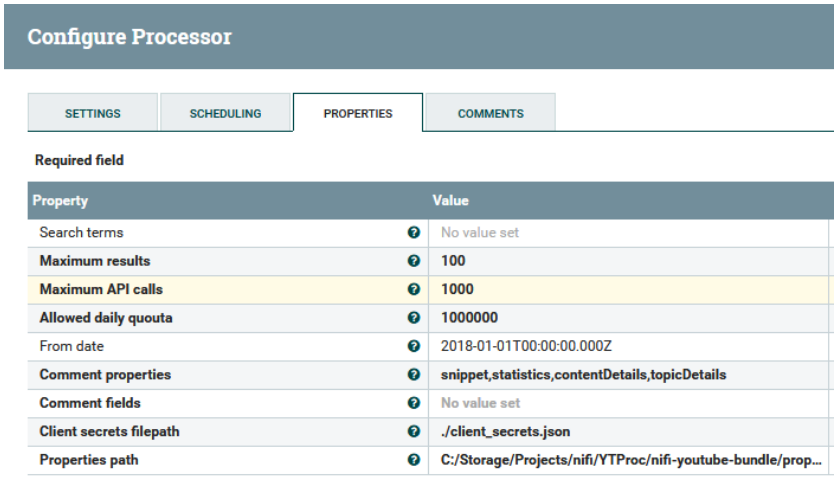
Za konfiguracijo proženja procesorja je podobno kot pri prejšnjem primeru tudi tu uporaben vmesnik Cron. Ker bi z dnevnimi prevzemi celotne vsebine komentarjev hitro trčili ob dovoljeno kvoto, je priporočeni cikel prevzema podatkov v tem primeru na mesečni ravni. Na tak način za izvedbo posamezne iteracije prevzema ne izrabljamo samo dnevne kvote ampak mesečno, kar omogoča vzdrževanje ažurnosti vsebine večje podatkovne zbirke, kot če bi se zamejili na dnevno kvoto.

Primer konfiguracije urnika Cron bi bila v tem primeru:
*0 0/5 * * 1-12 ?*

Eksplicitna nastavitev mesečne periode procesorju pove, da mora osveževalni cikel ponastaviti z vsakim mesecem.

Konfiguracija procesorja za zajem podatkov komentarjev je podobna procesorju za osveževanje podatkov video objav. Ker je v tem primeru količina podatkov še večja, je toliko bolj smiselno podrobneje definirati vsebino parametra „Comment fields“ in se na tak način zamejiti zares samo na tisto vsebino, ki nas zanima. Razlika je tudi v vrednosti parametra „Maximum results“, ki v tem primeru enemu klicu dovoljuje prevzem 100 zapisov name-

sto 50. Prav tako je dodan parameter „Search terms“, ki omogoča dodatno filtriranje po vsebini komentarjev. Primer konfiguracije je viden na sliki 4.8.



Slika 4.8: Konfiguracija procesorja za osveževanje podatkov komentarjev.

Primer delovanja procesorja za izbrano konfiguracijo: Podatkovna zbirka, ki obsega 20.000 video objav s povprečnim številom komentarjev 1.000 na objavo, zahteva 200.000 klicev za prevzem celotne vsebine. Ker je cena posameznega klica 3 enote, to pomeni, da bi za celotni prevzem porabili 600.000 enot dnevne kvote, opravili pa bi ga v šestnajstih urah in štiridesetih minutah.

4.3.2 Način delovanja procesorja

Zaradi velike količine podatkov, se je pri tem procesorju priporočljivo zamejiti samo na tista polja, ki so potencialno lahko zanimiva za analizo podatkov.

Konfiguracija parametra „Comment fields“ je v tem primeru enaka:
„etag, eventId, items(id, kind, replies(comments(snippet(authorDisplayName, likeCount, parentId, publishedAt, textDisplay, updatedAt, videoId, viewerRating))), snippet(topLevelComment(snippet(authorDisplayName, likeCount, pu-

blishedAt, textDisplay, updatedAt, viewerRating)), totalReplyCount, videoId)), nextPageToken, pageInfo“.

Slika 4.9 prikazuje primer posameznega komentarja na krovnem nivoju, ki je bil zamejen z zgoraj definiranim parametrom.


```
-{
   "kind": "youtube#commentThread",
   "id": "UgzwEqu4crx59mfctnd4AaABAg",
  -"snippet": {
    "videoId": "xi6r3hZe5Tg",
   -"topLevelComment": {
    -"snippet": {
       "authorDisplayName": "National Geographic",
       "textDisplay": "Singapore is considered one of the most advanced regions in
the world. What intrigues you the most about this fascinating city?",
       "viewerRating": "none",
       "likeCount": 466,
       "publishedAt": "2018-11-27T01:13:35.000Z",
       "updatedAt": "2018-11-27T01:13:35.000Z"
      }
     },
     "totalReplyCount": 117
   },
```


Slika 4.9: Primer posameznega zapisa komentarja z izbranim naborom stolpcev.

Iz priloženega primera vidimo, da ima zapis še 117 pripadajočih odgovorov (angl. „replies“) na izbrani komentar. Te prevzamemo s klicem na drugo metodo API-ja, ki prav tako vrača maksimalno 100 rezultatov. Za prevzem niti komentarjev iz primera moramo torej izvesti 3 klice, enega z metodo „commentThreads“ in naslednja dva z metodo „comments“.

Komentarji so torej shranjeni v hierarhični obliki, kjer posamezne niti, odgovori ter odgovori na odgovore tvorijo razvejano strukturo s številnimi vozlišči in vgnezdenimi komentarji. Pri poizvedovanju po komentarjih nam sicer ni potrebno izvajati rekurzivnih klicev po posameznih vozliščih, saj s klicem na metodo „comments“ prevzamemo vse odgovore, ne glede na njihovo hierarhijo. Posledično je potrebno poskrbeti le za pravilno dodajanje odgovorov na posamezen krovni komentar.

Sosledje izvajanja zajema je podobno kot pri procesorju za osveževanje podatkov video objav. V prvi fazi najprej izvedemo sortiranje podatkovne zbirke po padajočem vrstnem redu, nato se lotimo obdelave posameznih objav, od mlajših proti starejšim. Pri tem se s poizvedbami premikamo po posameznih straneh, dokler ne prevzamemo vseh zapisov. Na vsakem krovnem komentarju, ki ima pripadajoče odgovore, izvedemo dodatno zaporedje klicev po posameznih straneh, s čimer prevzamemo tudi to vsebino. Pri tem ves čas spremljamo porabo dovoljene dnevne kvote. V primeru, da smo dnevno kvoto že porabili, se zajem na tej točki zaključi, procesor pa si zabeleži identifikator objave ter žeton strani, ki jo je nazadnje uspešno prevzel.

