SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2652

Biblioteka za dohvat novinskog članka s portala i pridruženih komentara nadruštvenim mrežama

Šimun Matešić

Zagreb, lipanj 2021.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Broj internetskih stranica neprestano raste. Radi se o golemom sadržaju i količini informacija koje su gotovo svakome na dohvat ruke. U to mnoštvo uključeni su i portali, odnosno stranice na kojima se objavljuju vijesti. Njihovo svojstvo vrijedno istraživanja je da sadrže mišljenja korisnika, odnosno komentare. Komentari pojedinačno ne nose veliku vrijednost, međutim dovoljno velik i kvalitetan skup komentara može služiti kao podloga za golem broj analiza koje koriste forenzici, ekonomiji ali i sociologiji. Bilo da se radi o analizama kupovnih navika ili političkih mišljenja, nepobitno je da komentari u sebi nose skriveni značaj. Kako bi se ostvarila bilo kakva vrsta analize, potrebno je osmisliti sustav koji će dohvaćati komentare i njihove pripadajuće novinske članke iz raznovrsnih izvora, obraditi ih tako da su spremni za pohranu te naposljetku i pohraniti.

Unatoč jednostavnosti ideje, postoje prepreke koje implementaciju takvog sustava čine kompleksnom i vrijednom istraživanja. Postoji velik broj sustava komentiranja, a još i veći broj portala koji ih koriste. Neki portali koriste vanjski-razvijene sustave za komentiranje poput *Disqus* ili *Facebook comments*. Drugi portali razvijaju vlastite. Bez obzira na sustave komentiranja, sami portali međusobno se razlikuju u pravilima kojima podliježe izvorni kod stranice. Predstavljena heterogenost sustava komentiranja, ali i specifičnost pojedinih portala onemogućava jedinstvenu implementaciju sustava za dohvat komentara i zahtijeva da se implementaciji pristupi pažljivo kako bi nadogradnje uzrokovale minimalne promjene.

Osim predstavljenog problema, izazov predstavlja i činjenica da se sustavi komentiranja mijenjaju. Osim sustava komentiranja, mijenjaju se i pravila HTML koda portala. Promjene mogu biti uzrokovane jednostavnim nadogradnjama na sustav komentiranja koje se ne očituju izvana, ali mogu biti i promjene koje onemogućuju rad razvijanog sustava. Promjene mogu biti trajne, ali mogu biti i privremene kao rezultat tzv. *anti-scraping* mehanizama koji nastoje onemogućiti rad sustava za prikupljanje komentara. Kako bi razvijeni sustav opstao u opisanoj dinamičnoj okolini o koju se oslanja mora biti robustan i znati rukovati promjenama te se prilagoditi kad god je mo-

guće.

Svrha ovog rada je izgradnja lako nadogradivog i robustnog sustava za prikupljanje komentara s portala čija je implementacija napisana u skladu s najboljim praksama programskog inženjerstva. U sklopu ovog rada bit će prikazano na koji način se prikupljaju komentari s različitih sustava komentiranja, kako se detektira i rukuje promjenama koje onemogućuju ispravan rad sustava, kako se izbjegava detekcija rada sustava te na koje načine je osmišljena arhitektura sustava i provedena implementacija kako bi se omogućila što lakša nadogradnja.

Rad je strukturiran na sljedeći način. Prvo je objašnjena klasifikacija sustava, zatim teorijska podloga za implementaciju tog dijela sustava. Teorijska podloga uključuje objašnjenje načina rada dvaju sustava komentiranja, izbjegavanje detekcije te koje promjene su moguće u sustavima komentiranja. Sljedeće je objašnjena arhitektura sustava uz objašnjenje uloga pojedinih modula i razreda. Zatim je detaljnije objašnjena implementacija pojedinih modula, gdje je ujedno i objašnjeno na koje načine je osigurana nadogradivost i robusnost sustava u svrhu rukovanja promjenama. Zadnje poglavlje služi kao demonstracija rada sustava kao cjeline te njegovih modula.

2. Klasifikacija razvijenog sustava

Sustav za dohvat komentara primarno je *web scraper*, ali sadrži i svojstva *web crawlera*. Web scraperi su sustavi koji preuzimaju izvorne internetske stranice te ih obrađuju, odnosno ekstrahiraju podatke iz njih. Crawleri su pak sustavi koji samostalno pronalaze poveznice na internetske stranice. Opisani sustavi međusobno su komplementarni te surađuju na način da jedan pronalazi poveznice, a drugi preuzima njihov sadržaj. Crawler dio sustava relativno je jednostavan i ograničen na način da ne pronalazi sam nove portale, već samo nove članke na portalima koji su zadani u konfiguraciji. U tom smislu razvijeni sustav za prikupljanje komentara nije u strogom smislu crawler, no svakako sadrži svojstvo traženja novih poveznica koje je karakteristično crawlerima. Konkretan način rada bit će opisan u poglavlju o detaljima implementacije, a za sada je dovoljno reći da je sustav sposoban samostalno pronalaziti nove članke.

Scraping predstavlja srž sustava i najzanimljiviji skup implementacijskih problema koje je ovaj rad nastojao riješiti. Najveći problemi rezultat su različitosti sustava za komentiranje zbog kojih je potrebno prilagoditi scraping za svaki od sustava komentiranja. Osim temeljnog izazova postoje dva ozbiljna problema koja je također potrebno riješiti kako bi sustav bio robusan. Navedeni problemi su izbjegavanje detekcije scrapinga te prilagodba promjenama u sustavima komentiranja. Prilagodba promjenama vrlo je usko vezana uz implementaciju, no izbjegavanje detekcije scrapinga uže je vezano uz teorijsku podlogu nad kojom je sustav izgrađen. Više o izbjegavanju detekcije i rukovanju promjenama bit će riječi u potpoglavlju 3.3. Konkretna implementacija rukovanja promjenama bit će objašnjena u poglavlju o implementaciji sustava.

Scraping u razvijenom sustavu može se podijeliti na dva nejednaka posla, a to su preuzimanje članaka te preuzimanje pripadajućih komentara. Preuzimanje članaka trivijalan je posao koji se svodi na preuzimanje HTML koda od kojeg se sastoji web stranica. Međutim, kako bi se komentari mogli preuzeti potrebno je *reversingom* otkriti na koji način sustav komentiranja funkcionira. Preciznije, potrebno je promatrati

mrežni promet i izdvojiti te analizirati zahtjeve koji su zaslužni za prijenos komentara sa poslužitelja do preglednika kako bi se mogli iskoristiti programatski za automatski dohvat komentara. Nakon što je otkriven osnovni način rada svakog od promatranih sustava komentiranja potrebno je otkriti i koje promjene ili blokade su moguće te kako ih izbjeći ili ublažiti. U sklopu ovog rada proučavani su sustavi za komentiranje *Facebook comments plugin*, *Disqus*, sustav Večernjeg Lista te sustav portala 24sata. U sljedećem poglavlju će na primjeru Facebook comments plugina te Disqus platforme biti objašnjeno kako reverzati protokole koje koriste te iskoristiti pronađene zahtjeve i informacije da bi se dohvatili komentari, ali i kako izbjeći detekciju te koje promjene su moguće.

