SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2652

Biblioteka za dohvat novinskog članka s portala i pridruženih komentara na društvenim 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 [4]. Radi se o golemom sadržaju i količini informacija koje su mnoštvu na dohvat ruke, čak više od polovici stanovništva svijeta [6]. U tu golemu količinu sadržaja 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 biblioteku, odnosno 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 vanjske sustave za komentiranje poput *Disqus* ili dodatka *Facebook comments*. Drugi portali razvijaju vlastite. Bez obzira na sustave komentiranja, sami portali međusobno se razlikuju u strukturi izvornog koda stranice. Predstavljena heterogenost sustava komentiranja, ali i specifičnost pojedinih portala onemogućava jednostavnu 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, mijenja se i struktura HTML koda portala. Promjene mogu biti uzrokovane jednostavnim nadogradnjama na sustav komentiranja koje se ne očituju izvana, ali mogu biti i promjene koje onemogućuju ispravan rad razvijane biblioteke, odnosno 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 robusan i znati rukovati promjenama te se prilagoditi kad god je moguće.

Svrha ovog rada je izgradnja biblioteke, odnosno lako nadogradivog i robusnog sustava za prikupljanje komentara s posebnom pažnjom vođenom o učenju i primjeni najboljih praksi 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, koja je arhitektura sustava te na koji način je implementacija izvedena u skladu s najboljim praksama.

Rad je strukturiran na sljedeći način. Prvo je objašnjena teorijska podloga sustava, odnosno rezultat primjene postupka reverznog inženjerstva nad sustavima komentiranja. Teorijska podloga uključuje objašnjenje načina rada dvaju izabranih polularnih sustava komentiranja, izbjegavanje detekcije tijekom rada te koje promjene su moguće u sustavima komentiranja na koje sustav mora paziti i reagirati. Slijedi poglavlje u kojem je objašnjena arhitektura sustava uz objašnjenje uloga pojedinih modula i razreda te prikaza najbitnijih dijelova sustava. U poglavlju o arhitekturi smješteno je i potpoglavlje o klasifikaciji koje daje terminološki kontekst razvijanom sustavu. Sljedeće poglavlje bavi se tehnologijama i metodologijom implementacije te implementacijskim detaljima koji su nužni za razumijevanje problema koje je rad riješio. Tu će biti objašnjeno koje su tehnologije odabrane za implementaciju i zašto, a bit će i objašnjeno na koji način je provedena implementacija u skladu s najboljim praksama. Nakon toga slijede potpoglavlja u kojima su prikazani implementacijski detalji kao nadogradnja na prethodna potpoglavlja o tehnologijama i metodologiji implementacije. Zadnje poglavlje služi kao demonstracija rada sustava.

2. Analiza polularnih sustava za komentiranje

Ovo poglavlje dat će detaljan pregled rezultata postupka reverznog inženjerstva nad dodatkom *Facebook comments* te sustavom za komentiranje *Disqus*. Rezultati prikazani u ovom poglavlju predstavljaju temelj nad kojim će se razvijati sustav za prikupljanje komentara. Uz to, zadnja dva potpoglavlja dat će teorijsku podlogu izbjegavanju detekcije te mehanizmu rukovanja promjenama implementiranog u sustavu.

2.1. Dodatak *Facebook comments*

Dodatak *Facebook comments* 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 moraju biti ispunjeni. Filtriranjem mrežnog prometa pomoću razvojne konzole u pregledniku pri učitavanju stranice lako je izdvojiti GET zahtjev čiji oblik prikazan u ispisu 2.1 daje do znanja da bi mogao biti odgovoran za učitavanje komentara. Odgovor na zahtjev vraća HTML kod koji 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.

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

Ispis 2.1: URL GET zahtjeva za inicijalni skup komentara

Kako bi se zahtjev mogao poslati potrebno je `href` parametar postaviti na vrijednost URL-a članka. Zaglavlja za početak nije potrebno postavljati, međutim pri kraju pisanja rada dogodila se promjena u dodatku *Facebook comments* koja zahtijeva dodavanje zaglavlja. Konkretno, potrebno je u zaglavlju uključiti `Cookie` ključ s vrijednošću `c_user=1`. Više o vrijednostima zaglavlja bit će u poglavlju 2.4.