Če z obdelavo zapisov ne zaključimo v trenutnem dnevu, nadaljujemo naslednji dan. Ob koncu meseca se prične nova iteracija zajemov ne glede na to, ali smo s predhodno zaključili. Z omenjenim načinom delovanja dosežemo kompromis med ažuronostjo vsebin, obremenitvijo API-ja ter izrabo dovoljene dnevne kvote. Če je količina zapisov kljub temu prevelika za celovito osveževanje, lahko časovno okno dodatno zamejimo z nastavitvijo parametra „from date“. Na tak način procesorju enostavno povemo, da nas vsebine, starejše od izbranega datuma, ne zanimajo. K temu lahko dodamo še iskalni pojem, s katerim se dodatno zamejimo samo na tiste komentarje, ki imajo relevantno vsebino.

Ker se pri tem procesorju pojavi izrazito veliko število klicev na API, se včasih zgodi naključna napaka. Te napake so sicer redke, zato pri trenutni implementaciji rešitve ne poskušamo ponavljati zajemov, ampak samo zabeležimo število pojavitev napak.

4.4 Arhitektura rešitve

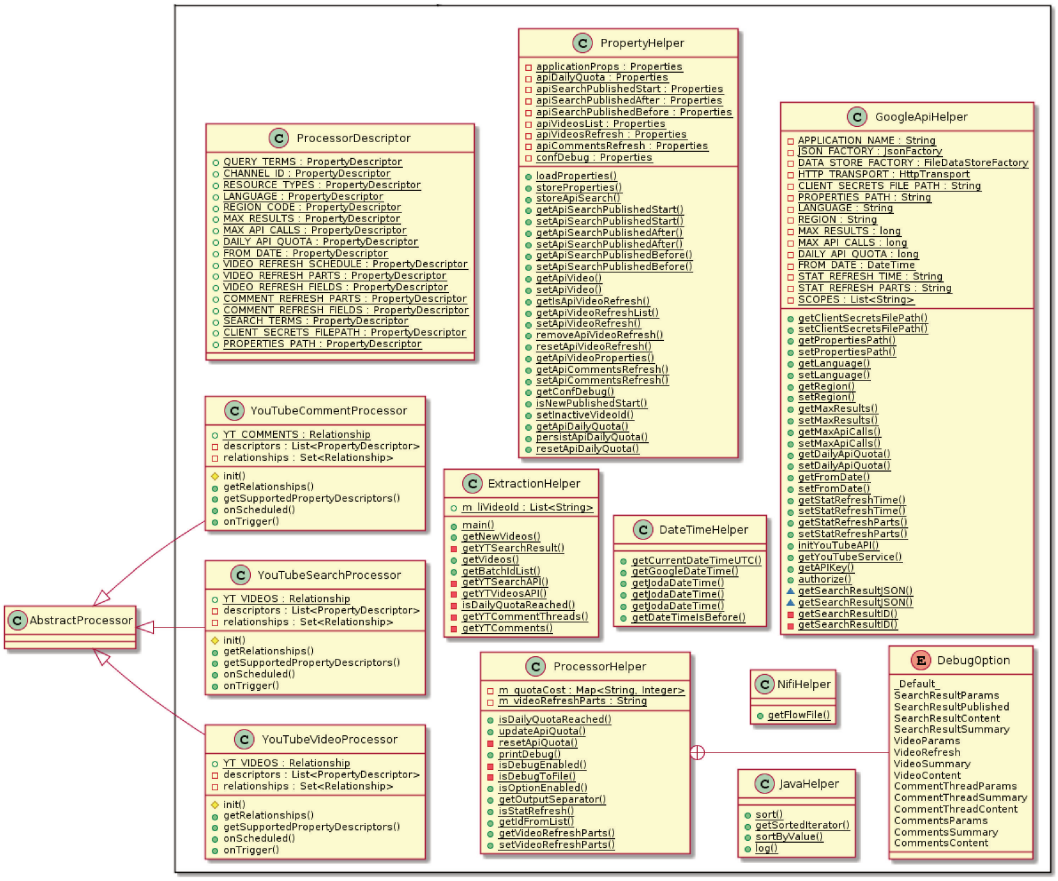
Rešitev je v celoti implementirana v Javi. Za osnovo je bil vzet paket „org.apache.nifi.processor“, ključen pa je razred „AbstractProcessor“, ki smo ga uporabili za vse procesorje. Razred ima definiranih nekaj metod, ki se prožijo z različnimi cikli procesorja. Ključna je metoda „onTrigger“, ki skrbi za izvedbo logike pri vsakem proženju procesorja in generira izhodne tokovne datoteke.

V metodi „init“ poskrbimo za inicializacijo deskriptorjev procesorja, ki določajo njegovo konfiguracijo ter relacijo, ki določa tip izhoda procesorja. Poleg tega je dodanih nekaj pomožnih javanskih razredov, ki skrbijo za pravilno izvedbo logike in so podrobneje predstavljeni na razrednem diagramu (slika 4.10).

4.4.1 Razredni diagram

Poleg dedovanih razredov, vezanih na razred AbstractProcessor, so bili uporabljeni še naslednji pomožni razredi:

- **GoogleApiHelper:** Vsebuje metode za avtentikacijo uporabnika ter inicializacijo aplikacijskega programskega vmesnika YouTube.
- **ExtractionHelper:** Logika izvajanja zajemov za vse tri tipe procesorjev.
- **ProcessorHelper:** Podporne metode za spremljanje porabe dnevne kvote ter razhroščevanje.
- **PropertyHelper:** Razred za zagotavljanje persistence stanj procesorjev s pomočjo uporabe razreda Java Properties.
- **DateTimeHelper:** Ker veliko delamo z datumi, se za nepogrešljivo izkaže knjižnica „joda.time“. Poleg tega opravlja konverzijo med lastnim časovnim formatom in časovnim formatom Googlovih knjižnic.



Slika 4.10: Razredni diagram.

- **NifiHelper:** Pomožni razred, namenjen paketu Nifi. Zankrat služi samo kot pomoč pri generiranju izhodnih tokovnih datotek procesorjev.

- **JavaHelper:** Različne podporne metode, predvsem za urejanje seznamov ter beleženje dnevniških datotek.

4.4.2 Primer implementacije razreda AbstractProcessor v Javi

Slika 4.11 prikazuje konkretno implementacijo procesorja za iskanje ter zajem podatkov video objav. V prvi fazi se najprej naložijo vrednosti vseh konfi-

guracijskih parametrov, ki smo jih odčitali v fazi inicializacije procesorja. S temi parametri se nato kliče metoda „initYouTubeAPI“, ki vzpostavi vmesnik za komunikacijo z API-jem.

Nato se v zanki sprehodimo čez število dovoljenih klicev, kot smo ga nastavili na konfiguracijski maski procesorja s parametrom „Maximum API calls“. V vsakem obhodu izvedemo klic na API z iskalnim geslom ali identifikatorjem kanala. Pri tem je lahko izpolnjen eden od obeh, oba ali nobeden. Uspešen klic na API vrne rezultat, ki vsebuje največ toliko zapisov, kot smo jih nastavili s parametrom „Maximum results“.