3. Teorija preuzimanja komentara

Ovo poglavlje dat će detaljni pregled postupka reverzanja protokola Facebook comments plugin i Disqus. Uz to, zadnja dva potpoglavlja dat će teorijsku podlogu izbjegavanju detekcije te mehanizmu rukovanja promjenama implementiranog u sustavu.

3.1. Facebook comments plugin

Facebook comments plugin najčešći je sustav komentiranja kojeg je moguće pronaći u uporabi na hrvatskim web portalima, ali i globalno popularnim stranicama. Razlog tomu je jednostavnost uporabe te činjenica da je za komentiranje potreban samo Facebook korisnički račun kojeg velik broj ljudi ima. Kako bi se mogla implementirati logika automatskog dohvata komentara s URL-a članka potrebno je istražiti koji zahtjevi su zaslužni za dohvat komentara. Filtriranjem mrežnog prometa pri učitavanju stranice lako je izdvojiti jedan GET zahtjev. Odgovor na zahtjev vraća HTML stranicu koja sadrži među ostalim i identifikator članka te početni skup komentara koji će se prikazati u pregledniku. Taj zahtjev zaslužan je za prvih nekoliko komentara koji će se prikazati kada korisnik učita članak. Oblik zahtjeva prikazan je u sljedećem odsječku.

```
https://www.facebook.com/plugins/feedback.php?href={}
```

Kako bi se zahtjev mogao poslati potrebno je `href` parametar postaviti na vrijednost URL-a članka. Pri početku istraživanja ustanovljeno je da nije potrebno postavljati zaglavlja zahtjevu, međutim pri kraju pisanja rada dogodila se promjena u Facebook comments pluginu koja zahtijeva dodavanje zaglavlja. Konkretno, potrebno je u zaglavlju uključiti `Cookie` ključ s vrijednošću `c_user=1`. Više o vrijednostima zaglavlja bit će u potpoglavlju o izbjegavanju detekcije.

S obzirom na to da ovaj zahtjev dohvaća samo prvih nekoliko komentara, potrebno je pronaći način za dohvatiti sve komentare povezane s člankom. Novi komentari se u slučaju Facebook comments plugina učitavaju pritiskom na gumb ‘Učitaj još komentara‘. Istraživanjem zahtjeva koji se šalje pritiskom na gumb dolazi se do rješenja kako

dohvatiti sve komentare u jednom pokušaju. Pritiskom na gumb šalje se POST zahtjev čiji je rezultat skup sljedećih nekoliko komentara kojima će se proširiti originalni skup. Zahtjev kojeg šalje preglednik pritiskom na gumb je oblika.

`https://www.facebook.com/plugins/comments/async/target_id/pager/time/`

Prikazani POST zahtjev kojeg šalje preglednik u tijelu zahtjeva sadrži više podataka od kojih nisu svi potrebni kako bi se zahtjev ispunio. Analizom ustanovljeno je da su samo dvije vrijednosti u tijelu zahtjeva potrebne kako bi zahtjev vratio sve komentare povezane s člankom. To su `__a` te `limit`. Vrijednost `__a` u tijelu zahtjeva proizvoljan je broj, odnosno bilo koji broj se može poslati dokle god ključ nije izostavljen. Vrijednost pod ključem `limit` ograničava broj komentara koji će biti vraćeni u odgovoru. S obzirom na to, jednostavno rješenje je postavljanje ograničenja na neki veliki broj poput deset tisuća. U slučaju da se izostavi ključ `limit` njegova vrijednost bit će pretpostavljenih sto. Zaglavlja zahtjeva nije potrebno postavljati kako bi se zahtjev ispunio.

Postoji međutim završni problem kojeg je potrebno rješiti kako bi prethodno pokazani POST zahtjev rezultirao punim skupom komentara. POST zahtjev sadrži parametar `target_id` koji je u stvarnosti identifikator članka u kontekstu Facebookovog comments plugina. Vrijednost `target_id` parametra moguće je dobiti iz HTML koda početnog GET zahtjeva. Unutar HTML koda postoji JSON struktura, a unutar nje nalazi se ključ `targetFBID`. Vrijednost pod tim ključem upravo je identifikator članka koji je potreban za slanje slanje asinkronog POST zahtjeva. Kako bi se parsirao HTML kod i dobio identifikator članka, potreban je regularni izraz

`"targetFBID":"([0-9]+)"`.

Vrijednost koju traženje po opisanom izrazu vraća je identifikator članka.

Nakon što je parsiran identifikator članka, svi preduvjeti za slanje POST zahtjeva su ispunjeni. Rezultat zahtjeva je JSON podatkovna struktura koji sadrži sve komentare i informacije o autorima. Takav JSON spreman je za obradu i spremanje u bazu podataka zajedno s preuzetim člankom.

3.2. Disqus

Sustav komentiranja Disqus nije popularan među hrvatskim novinskim portalima, ali predstavlja zanimljiv problem za riješiti zbog toga što je u širokoj uporabi u svijetu. Zbog toga je vrijedno saznati kako radi te osmisliti način da sustav prikupljanja komentara može podržati i stranice koje koriste Disqus.

Naime, kako bi se dohvatili komentari s članka koji koristi Disqus potrebno je prvo pronaći identifikator niza komentara, odnosno *thread id*. Identifikator niza komentara moguće je pronaći u rezultatu GET zahtjeva
`https://disqus.com/embed/comments/?f=news_portal_name&t_u=article_url`