S obzirom na to da zahtjev prikazan u ispisu 2.1 dohvaća samo prvih nekoliko komentara, potrebno je pronaći način kako dohvatiti sve komentare povezane s člankom. Dodatni komentari se u slučaju dodatka *Facebook comments* učitavaju pritiskom na gumb ‘Učitaj još komentara‘. Gumb se može pritiskati sve dok nije učitan puni skup komentara. Potrebno je analizirati zahtjev za dodatnim komentarima te iskoristiti ga kako bi se u jednom zahtjevu ili više njih dohvatili svi komentari povezani s člankom.

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, odnosno prethodni skup. Zahtjev kojeg šalje preglednik pritiskom na gumb je oblika prikazanog u ispisu 2.2. Parametar *target_id* potrebno je zamijeniti i njegova uloga bit će objašnjena kasnije.

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

Ispis 2.2: URL POST zahtjeva za dohvat dodatnog skupa komentara

POST zahtjev kojeg preglednik šalje na URL iz ispisa 2.2 u tijelu zahtjeva sadrži više podataka od kojih nisu svi potrebni kako bi se zahtjev ispunio. U ispisu 2.3 prikazano je cijelo tijelo ulovljenog zahtjeva u mrežnoj konzoli preglednika. Vrijednost `app_id` je identifikator stranice, odnosno novinskog portala kojeg promatramo. Vrijednost `after_cursor` služi slijednom dohvatu komentara, odnosno govori poslužitelju od kojeg komentara da učita sljedećih nekoliko. Za ostale vrijednosti otkrivanje značenja zahtijeva trud koji neće rezultirati boljim sustavom zbog toga što se njihove vrijednosti mogu izostaviti. Analizom je ustanovljeno da su samo dvije vrijednosti u tijelu zahtjeva potrebne kako bi zahtjev vratio sve komentare povezane s člankom. To su `__a` te `limit` koji su vidljivi u ispisu 2.3. Zbog toga nije potrebno istraživati značenja i vrijednosti ostalih elemenata tijela zahtjeva.

```
app_id:216060115209819
after_cursor:AQHRDjs097OAlV2dZ-
    oOgl93OQq6FIwSN78BBI53A7pyXLsdgEmibF035czYklxO9B8Z9uDGnA0HdI0bhpgbfIHDeg

limit:10
iframe_referer:https://www.index.hr/
__user:0
__a:1
__dyn:7xe6EgU4e3W3mbG2KmhwRwqo98nwgUbErxW5EyewSwMwyzEdU5i3K1bwOw-
    wpUe8hwem0nCq1ewbWbwmo62782Cwooa84u0gO0AbAy85iaw4Ugao2mw9i0D83rw9O0RE5a1qw8W0hC

__csr:
__req:3
__beoa:0
__pc:PHASED:plugin_feedback_pkg
dpr:1
__ccg:EXCELLENT
__rev:1003118246
__s:::u2bgql
__hsi:6907282813311570284
__comet_req:0
locale:hr_HR
__sp:1
```

Ispis 2.3: Tijelo POST zahtjeva za dodatan skup komentara

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.

Nakon što je ustanovljeno koje vrijednosti tijelo zahtjeva mora imati kako bi se zahtjev ispunio, potrebno je otkriti vrijednost parametra *target_id*. URL iz ispisa 2.2 sadrži parametar *target_id* koji je u stvarnosti identifikator članka u kontekstu dodatka *Facebook comments*. Vrijednost *target_id* parametra moguće je dobiti iz HTML koda početnog GET zahtjeva na URL prikazan u ispisu 2.1. Unutar HTML koda postoji JSON struktura, a unutar nje nalazi se ključ `targetFBID`. Ispis 2.4 prikazuje malen dio rezultata početnog GET zahtjeva u kojem je vidljiv identifikator članka. Ostatak vrijednosti u ispisu prikazane su samo radi ilustracije oblika podataka.

```
.. "meta":{"targetFBID":"3329164850466353","href":"https:\/\/www.
    index.hr\/clanak.aspx?id=2230197","userID":"0","actorsOptIn":[
    true],"actors":[{"id":"0","name":"","thumbSrc":"https:\/\/static.
    xx.fbcdn.net\/rsrc.php\/v1\/yi\/r\/odA9sNLrE86.jpg","uri":"","
    type":"user"}],"totalCount":19, ...
```