Za vsak zapis rezultata (možnih je največ 50) nato pripravimo izhodno tokovno datoteko, ki ji določimo tudi naziv. V tem primeru je naziv datoteke kar identifikator video objave. Takšen način poimenovanja izhodnih datotek zagotavlja generično rešitev, saj že iz naziva datoteke lahko sklepamo o njeni vsebini tudi v primeru, ko datoteke odlagamo na disk.

Obdelava posameznih objav v zanki je smiselna tudi zato, ker jih želimo čimprej in čimbolj enakomerno pošiljati na izhod procesorja. Na koncu izvedemo še preverjanje števila klicev. V primeru, da pri posameznem časovnem ciklu dovoljujemo izvedbo večjega števila klicev na API, to še ne pomeni, da jih v resnici tudi izvedemo. Vedno izvedemo samo prvi klic, ostale pa le v primeru, da z dosedanjimi klici še nismo prevzeli celotne vsebine, ki je na voljo. Če bi jo prevzeli, bi bilo število zapisov v rezultatu manjše od dovoljenega, kar je signal, da lahko zaključimo z izvedbo tekočega cikla.

```java
@Override
public void onTrigger(final ProcessContext context, final ProcessSession session) throws ProcessException {
    //load property descriptors
    String queryTerms = context.getProperty(QUERY_TERMS).evaluateAttributeExpressions().getValue();
    String channelId = context.getProperty(CHANNEL_ID).evaluateAttributeExpressions().getValue();
    String propertiesPath = context.getProperty(PROPERTIES_PATH).evaluateAttributeExpressions().getValue();
    String language = context.getProperty(LANGUAGE).evaluateAttributeExpressions().getValue();
    String regionCode = context.getProperty(REGION_CODE).evaluateAttributeExpressions().getValue();
    ...

    //init Google API
    GoogleApiHelper.initYouTubeAPI(propertiesPath,
            language,
            regionCode,
            ...
            );

    //transfer flow files
    List<Video> liVideo;
    for(int i = 0; i < GoogleApiHelper.getMaxApiCalls(); i++) {
        liVideo = getNewVideos(queryTerms, channelId);

        if(liVideo != null) {
            for (Video v : liVideo) {
                try {
                    session.transfer(getFlowFile(session, v.getId(), v.toPrettyString()), YT_VIDEOS);
                    session.commit();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            if (liVideo.size() < GoogleApiHelper.getMaxResults()) break;
        }
        else break;
    }
}
```

Slika 4.11: Implementacija metode „onTrigger“ na procesorju za iskanje ter prevzem podatkov video objav.

Poglavje 5

Analiza podatkov

S procesorji Nifi imamo torej pripravljeno rešitev za polnjenje podatkovnih zbirk s podatki iz družbenih medijev. Možnosti za nadaljno raziskovanje podatkov je veliko, predpogoj pa je, da so ti shranjeni v obliki, ki omogoča učinkovito poizvedovanje. Ker je velik del vsebine družbenih medijev podan v naravnem jeziku oziroma nestrukturirani obliki besedila, za analizo pogosto uporabljamo tehnike tekstovnega rudarjenja. Tekstovno rudarjenje je proces iskanja vzorcev v podatkih na nestrukturiranih podatkovnih zbirkah. Razlika v primerjavi s podatkovnim rudarjenjem je ta, da tekstovno rudarjenje predvideva dodatno fazo, v kateri se izvede preoblikovanje podatkov v strukturiano obliko [24].

Prvi korak po ekstrakciji podatkov je tako praviloma vselej ta, da vsebino dokumenta razčlenimo v manjše enote, poimenovane žetoni (angl. „tokens“). Žetoni v večini primerov vsebujejo posamezno besedo, lahko pa tudi ločila, če je to smiselno. Solr privzeto ignorira ločila in nam izlušči samo čiste besede. Za operacije tipa NLP so ločila v besedi včasih uporabna pri ugotavljanju gramatičnih pravil, kar lahko dosežemo z izbiro drugega razčlenjevalnika besedil [23].

V splošnem poznamo naslednje načine procesiranja žetonov:

- Korekcija začetnice, pri čemer ponavadi vse besede sprmenimo v male črke, kar nam pomaga pri iskanju.
- Odstranjevanje neuporabnih besed (angl. „stopwords“).
- Razširitev žetonov s sinonimi, akronimi ter kraticami zaradi enake obravnave različnih uporabniških vnosov.
- Zaznamovanje govorjenega dela (angl. „part of speech tagging“), kot so časi, sklanjatve, množina in podobno.
- Preoblikovanje besed v korensko obliko (angl. „stemming“)

Večino naštetega omogoča tudi okolje Solr z nič ali nekaj dodatne konfiguracije. Vidimo torej, da je dobro vnaprej vedeti, kaj bomo s podatki počeli, ker na tak način lahko zagotovimo sprotno shranjevanje podatkov v pravi obliki.

5.1 Podatkovna shema

Čeprav zna Solr v načinu „schemaless“ sam določiti podatkovne tipe in poimenovanja polj, je koristno eksplicitno definirati podatkovno shemo za posamezna polja, ker na tak način optimiziramo tako shranjevanje podatkov kot tudi poizvedovanje po njih.

Ker imajo lahko vhodne datoteke v formatu JSON vgnezdeno obliko, se je najprej potrebno odločiti, na kakšen način bomo definirali poimenovanja posameznih polj v dokumentu sistema Solr. Pri tem imamo na voljo dva pristopa. V poimenovanju stolpca lahko ohranimo hierarhijo elementov izvornega dokumenta. Vgnezdena polja so v tem primeru ločena s piko, iz imena polja pa lahko razberemo celotno pot od korenskega do končnega vozlišča. Primer poimenovanja polja datuma objave komentarja na sliki 4.9 bi bil v tem primeru: „snippet.topLevelComment.snippet.publishedAt“.

Prednost takšnega načina poimenovanja je, da pri oddaji datoteke na Solr ni

potrebno uporabljati mapirne sheme, ki definira preslikavo izvornih polj na končna. Ker v tem primeru nimamo sheme, je prednost tudi ta, da bo Solr sprejel celotno vsebino dokumenta, torej tudi tista polja, ki niso eksplicitno navedena v mapirni shemi. Slabost sta otežena berljivost in poizvedovanje po podatkih. Drugi pristop poimenovanja stolpcev zahteva, da ob oddaji podatkov na Solr navedemo mapirno shemo, v zameno za to pa so lahko poimenovanja kratka in enostavno berljiva. Enako polje iz predhodnega primera bi tako poimenovali: „publishedAt“.