Parametar `f` s vrijednošću `news_portal_name` jednostavno je naziv portala, no u blago izmijenjenom obliku. Primjerice, Escapist Magazine imat će parametar `f` 'escapist-magazine'. Parametar `t_u` je URL članka u neizmijenjenom obliku. U većini slučajeva ovakav će zahtjev biti uspješno ispunjen, međutim za neke portale potrebno je modificirati parametre GET zahtjeva, konkretno parametar `t_u`. U nekim slučajevima potrebno je zamijeniti `t_u` s parametrom `t_i` čija vrijednost nije URL članka već broj članka. Broj članka vrijednost je koja se može pronaći kao sufiks URL-ovima nekih članaka koji koriste Disqus. Vrijednost broja članka moguće je ekstrahirati iz URL-a članka korištenjem regularnog izraza `([0-9]{6})`. Ne postoje stroga pravila koja definiraju kada je potrebno jedan, a kada drugi parametar uključiti u zahtjev i zbog toga postoji određena doza pogađanja tijekom dohvata identifikatora niza komentara. Rezultat GET zahtjeva sadrži identifikator niza komentara kojeg je moguće ekstrahirati regularnim izrazom `"thread":"([0-9]+)"`.
Komentari se dohvaćaju GET zahtjevom na URL
`https://disqus.com/api/3.0/threads/listPostsThreaded?limit=100&thread={}&cursor={}&api_key={}`. Parametar `thread` je identifikator niza komentara koji je dobiven u prethodnom koraku. Parametar `api_key` nužan je parametar kojeg nije potrebno mijenjati. Odnosno, `api_key` se jednom pročita iz zahtjeva koje šalje preglednik i dalje se može koristiti bez vremenskih ograničenja. Parametar `cursor` služi kako bi se komentari mogli iterativno dohvaćati sve dok ih ne ponestane. Oblik `cursor` vrijednosti je `{}.0.0` gdje je vitičaste zagrade potrebno zamijeniti brojem stranice s komentarima počevši od nule. Tako prva stranica s komentarima ima cursor `0.0.0`, sljedeća ima vrijednost `1.0.0` na dalje. Naime, Disqus se tu razlikuje od Facebook comments plugina. Parametar `limit` je kod Disqus sustava ograničen na maksimalnu vrijednost sto. Bilo koja vrijednost iznad rezultirat će neispunjenim zahtjevom, tj. greškom. Zbog toga nije moguće postavljenjem visokog `limit` parametra dohvatiti sve komentare u jednom pozivu. Kako bi se dohvatili svi komentari, potrebno je iskoristiti podatke koji su vraćeni u JSON obliku kao rezultat zahtjeva. Pod ključem `cursor` nalazi se ugnježđena JSON struktura koja sadrži ključeve: `hasPrev`, `prev`, `total`, `hasNext`, te **`next`**. Posebno interesantne su vrijednosti pod ključevima `hasNext` te **`next`**. Naime, `hasNext` ključ sadrži vrijednost istine ili neistine te ukazuje na to ima

li komentara koji još nisu preuzeti. Komplementarno `hasNext` ključu stoji **`next`** ključ čija je vrijednost u stvari vrijednost `cursor` parametra za sljedeći skup komentara. Kada je `hasNext` neistinit, više ne postoji sljedeće stranice s komentarima i preuzimanje komentara je završeno.

3.3. Izbjegavanje detekcije

Mnoge stranice aktivno se bore protiv scrapinga, odnosno pokušavaju detektirati koji zahtjevi ne dolaze od legitimnih korisnika nego od scrapera. Postoji nekoliko metoda ScraperAPI (2019) kojima scraper izbjegava detekciju. Neke od njih koje je moguće implementirati u sustavu su dodavanje zaglavlja zahtjevima te implementiranje eksponencijalnog *backoff* algoritma kod ponovnih pokušaja. Naime, kada preglednik šalje zahtjev na URL postavlja određena zaglavlja. Tipična zaglavlja koja preglednik postavlja su `Accept`, `Accept-Encoding`, `Accept-Language`, `Connection`, `Cookie`, `Host` ali i brojne druge. Takva zaglavlja ukazuju da je riječ o legitimnom korisniku, a ne o *botu* koji šalje zahtjeve. Tako dodavanje vrijednosti `User-Agent`, `Accept-Encoding`, `Cookie`, `Referer` smanjuje mogućnost da će se sustav detektirati i dobiti privremenu zabranu zahtjeva. U suprotnom, zahtjev bez zaglavlja moguće je da će biti detektiran kao nelegitiman. Vjerojatnost detekcije dalje se može smanjiti nasumičnim odabirom ključeva u zaglavlju, ali i njihovih vrijednosti ScraperAPI (2019). Najbolja obrana od detekcije je promjena IP adrese ScraperAPI (2019), međutim takvo što izlazi iz okvira ovog rada.

Eksponencijalni backoff algoritam služi kako bi se prilikom neuspješnih zahtjeva napravilo više ponovnih pokušaja koji su međusobno odvojeni eksponencijalno rastućim vremenskim periodom. Takav mehanizam nije strogo mehanizam izbjegavanja detekcije već je način na koji se može boriti protiv blokiranja ako je sustav detektiran. U sustavu je implementiran jednostavan algoritam koji kao početnu vrijednost uzima nekoliko sekundi, a svaka sljedeća vrijednost jednaka je prethodnoj vrijednosti pomnoženoj predefiniranim faktorom dignutim na potenciju koja odgovara rednom broju pokušaja. Na taj način osigurava se da je između svakog rastuća vremenska razlika što povećava vjerojatnost uspješnog ponovnog pokušaja.

3.4. Promjene u sustavima

Sustavi za komentiranje ne mijenjaju se često, čak toliko rijetko da tijekom istraživanja i pisanja rada dogodila se samo jedna detektirana promjena. Unatoč tomu što su rijetka pojava, promjene koje nastupe imaju potencijal onemogućiti ispravan rad sustava. Na primjeru prethodno objašnjenog Disqus sustava komentiranja, promjena URL-a za dohvat identifikatora niza komentara rezultira nemogućnosti da sustav parsira identifikator. Bez identifikatora ne može izgraditi zahtjev koji će dohvatiti komentare. Na takav tip promjena sustav ne može imati spremno rješenje iz razloga što je nemoguće predvidjeti u koju vrijednost će se ispravan URL promijeniti. Zbog toga je osmišljen sustav testiranja i hijerarhija iznimnih slučajeva koji pomažu dijagnostici i brzom razrješavanju problema. Ideja iza sustava testiranja je da se periodički dohvaćaju komentari iz poznatih izvora čiji ishod obrade je poznat. Tako ako sustav ne može dohvatiti komentare najvjerojatnije se radi o promjeni koju je potrebno razrješiti. Hijerarhija iznimnih slučajeva služi u redovnom radu sustava kako bi se eventualni neočekivani slučajevi lako klasificirali i podijelili u trajne i privremene probleme. Trajni problemi ukazuju na promjene u sustavima komentiranja, dok privremeni najčešće ukazuju na mrežne probleme, nedostupnost stranice ili blokadu zahtjeva. Detaljnije objašnjenje bit će u poglavlju o implementaciji zajedno s primjerom prave promjene koju je osmišljeni sustav pomogao razrješiti.