Ispis 2.4: Dio JSON strukture unutar rezultata inicijalnog GET zahtjeva

Vrijednost pod ključem `targetFBID` upravo je identifikator članka koji je potreban za slanje asinkronog POST zahtjeva. Kako bi se parsirao HTML kod i dobio identifikator članka, potreban je regularni izraz prikazan u ispisu 2.5. Regularni izraz traži niz znakova ‘targetFBID‘ ali izdvaja vrijednost nakon dvotočke što predstavlja sam identifikator.

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

Ispis 2.5: Regularni izraz za parsiranje identifikatora članka

Vrijednost koju traženje po regularnom izrazu iz ispisa 2.5 vraća je identifikator članka. Nakon što je parsiran identifikator članka, svi preduvjeti za slanje POST zahtjeva na URL iz ispisa 2.2 su ispunjeni. Rezultat zahtjeva je JSON podatkovna struktura koja sadrži sve komentare i informacije o autorima. JSON struktura sadrži ključeve `__ar` , `payload` te `idMap`. Vrijednost pod ključem `__ar` je zanemariva. Vrijednost pod ključem `payload` sadrži `totalCount` koji govori koliko je komentara dohvaćeno te `commentIDs` koji je polje identifikatora komentara. Te informacije nisu bitne za razvoj sustava. Jedine bitne informacije za rad sustava su pod `idMap` ključem. Oblik vrijednosti pod ključem `idMap` prikazan je u ispisu 2.6.

```
{
   "108967884097341":{
      "id":"108967884097341",
      "name":"Ja sam \u0106ilim Ribi\u0107, car svih Uhljeba",
      "thumbSrc":"https:\/\/scontent.fzag4-1.fna.fbcdn.net\/v\/t1
          .6435-1\/cp0\/p48x48\/92466021
          _108968540763942_5464572798636654592_n.jpg?_nc_cat=111&ccb
          =1-3&_nc_sid=dbb9e7&_nc_ohc=Hw8Ut-nCerQAX8t5aW5&_nc_ht=
          scontent.fzag4-1.fna&tp=27&oh=
          e92006f34f22599cb95a740dffc419f3&oe=60D0E0DB",
      "uri":"https:\/\/www.facebook.com\/Ja-sam-\u0106ilim-Ribi\
          u0107-car-svih-Uhljeba-108967884097341\/",
      "isVerified":false,
      "type":"user"
   },
   "3329164850466353":{
      "id":"3329164850466353",
      "name":"",
```

```
      "uri":"https:\/\/www.index.hr\/clanak.aspx?id=2230197",
      "type":"ogobject"
   },
   "3329164850466353_3329184483797723":{
      "id":"3329164850466353_3329184483797723",
      "authorID":"108967884097341",
      "body":{
         "text":"Pobijedili su oni koronu ba\u0161 kao i HDZ ovdje."
      },
      "ranges":[

      ],
      "timestamp":{
         "time":1605288053,
         "text":"13. stu 2020. 09:20"
      },
      "targetID":"3329164850466353",
      "ogURL":"https:\/\/www.index.hr\/clanak.aspx?id=2230197",
      "likeCount":4,
      "hasLiked":false,
      "canLike":false,
      "canEdit":false,
      "hidden":false,
      "highlightedWords":[

      ],
      "reportURI":"\/report\/dispatch\/?content_type=98&cid
         =3329184483797723&rid=108967884097341&comment_id
         =3329184483797723",
      "spamCount":1,
      "canEmbed":true,
      "type":"comment"
   },
   "100042285277968":{
      "id":"100042285277968",
      "name":"Sumpor Cink",
      "thumbSrc":"https:\/\/scontent.fzag4-1.fna.fbcdn.net\/v\/t1
         .30497-1\/cp0\/c14.0.48.48a\/p48x48\/84241059
         _189132118950875_4138507100605120512_n.jpg?_nc_cat=1&ccb
         =1-3&_nc_sid=dbb9e7&_nc_ohc=_Jclsj48zTcAX9oG1IE&_nc_ht=
         scontent.fzag4-1.fna&tp=27&oh=63
         e8d39095f0a12502abd73f09083514&oe=60D137E6",
      "uri":"https:\/\/www.facebook.com\/sumpor.cink.5",
      "isVerified":false,
      "type":"user"
   }
}
```