5.1.1 Definicija sheme

Primer definicije sheme, ki smo je uporabili v našem primeru in uporablja kratka poimenovanja polj, je dokument za hranjenje vsebine video objav (slika 5.1).

```
<field name="id" type="string" multiValued="false" indexed="true" required="true" stored="true"/>
<field name="kind" type="text_general" multiValued="false" indexed="true" stored="true"/>
<field name="publishedAt" type="pdates" multiValued="false" indexed="true" stored="true"/>
<field name="title" type="text_general" multiValued="false" indexed="true" stored="true"/>
<field name="description" type="text_general" multiValued="false" indexed="true" stored="true"/>
<field name="channelId" type="text_general" multiValued="false" indexed="true" stored="true"/>
<field name="channelTitle" type="text_general" multiValued="false" indexed="true" stored="true"/>
<field name="tags" type="text_general" multiValued="true" indexed="true" stored="true"/>
<field name="categoryId" type="plongs" multiValued="false" indexed="true" stored="true"/>
<field name="liveBroadcastContent" type="text_general" multiValued="false" indexed="true" stored="true"/>
<field name="viewCount" type="plongs" multiValued="false" indexed="true" stored="true"/>
<field name="likeCount" type="plongs" multiValued="false" indexed="true" stored="true"/>
<field name="dislikeCount" type="plongs" multiValued="false" indexed="true" stored="true"/>
<field name="favoriteCount" type="plongs" multiValued="false" indexed="true" stored="true"/>
<field name="commentCount" type="plongs" multiValued="false" indexed="true" stored="true"/>
<field name="duration" type="text_general" multiValued="false" indexed="true" stored="true"/>
<field name="dimension" type="text_general" multiValued="false" indexed="true" stored="true"/>
<field name="definition" type="text_general" multiValued="false" indexed="true" stored="true"/>
<field name="caption" type="booleans" multiValued="false" indexed="true" stored="true"/>
<field name="licensedContent" type="booleans" multiValued="false" indexed="true" stored="true"/>
<field name="projection" type="text_general" multiValued="false" indexed="true" stored="true"/>
<field name="topicCategories" type="text_general" multiValued="true" indexed="true" stored="true"/>
<field name="_text_" type="text_general" multiValued="true" indexed="true" stored="false"/>
<copyField source="*" dest="_text_"/>
```

Slika 5.1: Definicija polj za posamezno video objavo v konfiguracijski datoteki „managed-schema.xml“.

Slika prikazuje osnovni nabor polj. Za tista polja, ki jih pri izvedbi operacije „facet“ (več o tem v poglavju 5.2) želimo spremljati v nerazčlenjeni obliki, pa je potrebno definirati kopijo polja v podatkovnem formatu „string“. Pri

poimenovanju takšnega polja ponavadi dodamo pripono „_str“, s čimer ga razlikujemo od originalnega polja, ki je indeksu na voljo v razčlenjeni obliki.

Ob oddaji podatkov na Solr je skladno z definicijo podatkovne sheme potrebno navesti tudi mapirno shemo, ki strukturo izvornih polj preslika na končna polja dokumenta. Shemo lahko definiramo z navedbo parametrov, ki jih posredujemo pri izvedbi operacije POST na izbranem naslovu URL.

Primer klica operacije POST za podatkovno shemo, kot je definirana na sliki 5.1:

```
http://192.168.40.131:8983/solr/nifi_db/update/json/docs?split=/items&
f=id:/items/id&f=kind:/items/id/kind&f=etag:/items/etag&f=publishedAt:
/items/snippet/publishedAt&f=title:/items/snippet/title&f=description:
/items/snippet/description&f=channelId:/items/snippet/channelId&
f=channelTitle:/items/snippet/channelTitle&f=tags:/items/snippet/
tags&f=categoryId:/items/snippet/categoryId&f=liveBroadcastContent:
/items/snippet/liveBroadcastContent&f=viewCount:/items/statistics/
viewCount&f=likeCount:/items/statistics/likeCount&f=dislikeCount:
/items/statistics/dislikeCount&f=favoriteCount:/items/statistics/
favoriteCount&f=commentCount:/items/statistics/commentCount&f=duration:
/items/contentDetails/duration&f=dimension:/items/contentDetails/
dimension&f=definition:/items/contentDetails/definition&f=caption:
/items/contentDetails/caption&f=licensedContent:/items/contentDetails/
licensedContent&f=projection:/items/contentDetails/projection&f=
topicCategories:/items/topicDetails/topicCategories&commit=true
```

Vidimo, da definicija ukaza zahteva precejšnjo mero natančnosti, saj vsako polje, ki ga ne navedemo ali mu napačno določimo pot, ne bo zapisano v podatkovno bazo Solr.

Za prenos podatkov iz procesorjev YouTube na Solr v okolju Nifi uporabljamo privzeto komponento „PutSolrContentStream“. Nastavitev komponente za

sprejem podatkov iz naših procesorjev ni posebno zahtevna. Mapirno shemo v tem primeru navedemo v konfiguracijskem parametru „Content Stream Path“.

Poleg podatkovnega tipa je smiselno vsakemu posameznemu polju definirati še naslednje atribute:

- **indexed:** Določa, ali bo posamezno polje vključeno v indeksno kazalo, po katerem lahko poizvedujemo.
- **stored:** Omogoča vračanje vrednosti polja pri poizvedovanju po podatkih.
- **multiValued:** Dovoljuje shranjevanje seznama različnih elementov v enem polju.

Z definicijo sheme si torej nekoliko olajšamo kasnejše delo s podatki, ker jim damo vsaj neko obliko strukture. Nestrukturirana besedila so namreč neprimerljivo bolj kompleksna za analizo kot pa enostavni atomarni podatki, ki smo jih vajeni iz sistemov RDBMS. Razen tega na področju tekstovne analitike še manjka nek sistematičen, ekzakten nabor tehnik, ki bi podjetjem omogočala enostavno ugotavljanje pomena iz nestrukturiranih besedil. Glede na kompleksnost besedil in naravnega jezika je to seveda pričakovano. [27].

5.1.2 Posodabljanje dokumentov

Z namenom zagotavljanja konsistence podatkov je pomembno, da zagotovimo možnost posodabljanja zapisov po nekem ključu. Nifi procesor namreč velik del vsebine prenaša vedno znova (statistike, komentarji), pri čemer iz očitnih razlogov želimo minimizirati podvajanje podatkov na bazi.

Pri posodabljanju obstoječih zapisov se Solr sklicuje na ključ, po katerem ločuje med dokumenti in tudi izvaja operacije tipa CRUD. Privzeto je ključ poimenovan kot „id“, lahko pa ga tudi preimenujemo. Pomembna lastnost

postopka posodabljanja dokumenta po ključu je ta, da Solr vedno prepiše dokument v celoti. S tem ni težav, če pri posodabljanju dokumenta vedno navedemo vsa obstoječa (ali nova) polja. Če katerega izmed obstoječih polj v novem dokumentu izpustimo, pa se polje na dokumentu v bazi pobriše.