4. Arhitektura sustava

Prvenstvena briga pri izgradnji sustava bila je laka nadogradivost. Zbog toga je građen modularno uz nastojanje da razredi koji blisko surađuju se nalaze u istom modulu, dok oni koji ne surađuju su razdvojeni. Drugim riječima, nastoji se održati visoku koheziju, ali nisku spregu. Takva organizacija pomoći će lakoj nadogradnji u slučaju dodavanja podrške za nove sustave komentiranja ili pak izmjenama nad postojećim implementacijama. Poglavlje će biti razdvojeno po potpoglavljima za svaki od modula ili pojedinih skripti koji su ključni za rad sustava.

4.1. article

Article modul srž je sustava. Sadrži logiku dohvata članaka i komentara te njihovog spremanja. Article modul komunicira s ostatkom sustava kroz ArticleFetcher razred čija je uloga čekanje na naredbu da se obradi URL. ArticleFetcher će po primitku valjane naredbe putem ArticleFactoryja instancirati odgovarajući razred koji će znati rukovati s tom naredbom. Razredi koji znaju rukovati naredbama zapravo su razredi koji odgovaraju jednom tipu sustava komentiranja. Svaki sustav komentiranja ima vlastit Article razred, odnosno u sustavu postoje ArticleFacebook, ArticleDisqus, ArticleVecernji, Article24sata te bazni Article razred. Razlog postojanju više zasebnih razreda goleme su implementacijske razlike zbog kojih je ujedinjavanje koda u jedan razred nepraktično. Više o tome, ali i u ulozi baznog Article razreda bit će u poglavlju o implementaciji. Kako bi razredi sustava mogli obaviti svoju zadaću obrade URL-a potrebni su im pomoćni DateParser razredi. Razreda izvedenih iz DateParser ima mnogo zbog toga što svaka stranica s vijestima ima različite načine pisanja kada je članak objavljen. S obzirom na to da postoji velik broj konfigurabilnih elemenata u Article modulu, osmišljen je i modul koji će olakšati učitavanje konfiguracije. O tome u sljedećem potpoglavlju.

4.2. config loader

Modul `config_loader` sadržajno je malen, međutim igra veliku ulogu u sustavu. Temeljna ideja iza pisanja ovog modula je činjenica da gotovo svaki dio sustava ima barem nekoliko atributa ekstrahiranih u konfiguracijske datoteke, a kako bi se olakšalo mijenjanje i pregled konfiguracije potrebno je omogućiti da se ona učitava ili iz baze podataka ili iz INI datoteka u conf/ direktoriju. Tako, `config_loader` sadrži jedan bazni razred ConfigLoader te dva izvedena razreda FileConfigLoader te DbConfigLoader. FileConfigLoader služi čitanju konfiguracije iz datoteka, a DbConfigLoader čitanju iz baze podataka. Više o specifičnostima implementacije te kako se koriste u sustavu bit će riječi u poglavlju o implementaciji.

4.3. utils

`Utils` modul sadrži velik broj razreda od kojih su neki jednostavne *helper* klase, dok su neki pak ključni za rad sustava i zajedno s Article modulom čine osnovu sustava. Tako, u ovom poglavlju posebna pažnja bit će predana razredima: `DownloadURL`, `NewsMonitor` te `RefreshComments`. Ta tri navedena razreda međusobno komuniciraju, a jedan od njih komunicira i s ArticleFetcher iz Article modula. `NewsMonitor` služi praćenju novih vijesti, odnosno prati izvore koji govore kad je novi članak objavljen te obavijesti ostatak sustava da ga treba i preuzeti. `DownloadURL` razred služi slanju naredbi Article modulu, odnosno ArticleFetcher razredu. Zadnji od tri navedena razreda, `RefreshComments`, ima zanimljivu ulogu. Naime, u životnom vijeku jednog online članka broj komentara će rasti, ali i opadati s vremenom. Ono što je konstantno je da unutar kratkog vremenskog perioda od kad je članak objavljen obično nema mnogo komentara. Komentari će vjerojatno nekoliko dana brzo rasti, a zatim će njihov rast stagnirati. `RefreshComments` služi kako bi se optimalno ažurirali već preuzeti komentari ovisno o vremenu koje je prošlo od njihova objavljivanja.

4.4. checks

U uvodu u rad spomenut je jedan od najvećih izazova s kojim se razvijeni sustav mora nositi, a to su promjene u sustavima komentiranja. Kako bi se osiguralo da sustav pravovremeno detektira promjene, potrebno je definirati testove s člancima za koje je ishod obrade poznat. U tu svrhu stvoren je `checks` modul koji sadrži `FormatValidator` razred te nekoliko izvedenih razreda - po jedan za svaki sustav komentiranja. Navedeni

razredi služe provjeravanju formata preuzetih podatkovnih struktura, odnosno provjerava se jesu li strukture onakve kakvima ostatak sustava ih očekuje. `Checks` modul uparen je s `validate.py` skriptom čija je svrha objašnjena u sljedećem potpoglavlju.

4.5. validate

Skripta `validate.py` pokreće preuzimanje članaka i komentara za nekoliko predefiniranih URL-ova za koje je ishod poznat te poziva odgovarajući FormatValidator kako bi se provjerio format dobivenih rezultata. Skripta bilježi rezultate testiranja te ih sprema u bazu podataka.

4.6. run

Kako bi se moglo konfigurirati pokretanje sustava, napravljena je skripta koja pokreće sustav prema predanim argumentima. Tako je moguće pokrenuti `ArticleFetcher`, `RefreshComments` ili `NewsMonitor` zajedno, ali i zasebno ovisno o predanim argumentima.

5. Implementacija

Ovo poglavlje objasnit će koje su tehnologije korištene i iz kojih razloga, nastojat će otkriti detalje implementacijski najzahtjevnijih dijelova sustava, a i na što se posebno pazilo tijekom implementacije kako bi sustav bio pisan u skladu s najboljim praksama programskog inženjerstva.

5.1. Tehnologije implementacije

Programski jezik odabran za implementaciju sustava je Python. Razlog odabira Pythona je jednostavnost sintakse, velika zajednica programera i golema baza napisanog koda dostupnog na internetu, dobre potporne biblioteke te činjenica da se razvojni tim dobro snalazi s odabranim jezikom. Osim programskog jezika, bitno je odabrati bazu podataka te mehanizam komunikacije u sustavu.

Baza podataka birana je prema strukturama podataka koje će se spremati. S obzirom na to da su članci i komentari u raznovrsnim formatima, a njihovo formatiranje ne predstavlja unaprjeđenje u sustavu, odabrana je NoSQL baza. NoSQL baze popularan su izbor za skladištenje nestrukturiranih podataka (MongoDB, 2021). U ovom slučaju odabir klasičnih baza podataka predstavljao bi dodatan implementacijski izazov koji ne donosi dobitak u radu sustava, štoviše usporio bi rad sustava zbog vremenskih troškova formatiranja. Konkretno, odabrana je MongoDB baza podataka zbog toga je dobro dokumentirana te postoji jednostavna podrška u obliku biblioteke za Python.