Ispis 2.6: Oblik `idMap` vrijednosti

Ključ `idMap` prikazan u ispisu 2.6 sadrži dvije zanimljive vrste unosa, a to su korisnici i komentari. Ključevi pod kojima se nalaze unosi upravo su njihovi identifikatori, odnosno identifikator korisnika ili identifikator komentara. Postoji i treća vrsta objekta čiji je `type` jednak ‘ogobject‘ koji nije interesantan u sklopu rada. Svaka vrijednost pod vršnim ključevima sadrži ključ `type` koji govori predstavlja li trenutni unos informacije o korisniku ili informacije o komentaru. Drugi način razlikovanja je promatranje ključa pod kojim je unos spremljen. U slučaju da se radi o ključu čije su vrijednosti odvojene znakom _, tada je unos komentar. Vrijednost s lijeve strane predstavlja identifikator članka, a vrijednost s desne strane identifikator komentara u članku. To svojstvo je zanimljivo zbog toga što se može iskoristiti za dohvat identifikatora članka u slučaju da se sustav komentiranja promijeni te više nije moguće parsirati identifikator regularnim izrazom iz ispisa 2.5. Drugo vrlo bitno svojstvo `idMap` zapisa je što komentari sadrže

tekst te identifikator autora. Tako su moguće i obrade da se bilježi što je koji autor napisao kroz cijelu povijest aktivnosti, a sve informacije o autoru dostupne su unutar istog `idMap` zapisa. Takav JSON spreman je za obradu i spremanje u bazu podataka s preuzetim člankom.

2.2. *Disqus*

Sustav komentiranja *Disqus* u širokoj je uporabi u svijetu i zato je interesantan za analizu. Skoro četvrtina milijuna stranica u svijetu koristi *Disqus* kao sustav komentiranja [3]. 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 potrebno je prvo pronaći identifikator niza komentara, odnosno *thread id*. To je sličan korak kao i onaj objašnjen u poglavlju o dodatku *Facebook comments*, samo što se ne radi o identifikatoru članka već identifikatoru samog niza komentara. Identifikator niza komentara moguće je pronaći u rezultatu GET zahtjeva na URL prikazan u ispisu 2.7. Navedeni GET zahtjev služi istoj svrsi kao i inicijalni GET zahtjev kod dodatka *Facebook comments*, odnosno dohvaća početni skup komentara koji se prikazuju pri učitavanju članka u pregledniku.

```
https://disqus.com/embed/comments/?f=news_portal_name&t_u=article_ur
```