Solr ponuja več mehanizmov za delno posodabljanje dokumentov, pri čemer je najbolj praktičen postopek atomarnih posodobitev zapisov. Ta omogoča, da eksplicitno navedemo, katera polja in na kakšen način želimo posodobiti. Originalni dokument JSON tako opremimo z dodatnim elementom, ki na vsakem polju definira tip operacije. Primer izvedbe posodbitve polja števila ogledov ter brisanja elementov iz seznama:
"viewCount" : {"set" : "45876"},
"tags" : {"remove" : ["nature","forest"]}

Tista polja, ki niso navedena v dokumentu takšnega formata, ostanejo na bazi nespremenjena. Težava pri eksplicitnem navajanju ukazov na atomarnem nivoju je, da bi to pomenilo konkreten poseg v originalno strukturo datoteke JSON. S tem bi izgubili nekaj generičnosti rešitve, saj bi izhodne datoteke procesorjev Nifi morali dodatno transformirati v obliko, ki zagotavlja pravilno izvedbo posodbitve vsebine na bazi Solr. Če bi pri tem želeli biti natančni in zahtevati posodobitev samo tistih vsebin, ki so se dejansko spremenile, bi morali vsebino novih datotek primerjati z obstoječimi, kar bi dodatno upočasnilo zajeme in znatno zapletlo implementacijo rešitve.

Solr na srečo pozna tudi hibridni način, pri čemer ob detekciji atomarnega načina posodabljanja dokumenta sam posodobi le tista polja, ki so bila posredovana v novem dokumentu. Da bi sprožili tak postopek, je dovolj, da na katerem koli polju v dokumentu izvedemo eno izmed atomarnih operacij, na primer ukaz tipa „set“. Takoj ko sistem v strukturi dokumenta zazna omenjeni ukaz, bo posodobil le polja iz novega dokumenta, ostale pa bo pustil pri miru.

Takšen način se izkaže za dovolj dobro rešitev, ki se jo da tudi posplošiti. Namesto da bi ukaz „set“ izvedli na vsebinskem polju, je dovolj, da ga izvedemo na umetno generiranemu polju, ki nima vsebinske vrednosti. Primer takšnega atributa, ki ga vstavimo v vsako posodobitveno datoteko JSON, je: "sysUpd" : {"set" : "1"}

Z dodanim atributom dosežemo željeni način delovanja z minimalno modifikacijo vsebine originalne datoteke. Atribut lahko dodamo znotraj procesorja Nifi za prevzem podatkov družbenih medijev ali s katerim izmed transformacijskih komponent za delo z datotekami JSON. Delovanje dodatno pojasnjuje spodnji ilustrativni primer.

Originalni dokument:
"id" : "101"
"sysUpd" : "1"
"a" : "One"
"b" : "Two"

Posodobitveni dokument:
"id" : "101"
"sysUpd" : {'set' : "1"}
"b" : "Three"
"c" : "Four"

Novo stanje dokumenta:
"id" : "101"
"sysUpd" : "1"
"a" : "One"
"b" : "Three"
"c" : "Four"

5.2 Analitični primer

Proces ažuriranja podatkov smo zagotovili, v nadaljevanju si bomo pogledali nekaj primerov pregleda in analize podatkov. Pri tem nam je v veliko pomoč močan poizvedovalni jezik okolja Solr, ki ponuja številne načine analize podatkov. Poleg podpore poivedovanju, ki jo ponuja knjižnica Lucene, sta na primer dostopna še poizvedbeni razčlenjevalnik „DisMax“ ter analitični modul poimenovan „JSON Facet API“. Prvi omogoča uporabo enostavnejše sintakse, ponuja dodatne parametre, skrbi za minimalno število napak, omogoča uteženo poizvedovanje po več poljih in si za vzor jemlje način poizvedovanja Google. JSON Facet API po drugi strani omogoča izvedbo kompleksnejših poizvedb, ki jih posredujemo preko vmesnika JSON Request API.

Za testno množico bomo izbrali podatkovno zbirko „news_db“, ki smo jo predhodno napolnili s procesorji Nifi. Zbirka vsebuje objave iz družbenega medija YouTube, ki so jih v letu 2018 objavile nekatere večje medijske hiše. Polnjenje je potekalo s paralelno izvedbo procesorjev Nifi, pri čemer je vsak zajemal podatke za svoj kanal.

Z nekaj enostavnimi analitičnimi poizvedbami lahko hitro odgovorimo na nekatera zanimiva vprašanja, vezana na izbrano podatkovno zbirko. Med drugimi lahko ugotovimo, v katerih mesecih se je pojavilo največ objav, katera medijska hiša ima največ ogledov, katere besede so se najbolj pogosto pojavljale v novicah, katere objave so bile najbolj komentirane, katere so bile sprejete najbolj pozitivno ali negativno in podobno. Odgovore na omenjena vprašanja lahko pridobimo tudi neposredno v analitičnem oziroma vizualizacijskem orodju, katerega primer je predstavljen v poglavju 5.3.

Pri kompleksnejšem poizvedovanju se ponavadi ne moremo izogniti uporabi operacije „faceting“, ki jo ponuja okolje Solr. Ta omogoča številne načine uporabe, pri vseh pa gre v osnovi za agregacijo oziroma grupiranje iskalnih zadetkov v kategorije. Slika 5.2 prikazuje primer vgnezdene poizvedbe tipa

JSON Facet API, ki jo lahko pošljemo na URL naslov:
http://.../solr/news_db/query.

```
{
    query:"*:*",
    limit:0,
    facet:{
        top_channels:{
            type:terms,
            field:channelTitle_str,
            sort:{avg_views:desc},
            limit:2
            facet:{
                avg_views:"avg(viewCount)",
                avg_comments:"avg(commentCount)",
                avg_likes:"avg(likeCount)",
                top_titles:{
                    type:terms,
                    field:title_str,
                    sort:{comments:desc},
                    limit:2,
                    facet:{
                        views:"max(viewCount)",
                        comments:"max(commentCount)",
                        likes:"max(likeCount)"
                    }
                }
            }
        }
    }
}
```

Slika 5.2: Primer strukture vgnezdenega zahtevka tipa „facet“.

S poizvedbo pridobimo prva dva kanala z največjim povprečnim številom ogledov na posamezno objavo, za vsakega od obeh pa poiščemo še dve objavi, ki sta bili najbolj komentirani. Za oba kanala nas poleg ogledov zanimata še povprečno število komentarjev ter povprečno število všečkov na posamezno objavo. To informacijo želimo pridobiti tudi za najbolj komentirani objavi. Poizvedba je strukturirana na naslednji način:

- S prvima vrsticama povemo, da želimo iskati po celotni podatkovni zbirki, vendar ne vračamo posameznih dokumentov iz zbirke.
- V naslednjem nivoju deklariramo množico rezultatov, ki bo sestavljena iz posameznih kanalov in definiramo vsa tri povrečenja. Množico uredimo glede na povprečno število komentarjev kanala po padajočem vrstnem redu.