Komunikacija u sustavu bazirana je na komunikaciji zadataka. Za takve potrebe najprimjerenije je odabrati komunikaciju temeljenu na redovima, odnosno engl. *queue-based communication*. Konkretna tehnologija koju se koristi je RabbitMQ. Razlog odabira su dobra dokumentacija, jednostavnost korištenja te podrška u Pythonu.

5.2. Metodologija implementacije

Tijekom implementacije posebna pažnja dana je poštivanju najboljih praksi programskog inženjerstva, ali i praksi specifičnih za Python. Velik izazov tijekom implementacije bio je osmišljavanje rješenja problema na način da ne samo da radi, već da je u skladu s dobrim praksama. Za sve stilske odluke konzultirao se PEP 8. PEP 8 je stilski vodič za pisanje koda u Pythonu koji osigurava jedinstvene upute čije praćenje rezultira kodom koji je u duhu Pythona. Važnost praćenja stilskog vodiča možda se ne čini na prvi pogled velikom, međutim nekonzistentnost u kodu uzrokuje teže čitanje koda. Takvo što je posebno velik faktor u projektima čiji je broj linija koda, odnosno *LoC* relativno velik. U slučaju sustava koji je razvijen u sklopu diplomskog rada radi se o više od četrdeset razreda i više od pet tisuća linija koda. Konzistentnost pomaže bržem čitanju i snalaženju u kodu što povećava produktivnost, a i značajno olakšava promjene ili nadogradnje nakon je prošlo puno vremena (Gefroh, 2018). Zbog tih razloga tijekom razvoja nastojalo se maksimalno pridržavati PEP 8 vodiča uz određene iznimke.

Osim stilskog vodiča, primijenjivale su sve najbolje prakse programskog inženjerstva. Posebna pažnja u razvoju sustava posvećena je razrješavanju problematičnog koda. Tijekom pisanja koda prioritet je pisanje rješenja koje radi. Nuspojava tog pristupa ponekad je pojava problematičnog koda. Primjeri problema koje je pokazivao kod razvijenog sustava su duplikacija koda, dugačke metode, loše imenovanje, metode s previše argumenata te kršenje jedinstvene odgovornosti.
Jedan od principa koji pomažu očuvati kvalitetu koda u cijelom sustavu objašnjen je u knjizi Clean Code, a glasi da je potrebno prilikom svake izmjene nad kodom pogledati kod i napraviti *refactor* gdje god je moguće (Martin, 2008). Refactor je restrukturiranje postojećeg koda u svrhu poboljšanja kvalitete bez promjene vanjskog ponašanja. Refactor postiže bolju čitljivost, lakšu održivost, lakšu nadogradnju i poboljšava opću kvalitetu koda (Fowler, 2018). Primjenom malih refactora često tijekom životnog ciklusa razvoja osigurava se da kod kroz vrijeme ne postaje sve neuredniji, već u najgorem slučaju kompleksnost stagnira.

5.2.1. Primjeri refactora

Tehnike refactora najčešće korištene tijekom razvoja sustava su ekstrahiranje metode, ekstrahiranje varijable, micanje ponašanja u drugi razred, ekstrahiranje klase, pojednostavljivanje uvjeta te zamjena uvjeta uvođenjem polimorfizma. Najčešći problemi koje je refactor nastojao riješiti bili su predugačke metode, loše imenovanje, kršenje jedinstvene odgovornosti, duplikacija koda, kompleksni uvjeti te magične brojke.

Ovo potpoglavlje nastojat će prikazati primjere problematičnog koda te na koji način je razrješen. Prvi primjer odnosi se na dugačke metode. Dugačke metode teške su za čitati i obično sugeriraju na veći problem kršenja principa jedinstvene odgovornosti, engl. *single-responsibility principle*. Refaktoriranje dugačkih metoda znači ekstrahiranje koda u zasebne metode uz prikladno imenovanje novostvorenih metoda. Primjer dugačke metode moguće je pronaći u prvoj verziji razreda `ArticleFetcher` u metodi `commentsFetchCallback`.

```
def commentsFetchCallback(self, ch, method, properties, message):
  articleMsg = json.loads(message.decode('utf-8'))

  published = None
  if 'published' in articleMsg:
    published = articleMsg['published']

  if articleMsg['url'] == 'TESTURL':
    if published == None:
    published = 0
    print("Received␣test␣message.␣cannonicalURL={}␣published={}␣({})
        ".format(articleMsg['cannonicalURL'], time.strftime("%a,␣%d␣%
        b␣%Y␣%H:%M:%S␣%Z", time.localtime(int(published))), published
        ))
    self.channel.basic_ack(method.delivery_tag)
    return

  ignore_URLs = ('finirecepti.net.hr', 'www.tportal.hr/lifestyle/
      clanak/', 'specijali.rtl.hr/auto-po-mjeri/', 'www.novilist.hr/
      novosti/hrvatska/489603/')
  for iurl in ignore_URLs:
    if articleMsg['url'].find(iurl) > 0:
      print("Ignoring␣URL␣{}␣(substring␣{})...".format(articleMsg['
          url'], iurl))
```

```
    self.channel.basic_ack(method.delivery_tag)
                return

if len(articleMsg['url']) > 0:
  print("Fetching␣URL␣{}...".format(articleMsg['url']))
elif len(articleMsg['cannonicalURL']) > 0:
  print("Fetching␣cannonical␣URL␣{}...".format(articleMsg['
     cannonicalURL']))
else:
  print("Error!␣No␣URL␣nor␣CannonicalURL␣given.␣Ignoring")
self.channel.basic_ack(method.delivery_tag)
  return

    DELAY=600000

try:

  if articleMsg['commentType'] == 'facebook' and articleMsg['url'
     ].find('24sata.hr') < 0:
    try:
      article = ArticleFacebook(articleMsg['url'], cannonicalURL=
         articleMsg['cannonicalURL'], userAgent = self.user_agent,
          published = published)

            except requests.exceptions.ReadTimeout:
                print("Timeout␣connecting␣to␣URL.␣Requeuing␣
                   message...")
                self.channel.basic_ack(method.delivery_tag)
                self.channel.basic_publish(exchange='url-
                   exchange',
                        routing_key='articles',
                        properties=pika.BasicProperties(
        headers={'x-delay': DELAY},
                            delivery_mode = 2     # make
                               message persistent
                      ),
                        body=message.decode('utf-8'))
                return

            except UnexpectedCommentsFormat:
                print("Unexpected␣comments␣format.␣Probably␣we
                   ␣were␣temporarily␣blocked.␣Requeuing␣
                   message...")
```

```
                    self.channel.basic_ack(method.delivery_tag)
                    self.channel.basic_publish(exchange='url-
                        exchange',
                            routing_key='articles',
                            properties=pika.BasicProperties(
                                headers={'x-delay': DELAY},
                                delivery_mode = 2     # make
                                    message persistent
                            ),
                            body=message.decode('utf-8'))
                    return

elif articleMsg['commentType'] == 'vecernji':
                article = VecernjiComments(articleMsg['url'],
                    cannonicalURL=articleMsg['cannonicalURL'],
                    userAgent = self.user_agent, published =
                    published)

            elif articleMsg['commentType'] == '24sata' or
                articleMsg['url'].find('24sata.hr') >= 0:
  try:
    article = TwentyFourHoursComments(articleMsg['url'],
        cannonicalURL=articleMsg['cannonicalURL'], userAgent =
        self.user_agent, published = published)
                except:
                    print("Error␣fetching␣URL␣{}.␣Ignoring.".
                        format(articleMsg['url']))
                    self.channel.basic_ack(method.delivery_tag)
                    return

else:
  if len(articleMsg['url']) > 0:
                print("Unknown/unsupported␣commenting␣type␣{}␣for
                    ␣URL␣{}.␣Ignoring".format(articleMsg['
                    commentType'], articleMsg['url']))
                else:
    print("Unknown/unsupported␣commenting␣type␣{}␣for␣URL␣{}.␣
        Ignoring".format(articleMsg['commentType'], articleMsg['
        cannonicalURL']))

                self.channel.basic_ack(method.delivery_tag)
                return
```

```
            # Instead of the CommentFetcher persisting the article,
               it just calls save() on it
            # In the future, the CommentFetcher will not call save()
               , as this will be somebody else's responsibility.
            # article.save()

    status = self.mongo_collection.insert_one(article.getAsDict())
             print(status.inserted_id)

  except (ArticleFetchError, VLArticleFetchError) as articleNotFound
      :
             open("CommentFetcher.url", "a").write(articleMsg['url'
                ] + '\n')
             print("Article␣{}␣not␣found.␣Ignoring.".format(
                articleMsg['cannonicalURL']))
    self.channel.basic_ack(method.delivery_tag)
```