Ispis 2.7: URL za GET zahtjev kojim se dohvaća inicijalni skup komentara

Parametar `f` s *placeholder* 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 parsirati 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 slanja GET zahtjeva na URL iz ispisa 2.7 prikazan je u ispisu 2.8 i sadrži identifikator niza komentara pod `posts` ključem kojeg je moguće ekstrahirati regularnim izrazom prikazanim u ispisu 2.9. Regularni izraz traži niz znakova 'thread' nakon kojeg slijede dvotočka te niz brojeva koji je identifikator niza. Također, regularni izraz izdvaja sam identifikator niza. U ispisu 2.8 vidljiv je `cursor` ključ koji će kasnije biti ključan za slijedni dohvat komentara. Njegova uloga bit će objašnjena kasnije. Osim toga, u ispisu 2.8 vidljiv je `response` ključ koji sadrži početne komentare i informacije o njima. Taj dio nije prikazan u potpunosti jer nije korišten u implementaciji, odnosno rezultat početnog GET zahtjeva služi samo kako bi se mogao parsirati identifikator niza komentara.

```
{
   "cursor":{
      "hasPrev":false,
      "prev":null,
      "total":159,
      "hasNext":true,
      "next":"1:0:0"
   },
   "code":0,
   "response":{
      "lastModified":1618782958,
      "posts":[
         {
            "editableUntil":"2021-04-23T15:18:17",
            "dislikes":0,
            "thread":"8480376159",
            "numReports":0,
            "likes":15
         }
         ..
      ]
      ...
   }
   ...
}
```

Ispis 2.8: Prikaz dijela rezultata inicijalnog GET zahtjeva

```
"thread":"([0-9]+)"
```

Ispis 2.9: Regularni izraz za parsiranje identifikatora niza komentara

Kada je parsiran identifikator niza komentara, moguće je krenuti u slijedni dohvat komentara. Kao i u slučaju dodatka *Facebook comments*, postoji drugi zahtjev koji služi slijednom dohvatu novih skupova komentara kao proširenja inicijalnom, odnosno prethodno učitanom skupu. Zahtjev se šalje pritiskom na gumb 'Učitaj još komentara' čija je funkcija istovjetna onoj objašnjenoj u poglavlju o analizi dodatka *Facebook comments*. Sljedeći skup komentara može se dohvatiti slanjem GET zahtjeva na URL čiji je oblik bez konkretnih vrijednosti parametara prikazan u ispisu 2.10.

```
https://disqus.com/api/3.0/threads/listPostsThreaded?limit=100&
   thread={}&cursor={}&api_key={}
```

Ispis 2.10: URL za GET zahtjev za dohvat komentara

Kako bi se zahtjev na URL iz ispisa 2.10 ispunio, potrebno je popuniti vrijednosti parametara koje su u ispisu obilježene praznim vitičastim zagradama. 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 kojeg 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`, i tako dalje. Naime, *Disqus* se tu razlikuje od dodatka *Facebook comments*. Parametar `limit` je kod *Disqus* sustava

ograničen na maksimalnu vrijednost sto, dok je kod dodatka *Facebook comments* ograničenje bilo pretpostavljeno sto, no dozvoljen je bilo koji broj. U sustavu *Disqus* bilo koja vrijednost ograničenja iznad sto rezultirat će neispunjenim zahtjevom. Zbog toga nije moguće postavljanjem visokog `limit` parametra dohvatiti sve komentare u jednom pozivu kao što je slučaj s dodatkom *Facebook comments*. Kako bi se dohvatili svi komentari, potrebno je iskoristiti podatke koji su vraćeni u JSON-u kao rezultat zahtjeva te tako omogućiti slijedni dohvat komentara koji u stvarnosti simulira korisnikove upite za novim komentarima na članku pritiskom gumba. U rezultatu GET zahtjeva na URL iz ispisa 2.10 pod ključem `cursor` nalazi se ugniježđena JSON struktura koja sadrži ključeve: `hasPrev`, `prev`, `total`, `hasNext` te `next`. Ispis 2.11 prikazuje JSON rezultat na zahtjev iz ispisa 2.10. U rezultatu bitan je `cursor` ključ.

```
{
    "cursor": {
        "prev": null,
        "hasNext": true,
        "next": "1:0:0",
        "hasPrev": false,
        "total": 2,
        "id": "1:0:0",
        "more": true
    },
    ...
}
```