- Sledi definicija množice drugega nivoja, ki tokrat prikazuje naslove objav. Tudi v tem primeru določimo vsa štetja, objave pa uredimo padajoče po številu komentarjev.
- Pri obeh množicah se zaradi lažje ponazoritve omejimo samo na prva dva zapisa.

Poslano poizvedbo bi namesto v formatu JSON lahko zapisali tudi klasično, z navedbo posameznih parametrov. Slika 5.3 prikazuje primer rezultata, ki ga Solr vrne za zgornjo poizvedbo. Vidimo, da rezultat vsebuje vse podatke, po katerih smo povpraševali. Vsebina posameznih množic je navedena v elementu „buckets“, naziv posameznih elementov pa z elementom „val“. Pomemben je tudi podatek „count“, ki navaja, koliko elementov je na voljo v bazi za posamezno vozlišče. S poljubnim gnezdenjem elementov lahko na tak način enostavno generiramo kompleksne hierarhije poizvedb.

5.3 Vizualizacija podatkov

Podatkovno zbirko iz poglavja 5.2 smo na koncu želeli prikazati tudi v vizualni obliki, ki omogoča enostavno pregledovanje podatkov in spremljanje rezultatov z minimalnim zamikom glede na čas objave na družbenem mediju.

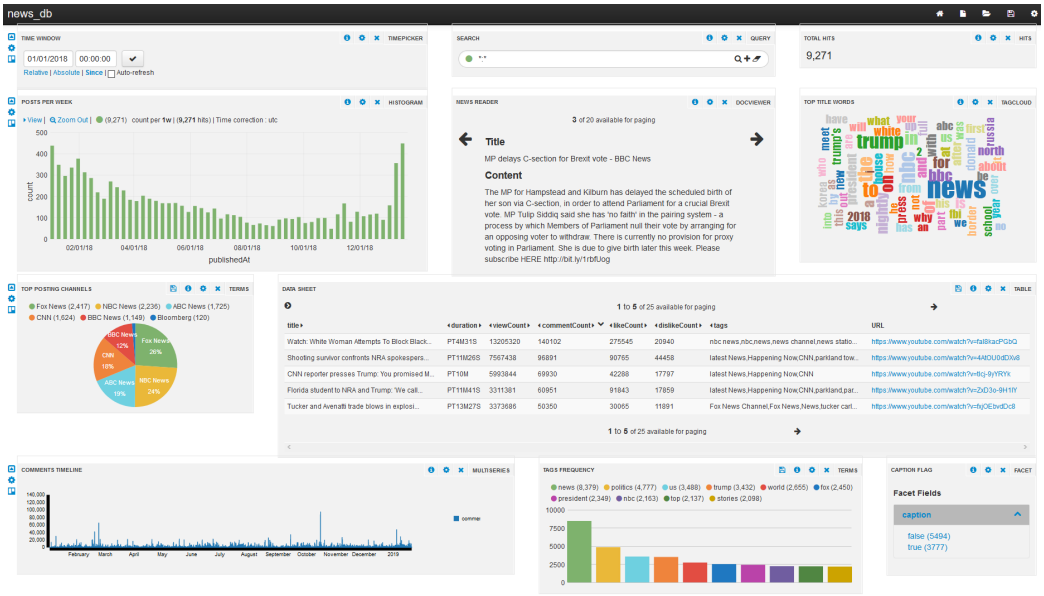
V orodju Banana smo definirali novo nadzorno ploščo tipa „Time-series dashboard“, ker želimo objave spremljati skozi čas. Sledi izbira podatkovne zbirke ter časovnega parametra, ki se potem uporabi na vseh komponentah s prikazom podatkov na časovni premici. V našem primeru je to časovna točka objave, ki je poimenovana „publishedAt“. S tem je osnovna konfiguracija nadzorne plošče zaključena in pričnemo lahko z dodajanjem posameznih komponent za različne analize podatkov. Okolje Banana ponuja precejšnje število takšnih komponent, za demonstracijo smo izbrali nekaj različnih. Končni rezultat nadzorne plošče prikazuje slika 5.4.

```
{
  "count":9271,
  "top_channels":{
    "buckets":[{
      "val":"CNN",
      "count":1624,
      "avg_comments":3499.7272167487686,
      "top_titles":{
        "buckets":[
          {
            "val":"Shooting survivor confronts NRA spokesperson Dana Loesch",
            "count":1,
            "comments":96891,
            "views":7567438,
            "likes":90765
          },
          {
            "val":"CNN reporter presses Trump: You promised Mexico would pay for wall",
            "count":1,
            "comments":69930,
            "views":5993844,
            "likes":42288
          }
        ]},
      "avg_likes":3174.4254926108374,
      "avg_views":376414.14285714284
    },
    {
      "val":"Bloomberg",
      "count":120,
      "avg_comments":533.4166666666666,
      "top_titles":{
        "buckets":[
          {
            "val":"How Australia's Gun Control Experiment Worked",
            "count":1,
            "comments":2852,
            "views":223557
          },
          {
            "val":"The Glamorous Life of a Pro Gamer",
            "count":1,
            "comments":2496,
            "views":999328,
            "likes":13700
          }
        ]},
      "avg_likes":4577.36974789916,
      "avg_views":260913.7
    }]
  }
}
```


Slika 5.3: Primer odgovora na zahtevek iz slike 5.2.

Na nadzorni plošči so od leve proti desni in od zgoraj navzdol prikazane naslednje komponente:

- **Timepicker:** Komponenta za izbiro časovnega intervala prikaza podatkov. Omogoča izbiro relativnega in absolutnega časa ali začetne točke prikaza. Na primeru so izpisani vsi podatki od leta 2018 dalje.
- **Query:** Vnos poljubne poizvedbe za omejevanje rezultatov prikaza.
- **Hits:** Število zadetkov, ki ustreza kombinaciji vseh trenutno vključenih filtrov.
- **Histogram:** Prikaz števila zapisov skozi čas s kopico možnih nastavitev. Primer beleži število novih objav na tedenskem nivoju.



Slika 5.4: Primer vizualizacije podatkov, prevzetih s procesorji Nifi.

- **Docviewer:** Komponenta za pregledovanje vsebine dokumentov. V našem primeru jo uporabljamo za branje vsebine zadnjih dvajsetih novic.

- **Tagcloud:** Besedni oblak, ki na prikazanem primeru prikazuje 100 najpogosteje uporabljenih besed v naslovih objavljenih novic.

- **Terms:** Vizuelni prikaz rezultata poizvedbe tipa „facet“. Izbrali smo možnost tortnega grafikona, ki prikazuje delež objav po posameznih kanalih.

- **Table:** Komponenta za pregledovanje dokumentov v tabelarični obliki. Omogoča enostavno dodajanje ali odvzemanje prikazanih polj, sortiranje zapisov in podobno. Vsebine, ki niso prikazane na maski, enostavno vpogledamo z odpiranjem posameznih vrstic. Komponenti smo dodali tudi poljubno definirano polje URL, ki sicer ne obstaja v izvornem dokumentu. S kombinacijo URL naslova ter identifikatorja video objave enostavno generiramo hiperaktivne povezave, ki nas s klikom pripeljejo

do originalnih objav.