Ispis prikazuje `commentsFetchCallback` metodu razreda `ArticleFetcher`. Problem s metodom je što je dugačka i teško čitljiva, a krši i princip jedinstvene odgovornosti. Razlog tomu je dugačak **`if-elif-elif-else`** uvjet. Taj uvjet služi instanciranju odgovarajućeg Article razreda u ovisnosti o sadržaju poruke koju je dobio iz reda. Prikladna tehnika refactora u ovom slučaju bila je ekstrahiranje metode, ali i uvođenje polimorfizma. Stvoren je novi razred `ArticleFactory` s metodom `create_article` čija je jedina odgovornost instanciranje odgovarajućeg `Article` razreda. Međutim, ako se samo kod ekstrahira onda je neuredan kod premješten. To je napredak, međutim postoji potencijal napraviti bolje rješenje. Sljedeći korak je osigurati polimorfizam korištenjem refleksije u Pythonu. Naime, `ArticleFactory` će pokušati instancirati razred prema nazivu sustava komentiranja kojeg metoda `create_article` prima kao parametar. Primjerice, ako metoda primi `comment_type` s vrijednošću "Disqus" pokušat će instancirati `ArticleDisqus` razred.

```
class ArticleFactory:
    """ Factory class for creating objects out of classes which are
       derived from the abstract article class"""

    @staticmethod
    def create_article(comment_type, url, cannonical_url=None,
       user_agent=None, date_published=None):

        """ Creates an article object which corrresponds to the
           comment_type"""
```

```
comment_type = comment_type.capitalize()

try:
    module_name = "article.Article" + comment_type
    class_name = "Article" + comment_type

    article_class = getattr(importlib.import_module(
       module_name),
                            class_name)
    article = article_class(url, cannonical_url, user_agent,
        date_published)
    return article
except ModuleNotFoundError:
    raise NotYetSupportedError
```

Rješenje prikazano u ispisu lakše je čitljivo, a predstavlja i poboljšanje u odnosu na izvorni kod. Naime, ovakvo rješenje omogućava nadogradnju bez promjene. Kada je potrebno dodati novi sustav komentiranja, `ArticleFactory` ne mora se mijenjati.

```
try:
    article = CommentFactory.create_comment(articleMsg['url'
       ], articleMsg['cannonicalURL'], self.user_agent,
       published)
except requests.exceptions.ReadTimeout:
    # Handle ReadTimeout exception
    print "Timeout␣connecting␣to␣URL.␣Requeuing␣..."
    requeue_msg(method.delivery_tag, message)
    return
except UnexpectedCommentsFormat:
    # Handle UnexpectedCommentsFormat exception
    print "Unexcepted␣comments␣format.␣Temporary␣block␣
       probable.␣Requeuing␣..."
    requeue_msg(method.delivery_tag, message)
    return
except:
    # Handle base exception
    print("Error␣fetching␣URL␣{}.␣Ignoring.".format(
       articleMsg['url']))
    self.channel.basic_ack(method.delivery_tag)
    return
```

Ispis pokazuje odsječak koji instancira `Article` razred u metodi `commentsFetchCallback` razreda `ArticleFetcher`. Vidljiva je znatna razlika u čitljivosti.

Primjera dugačkih metoda koje nemaju jedinstvenu odgovornost bilo je mnogo. Drugi

primjer je `__init__` metoda svih `Article` razreda koja je sadržavala logiku parsiranja datuma, dohvata i parsiranja komentara te spremanja HTML stranice. S obzirom na ulogu razreda, `__init__` metoda ostavljena je da radi sve navedene akcije, međutim poboljšana je na način da ih obavlja pozivima drugim metodama ili razredima. Na taj način je metoda djelomično rasterećena te lakša za čitati. Također, u slučaju nadogradnje manja je šansa da će se temeljni razred poput `ArticleFacebook` morati promijeniti.

Drugi problem koji je rezultat brzog pisanja koda je loše imenovanje varijabli i metoda. Loše imenovanje onemogućuje programeru da čitanjem koda na prvi pogled zaključi značaj varijable ili metode, odnosno potrebno je potrošiti više energije kako bi se shvatilo njihovo značenje. Takvo što oduzima od produktivnosti programera te usporava napredak projekta. Pri refactoru, odnosno preimenovanjima, referenciralo se na knjigu Clean Code. Preimenovanje je izvedeno u skladu s nekoliko pravila, a najbitnija su da imena varijabli i metoda vjerno reflektiraju njihovu ulogu, zatim da nema redudancije u imenovanju te da se izbjegavaju općenita imena poput **`list`** ili `data`. Primjer lošeg imena je varijabla `data` koja sadrži učitane članke iz baze podataka. Bolje ime bilo bi `articles`.
Tijekom pisanja sustava jedan od prvih refactora bio je preimenovanje razreda. Tako su razredi `FacebookComments`, `VecernjiComments`, `TwentyFourHoursComments` i `CommentFetcher` postali `ArticleFacebook`, `ArticleVecernji`, `Article24sata` te `ArticleFetcher`. Postoje dva razloga koja su uvjetovala preimenovanje. Prvi razlog je što se radi o spremanju članaka, a ne isključivo komentara. Zbog toga je je `Comments` sufiks zamijenjen s `Article`. Drugi faktor je činjenica da PEP 8 predlaže da se srodni razredi prefiksiraju zajedničkim dijelom naziva. Rezultat toga je da se u sučelju za rad prikazuju skupa, a i pregledom direktorija bilo kojim drugim alatom bit će vizualno lakše prepoznati koji razredi su srodni.

Kršenje jedinstvene odgovornosti ozbiljan je problem zbog toga što otežava praćenje logike koda. Na primjeru iz razvijanog sustava, ime razreda `ArticleFacebook` sugerira da je kod isključivo vezan uz Facebook comments plugin. Međutim, u tom razredu nalazi se i dugačak odsječak koda koji se bavi parsiranjem datuma sa stranice. U ovom slučaju slično je rješenje kao i u primjeru razrješavanja dugačkih metoda, odnosno izdvajanje koda u razred te uvođenje polimorfizma.

```
if self.published is None:
```

```
if url.find("dnevnik.hr/") > 0:
dates = re.findall('"datePublished":"([0-9]+)-([0-9]+)-([0-9]+)T
    ([0-9]+):([0-9]+):([0-9]+)', self.webPage)
  if len(dates) == 1 and len(dates[0]) == 6:
    self.published = datetime.datetime(int(dates[0][0]),int(dates
        [0][1]),int(dates[0][2]),int(dates[0][3]),int(dates[0][4]))
        .timestamp()
  else:
    dates = re.findall('([0-9]{2})\. ([^.]+)\. ([0-9]{4})\. @
        ([0-9]{2}):([0-9]{2})', self.webPage)
                if len(dates) == 1 and len(dates[0]) == 5:
      if dates[0][1][:4] == 'sije': mjesec = 1
      elif dates[0][1][:4] == 'velj': mjesec = 2
      elif dates[0][1][:4] == 'ozuj': mjesec = 3
      elif dates[0][1][:4] == 'trav': mjesec = 4
      elif dates[0][1][:4] == 'svib': mjesec = 5
      elif dates[0][1][:4] == 'lipa': mjesec = 6
      elif dates[0][1][:4] == 'srpa': mjesec = 7
      elif dates[0][1][:4] == 'kolo': mjesec = 8
      elif dates[0][1][:4] == 'ruja': mjesec = 9
      elif dates[0][1][:4] == 'list': mjesec = 10
      elif dates[0][1][:4] == 'stud': mjesec = 11
      elif dates[0][1][:4] == 'pros': mjesec = 12

      self.published = datetime.datetime(int(dates[0][2]),mjesec,
          int(dates[0][0]),int(dates[0][3]),int(dates[0][4])).
          timestamp()
               else:
                   raise ArticleDateNotFound()

elif url.find("dnevno.hr/") > 0:
  dates = re.findall('<meta property="article:published_time"
      content="([0-9]+)-([0-9]+)-([0-9]+)T([0-9]+):([0-9]+)', self.
      webPage)
  if len(dates) == 1 and len(dates[0]) == 5:
    self.published = datetime.datetime(int(dates[0][0]),int(dates
        [0][1]),int(dates[0][2]),int(dates[0][3]),int(dates[0][4]))
        .timestamp()
  else:
    dates = re.findall('<meta property="article:modified_time"
        content="([0-9]+)-([0-9]+)-([0-9]+)T([0-9]+):([0-9]+)',
        self.webPage)
                if len(dates) == 1 and len(dates[0]) == 5:
```

```
        self.published = datetime.datetime(int(dates[0][0]),int(
            dates[0][1]),int(dates[0][2]),int(dates[0][3]),int(dates
            [0][4])).timestamp()
      else:
                            raise ArticleDateNotFound()

elif url.find("index.hr/") > 0:
  dates = re.findall('"datePublished":"([0-9]+)-([0-9]+)-([0-9]+)"
      ', self.webPage)
  if len(dates) == 1 and len(dates[0]) == 3:
    self.published = datetime.datetime(int(dates[0][0]),int(dates
        [0][1]),int(dates[0][2]),0,0).timestamp() + 12 * 3600
  else:
    open('FacebookComments.error').write(self.webPage)
    raise ArticleDateNotFound()

elif url.find("jutarnji.hr/") > 0:
  dates = re.findall("pubdate:␣'([0-9]+)-([0-9]+)-([0-9]+)T
      ([0-9]+):([0-9]+)\+00:00'", self.webPage)
  if len(dates) == 1 and len(dates[0]) == 5:
    self.published = datetime.datetime(int(dates[0][0]),int(dates
        [0][1]),int(dates[0][2]),int(dates[0][3]),int(dates[0][4]))
        .timestamp()
  else:
    raise ArticleDateNotFound()

elif url.find("korona.net.hr/") > 0:
  dates = re.findall('<meta␣property="article:published_time"␣
      content="([0-9]+)-([0-9]+)-([0-9]+)T([0-9]+):([0-9]+)', self.
      webPage)
  if len(dates) == 1 and len(dates[0]) == 5:
    self.published = datetime.datetime(int(dates[0][0]),int(dates
        [0][1]),int(dates[0][2]),int(dates[0][3]),int(dates[0][4]))
        .timestamp()
  else:
    raise ArticleDateNotFound()

elif url.find("net.hr/") > 0:
  dates = re.findall('"pubdate":"([0-9]+)-([0-9]+)-([0-9]+)T
      ([0-9]+):([0-9]+):([0-9]+)\+([0-9]+)"', self.webPage)
  if len(dates) == 1 and len(dates[0]) == 7:
    self.published = datetime.datetime(int(dates[0][0]),int(dates
        [0][1]),int(dates[0][2]),int(dates[0][3]),int(dates[0][4]),
```

```
        int(dates[0][5]),int(dates[0][6])).timestamp()
    else:
      raise ArticleDateNotFound()

  elif url.find("novilist.hr/") > 0:
    dates = re.findall('"datePublished":"([0-9]+)-([0-9]+)-([0-9]+)T
       ([0-9]+):([0-9]+):([0-9]+)\+01:00"', self.webPage)
    if len(dates) == 1 and len(dates[0]) == 6:
      self.published = datetime.datetime(int(dates[0][0]),int(dates
         [0][1]),int(dates[0][2]),int(dates[0][3]),int(dates[0][4]),
         int(dates[0][5])).timestamp()
    else:
      raise ArticleDateNotFound()
  ...
```