- **Multiseries:** Dodatna časovna komponenta, ki omogoča prikaz drugačnih podatkov kot je število zapisov. Uporabili smo jo za prikaz intenzivnosti komentiranja skozi čas.
- **Terms:** Še enkrat komponenta za vizualizacijo podatkov tipa „facet“, tokrat uporabljena za prikaz desetih najpogostoje uporabljenih zaznamkov.
- **Facet:** Komponenta za prikaz rezultata poizvedbe tipa „facet“ v tekstovni obliki. Na prikazanem primeru označujemo video objave, ki imajo podnapise.

Vidimo, da okolje ponuja precej možnosti za poizvedovanje po podatkih. Poleg jasnosti samega prikaza je prednost tudi ta, da se končnim uporabnikom nadzorne plošče v večini primerov niti ni potrebno ukvarjati s pisanjem konkretnih poizvedb. Z nekaj kliki lahko povsem spremenimo postavitev nadzorne plošče in damo prednost tistim vsebinam, ki nas v danem trenutku najbolj zanimajo.
Frekvenco osveževanja lahko nastavimo na nekaj sekund, s čimer uporabniki dobijo visoko stopnjo ažurnosti podatkov. Če imamo v ozadju delujočih nekaj procesorjev Nifi, ki se recimo prožijo na vsakih 30 sekund, to pomeni, da lahko uporabniki novo vsebino pregledujejo na nadzorni plošči s samo nekajsekundnim zamikom.

Poglavje 6

Zaključek

Na konkretnem primeru smo uspešno izvedli celovit proces prevzema in priprave podatkov za nadaljno analizo. Hkrati smo ugotovili, da je tematika izjemno obsežna in ponuja brezmejne možnosti za nadaljno raziskovanje. Primer je transformacija podatkov, ki smo se je le bežno dotaknili, v praksi pa lahko vzame velik delež časa pri izvedbi. Zanimivi so še številni analitični primeri, ki se odpirajo že na eni sami podatkovni zbirki. Odziv na objave bi lahko dodatno ugotavljali z analizo sentimenta na besedilu komentarjev, primerjali bi lahko način poročanja različnih medijskih hiš za isto novico, analizirali dejansko vsebino objav iz podnapisov in podobno.

6.1 Možnosti za izboljšavo

Sama narava rešitve je dovolj kompleksna, da ponuja veliko možnosti za izboljšavo. Procesorje Nifi za zajem podatkov bi bilo z nekaj dodatnega dela mogoče še občutno izboljšati. Če navedemo samo nekatere primere, je to recimo možnost prevzema vsebine v formatu „gzip“, s čimer bi dodatno zmanjšali količino prometa, ki poteka med sistemi. Veliko lahko naredimo še na sami robustnosti delovanja z boljšo podporo večnitnemu delovanju, izboljšavo dnevnika za beleženje napak, ponavljanjem zajemov v primeru napak in zagotavljanjem višje stopnje varnosti pri delu s poverilnicami.

Dodatno bi lahko razširili nabor nastavitev, ki jih ponuja procesor, z večjim številom možnosti pri določanju načina zajema ter omejevanja izvedbe zajema podatkov. Pri rednem osveževanju vsebin, ki smo jih že prenesli, bi recimo lahko ponudili opcijo, da se osvežuje samo določeno število objav po nekem kriteriju, na primer samo tiste z največjim številom ogledov. Nadalje bi lahko navzven odpri cenik posameznih klicev, da bi ga skrbniki sistema lahko enostavno urejali brez posega v kodo rešitve.

Smiselno bi bilo implementirati tudi dodatne procesorje, s katerimi bi lahko v celoti pokrili vsebine, ki jih ponuja vmesnik YouTube API. Primer so recimo predvajalni seznami, aktivnosti, podnapisi, kategorije kanalov, subskripcije na kanale, t.i. „watermarks“, prevodi vsebin v druge jezike in podobno. Poseben podvig bi predstavljala implementacija prenosov dejanskih video posnetkov, ki so prav tako uporabni za nekatere analize, napr. t.i. „video clipping“.

Precej bi bilo mogoče narediti tudi z vidika generičnosti rešitve. Z definicijo javanskih vmesnikov (angl. „interface“) bi lahko zagotovili strukturo vseh potrebnih metod, ki bi se implementirale pri programiranju zajemov iz drugih družbenih medijev in podobno.

Veliko prostora za izboljšavo ostane seveda tudi na okolju Solr, predvsem z vidika optimizacije podatkovnega modela in izrabe širšega nabora funkcionalnosti, ki jih omogoča takšen sistem.

6.2 Sklepne ugotovitve

V nalogi ugotavljamo, da se ravno v času njenega nastanka dogajajo večje spremembe na področju dostopa do javno objavljenih vsebin družbenih medijev. Številni incidenti in zlorabe osebnih podatkov ter uvedba novih regulativ

so povzročili večjo mero previdnosti pri vseh ponudnikih družbenih omrežij. Ti so vedno bolj pazljivi pri načinu ponujanja dostopa do vsebin in uvajajo nove pogoje uporabe ter omejitve. Presenetila nas je zlasti napoved enega izmed največjih družbenih medijev Google+, ki v kratkem umika svojo storitev.

Kljub zaostrovanju pogojev uporabe večina družbenih medijev v tem hipu še vedno omogoča visoko mero dostopnosti do svojih podatkovnih zbirk, kamor lahko dostopamo preko spletnih aplikacijskih vmesnikov. To seveda velja za vse javno objavljene vsebine, medtem ko do zasebnih vsebin nimamo dostopa. Ponudniki storitev se sicer trudijo uporabnikom omogočiti čim višjo stopnjo varnosti in razpoložljivosti platforme, čemur so pravzaprav namenjene tudi kvote in omejitve dostopa na izbranem časovnem intervalu.

Dobre prakse navajajo uporabo protokola OAuth prve ali druge generacije in čim bolj enakomerno obrementev API-ja skozi čas. Prizadevati si moramo k zmanjševanju prometa, ki ga ustvarjamo med različnimi sistemi. Pri prevzemu podatkov moramo paziti na njihovo vestno uporabo in spoštovati zasebnost uporabnikov ter upoštevati pogoje uporabe, četudi gre za podatke javnih objav.

Pri pregledu konkretnih primerov družbenih medijev smo ugotovili, da vsi omogočajo relativno enostaven zajem vzorca podatkov. Če želimo opraviti celovit zajem za izbrano vsebino in zagotoviti trajno delujočo rešitev, pa je potrebno nekoliko več truda pri implementaciji. Pri tem moramo upoštevati tako zajem sprotnih objav kot prevzem zgodovine objav za nazaj, predvideti je potrebno tudi možnosti izpada delovanja ter omogočiti čim bolj enostavno prilagajanje spremembam omejitev pri uporabi API-ja.