Ispis prikazuje dio kompleksne logike parsiranja datuma koja zauzima velik dio koda u `ArticleFacebook` razredu. Kako bi se refactor proveo potrebno je uvesti novi razred `DateParser`, odnosno niz razreda, koji sadrže logiku parsiranja za svaku od pojedinih stranica. Osim toga, potrebno je osigurati da se `ArticleFacebook` ne mora mijenjati u slučaju dodavanja novih stranica. To se postiže uvođenjem polimorfizma pomoću refleksije.

Nova implementacija `ArticleFacebook` sadrži jednostavan poziv kojim se dohvaća instanca `DateParser`.

```
  date_parser = get_date_parser(get_host_from_url(url))
      pub = date_parser.parse_date(self.web_page)
      if pub is None:
          self.log.error("Failed␣parsing␣date␣published.")
          raise DateNotFoundError("Failed␣parsing␣date␣published.␣")

      self.published = date_parser.parse_date(self.web_page)
```

Ispis prikazuje odsječak koda koji dohvaća instancu odgovarajućeg `DateParser` razreda za pojedinu stranicu. Bitno je napomenuti da zbog razlika između stranica, svaka stranica ima vlastiti DateParser te se instanciranje vrši korištenjem refleksije po uzoru na kod iz `ArticleFactory`.

```
def get_date_parser(host_name):
    class_name = "DateParser" + host_name.capitalize()
    module_name = "article.DateParser"
    date_parser_class = getattr(importlib.import_module(module_name)
       , class_name)
```

```
    date_parser = date_parser_class()
    return date_parser
```

Ispis prikazuje metodu `get_date_parser` koja instancira i vraća odgovarajući `DateParser` razred.

Jedan problem koji se nije često pojavio tijekom pisanja koda je duplikacija koda. To je problem koji uzrokuje krhkost koda zbog toga što kada je kod dupliciran, promjene moraju biti uvišestručene i potencijal pogreške značajno je veći nego li u slučaju da se mora promijeniti kod na samo jednom mjestu. Deduplikacija je provedena izdvajanjem koda u metode ili razrede. Primjer duplikacije koda u ranoj verziji sustava odnosi se na vraćanje poruke u red prilikom nemogućnosti obrade zahtjeva.

```
try:
  article = FacebookComments(articleMsg['url'], cannonicalURL=
      articleMsg['cannonicalURL'], userAgent = self.user_agent,
      published = published)

except requests.exceptions.ReadTimeout:
  print("Timeout␣connecting␣to␣URL.␣Requeuing␣message...")
      self.channel.basic_ack(method.delivery_tag)
      self.channel.basic_publish(exchange='url-exchange',
               routing_key='articles',
               properties=pika.BasicProperties(
                   headers={'x-delay': DELAY},
                   delivery_mode = 2     # make message persistent
               ),
               body=message.decode('utf-8'))
      return

except UnexpectedCommentsFormat:
      print("Unexpected␣comments␣format.␣Probably␣we␣were␣
         temporarily␣blocked.␣Requeuing␣message...")
      self.channel.basic_ack(method.delivery_tag)
      self.channel.basic_publish(exchange='url-exchange',
               routing_key='articles',
               properties=pika.BasicProperties(
                   headers={'x-delay': DELAY},
                   delivery_mode = 2     # make message persistent
               ),
               body=message.decode('utf-8'))
      return
```

Ispis prikazuje duplicirani kod za vraćanje poruke u red kako bi se ponovila obrada članka. Ovaj slučaj duplikacije vrlo je blag, no bez obzira na ozbiljnost može uzrokovati neočekivano ponašanje ako se promjeni jedan odsječak koda, a ne i njegov duplikat. Najprikladnije rješenje u ovom slučaju je izdvajanje logike ponovnog pokušaja u zasebnu metodu.

```
def requeue_msg(self, delivery_tag, message):
  self.channel.basic_ack(method.delivery_tag)
  self.channel.basic_publish(exchange='url-exchange',
              routing_key='articles',
              properties=pika.BasicProperties(
                headers={'x-delay': DELAY},
                                    delivery_mode=2  # make
                                        message persistent
                                  ),
                                  body=message.decode('utf-8'))
```

Ispis prikazuje metodu novonastalu `requeue_msg` metodu koja služi vraćanju poruke u red.

```
try:
  article = ArticleFactory.create_article(articleMsg['commentType'],
       articleMsg['url'], articleMsg['cannonicalURL'], self.
      user_agent, published)
except requests.exceptions.ReadTimeout:
  # Handle ReadTimeout exception
  print("Timeout␣connecting␣to␣URL.␣Requeuing␣...")
  requeue_msg(method.delivery_tag, message)
  return

except UnexpectedCommentsFormat:
  # Handle UnexpectedCommentsFormat exception
  print("Unexcepted␣comments␣format.␣Temporary␣block␣probable.␣
      Requeuing␣...")
  requeue_msg(method.delivery_tag, message)
  return
```

Ispis prikazuje novo stanje u `ArticleFetcher` koje više nema duplikacije koda.

6. Prikaz rada sustava

7. Zaključak

Zaključak.

LITERATURA

Robert Fowler. *Refactoring: Improving the Design of Existing Code*. 2018.

Joseph Gefroh. Why consistency is one of the top indicators of good code, Rujan 2018. URL https://jgefroh.medium.com/why-consistency-is-one-of-the-top-indicators-of-good-code-352ba5d62 [Online; pristup ostvaren 15.6.2021].

Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Kolovoz 2008.

MongoDB. When to use a nosql database, unknown 2021. URL https://www.mongodb.com/nosql-explained/when-to-use-nosql. [Online; pristup ostvaren 15.6.2021].

ScraperAPI. 5 tips for web scraping without getting blocked or blacklisted, Prosinac 2019. URL https://www.scraperapi.com/blog/5-tips-for-web-scraping/. [Online; pristup ostvaren 15.6.2021].

Biblioteka za dohvat novinskog članka s portala i pridruženih komentara nadruštvenim mrežama

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.

Title

Abstract

Abstract.

Keywords: Keywords.