Ker na družbenih medijih nastajajo velike količine podatkov, moramo zagotoviti primerno obliko njihovega shranjevanja. Temu so namenjene teks-

tovne iskalne baze, ki omogočajo učinkovito poizvedovanje ter analizo velikih količin besdil. Kljub temu da takšna baza ne zahteva stroge navedbe strukture podatkov, je smiselna uporaba podatkovnih shem, s katerimi optimiziramo shranjevanje ter poizvedovanje po podatkih.

Za lažji nadzor nad velikimi podatkovnimi zbirkami so nam v veliko pomoč tudi vizualizacijska orodja, kjer z raznimi oblikami časovnic ter grafikonov enostavno spremljamo kvantiteto in kvaliteto podatkov.

Družbeni mediji v splošnem še vedno pridobivajo priljubljenost in imajo vedno večjo vlogo v družbi. To je dodaten razlog za čim bolj varno in odgovorno ravnanje s podatki, ki sicer ponujajo visoko dodano vrednost ne samo v poslovnem smislu, ampak tudi z vidika spremljanja aktualnega dogajanja na vseh področjih, na primer znanosti ter poglobljenega razumevanja socialnih interakcij med uporabniki družbenih omrežij oziroma ljudi na splošno.

Literatura

[1] Software Integrity Blog. Dosegljivo: https://www.synopsys.com/blogs/software-security/oauth-2-0-vs-oauth-1-0/, 2016. [Dostopano 23. 12. 2018].

[2] About LinkedIn. Dosegljivo: https://news.linkedin.com/about-us#statistics, 2018. [Dostopano 28. 12. 2018].

[3] Facebook Reports Third Quarter 2018 Results. Dosegljivo: https://investor.fb.com/investor-news/press-release-details/2018/Facebook-Reports-Third-Quarter-2018-Results/default.aspx, 2018. [Dostopano 3. 1. 2019].

[4] Facebook–Cambridge Analytica data scandal. Dosegljivo: https://en.wikipedia.org/wiki/Facebook%E2%80%93Cambridge_Analytica_data_scandal, 2018. [Dostopano 3. 1. 2019].

[5] GDPR Key Changes. Dosegljivo: https://eugdpr.org/the-regulation, 2018. [Dostopano 23. 12. 2018].

[6] java.util.Properties. Dosegljivo: https://docs.oracle.com/javase/tutorial/essential/environment/properties.html, 2018. [Dostopano 1. 10. 2018].

[7] Rate Limiting. Dosegljivo: https://developer.twitter.com/en/docs/basics/rate-limiting.html, 2018. [Dostopano 1. 12. 2018].

[8] Rate Limiting on the Graph API. Dosegljivo: https://developers.facebook.com/docs/graph-api/advanced/rate-limiting, 2018. [Dostopano 1. 12. 2018].

[9] Tumblr. Dosegljivo: https://en.wikipedia.org/wiki/Tumblr, 2018. [Dostopano 23. 12. 2018].

[10] Varstvo podatkov in zasebnost na spletu. Dosegljivo: https://europa.eu/youreurope/citizens/consumers/internet-telecoms/data-protection-online-privacy/index_sl.htm, 2018. [Dostopano 22. 12. 2018].

[11] Access Tokens. Dosegljivo: https://developers.facebook.com/docs/facebook-login/access-tokens, 2019. [Dostopano 3. 1. 2019].

[12] API access that scales with you and your solution. Dosegljivo: https://developer.twitter.com/en/pricing.html, 2019. [Dostopano 4. 1. 2019].

[13] Google+. Dosegljivo: https://en.wikipedia.org/wiki/Google%2B, 2019. [Dostopano 22. 1. 2019].

[14] Importance of social media for different corporate functions. Dosegljivo: https://commons.wikimedia.org/wiki/File:Importance_of_social_media_for_different_corporate_functions.png, 2019. [Dostopano 4. 1. 2019].

[15] Rate Limiting. Dosegljivo: https://developer.twitter.com/en/docs/basics/rate-limiting.html, 2019. [Dostopano 4. 1. 2019].

[16] Refreshing User Access Tokens. Dosegljivo: https://developers.facebook.com/docs/facebook-login/access-tokens/refreshing, 2019. [Dostopano 3. 1. 2019].

[17] Using the Graph API. Dosegljivo: `https://developers.facebook.com/docs/graph-api/using-graph-api/`, 2019. [Dostopano 3. 1. 2019].

[18] Bogdan Batrinca and Philip C. Treleaven. Social media: analytics a survey of techniques, tools and platforms. *AI & SOCIETY*, 30(118):89–116, 2014.

[19] Bryan Bende. Indexing Tweets with NiFi and Solr. Dosegljivo: `https://blogs.apache.org/nifi/entry/indexing_tweets_with_nifi_and`, 2015. [Dostopano 28. 12. 2018].

[20] Charles Bihis. *Mastering OAuth 2.0*. Packt Publishing, 2015.

[21] Saurabh Chhajed. *Learning ELK Stack*. Packt Publishing, 2015.

[22] Christopher Gambino, Dan Rice, Joseph Niemiec, Mark Johnson, and Jordan Martz. *Apache® NiFi™ For Dummies®, Hortonworks and Attunity Special Edition*. John Wiley & Sons, Inc, 2018.

[23] Grant S. Ingersoll, Thomas S. Morton, and Andrew L. Farris. *Taming Text - How to Find, Organize, and Manipulate It*. Manning Publications, 2012.

[24] Parul Kalra. Text mining: Concepts, process and applications. *Journal of Global Research in Computer Science*, 4(3), 2013.

[25] Brittany Leaning Laura Fitton, Anum Hussain. *Twitter® For Dummies®, 3rd Edition*. John Wiley & Sons, Inc, 2015.

[26] Andrew Lim. Configuring an external ZooKeeper to work with Apache NiFi. Dosegljivo: `https://community.hortonworks.com/articles/135820/configuring-an-external-zookeeper-to-work-with-apa.html`, 2017. [Dostopano 28. 12. 2018].

[27] Tom Reamy. *Deep Text: Using Text Analytics to Conquer Information Overload, Get Real Value From Social Media, and Add Big(ger) Text to Big Data*. Information Today, Inc., 2016.

[28] Dikshant Shahi. *Apache Solr: A Practical Approach to Enterprise Search*. Apress®, 2015.

[29] David Smiley, Eric Pugh, Kranti Parisa, and Matt Mitchell. *Apache Solr Enterprise Search Server, Third Edition*. Packt Publishing, 2015.

[30] Rob Johnson Yoel Roth. New developer requirements to protect our platform. Dosegljivo: `https://blog.twitter.com/developer/en_us/topics/tools/2018/new-developer-requirements-to-protect-our-platform.html`, 2018. [Dostopano 4. 1. 2019].