Ispis 2.11: Prikaz vrijednosti `cursora` u odgovoru na POST zahtjev za komentare

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 sljedeća stranica s komentarima i preuzimanje komentara je završeno. Ispisi 2.12, 2.13 te 2.14 prikazuju početak dijela JSON-a nakon `cursor` zapisa iz ispisa 2.11 koji sadrži informacije o komentarima i autorima. Unutar `response` ključa vidljivog u ispisu 2.12 nalaze se komentari. Svaki od komentara ima vrijednosti poput `dislikes`, `editableUntil`, `likes` i drugih koje nisu od posebne koristi. U ispisu 2.14 osobito je vidljivo da ima mnogo informacija koje nisu bitne za osnovni rad sustava, no možda bi bile korisne u nadogradnjama. Primjer takvih informacija su `isSpam`, `isDeletedByAuthor`, `isApproved`, `canVote` i drugi. Takve informacije nisu potrebne jer samo dodaju kontekstu preuzetih komentara, no za trenutne potrebe sustava stvarale bi nepotreban pritisak na obradi i spremanju. Ono što je bitno da bi se komentari mogli koristiti u analizama su sami tekstovi komentara te osnovne informacije o autoru. Informacije o autoru detaljno su izložene u ispisu 2.12 gdje je pod ključem `author` prikazan niz informacija poput korisničkog imena, prikaznog imena, razine reputacije, URL-a profila korisnika, lokacije, datuma uključivanja na forum te samog identifikatora korisnika. Sam tekst komentara sadržan je ispod ključa `message`. U ispisu 2.12 vidljivo je da prvi komentar ima prazan niz kao poruku, dok u ispisu 2.14 postoji tekst komentara koji nije prazan. U ispisu 2.14 vidljiv je početak novog komentara s formatom prikazanim u ispisu 2.12. Ispis 2.13 prikazuje medijski sadržaj komentara koji se ne bi mogao prikazati kao tekst komentara. U to spadaju poveznice, slike i videozapisi. Ti podaci nisu bitni za početnu implementaciju sustava, međutim mogli bi biti korisni u analizama tipova sadržaja koje korisnici objavljuju uz komentare.

```
"response": [
        {
            "editableUntil": "2021-04-23T15:18:17",
            "dislikes": 0,
            "thread": "8480376159",
            "numReports": 0,
            "likes": 15,
            "message": ``,
            "id": "5347473378",
            "createdAt": "2021-04-16T15:18:17",
            "author": {
                "username": "modernmethod-
                   ca38cba2e6c8d5ffb15a3b87e20fe09b",
                "about": "",
                "name": "TheBlondeBass",
                "disable3rdPartyTrackers": false,
                "isPowerContributor": false,
                "joinedAt": "2015-12-04T05:33:28",
                "rep": 7.832951000000001,
                "profileUrl": "https://disqus.com/by/modernmethod-
                   ca38cba2e6c8d5ffb15a3b87e20fe09b/",
                "url": "http://destructoid.com/blogs/TheBlondeBass",
                "reputation": 7.832951000000001,
                "location": "",
                "isPrivate": false,
                "signedUrl": "http://disq.us/?url=...",
                "isPrimary": true,
                "isAnonymous": false,
                "id": "185784770",
                "reputationLabel": "High",
```


Ispis 2.12: Prikaz dijela odgovora na GET zahtjev za komentare (prvi dio)


```
                "avatar": {
                    "small": {
                        "permalink": "https://disqus.com/api/users/
                           avatars/modernmethod-
                           ca38cba2e6c8d5ffb15a3b87e20fe09b.jpg",
                        "cache": "https://c.disquscdn.com/uploads/
                           users/18578/4770/avatar32.jpg?1623981673"
                    },
                    "isCustom": true,
                    "permalink": "https://disqus.com/api/users/
                       avatars/modernmethod-
                       ca38cba2e6c8d5ffb15a3b87e20fe09b.jpg",
                    "cache": "https://c.disquscdn.com/uploads/users
                       /18578/4770/avatar92.jpg?1623981673",
                    "large": {
                        "permalink": "https://disqus.com/api/users/
                           avatars/modernmethod-
                           ca38cba2e6c8d5ffb15a3b87e20fe09b.jpg",
                        "cache": "https://c.disquscdn.com/uploads/
                           users/18578/4770/avatar92.jpg?1623981673"
                    }
                }
            },
            "media": [
                {
                    "thread": "8480376159",
                    "resolvedUrl": "https://www.youtube.com/watch?v
                       =6bfBhIM5tb4",
                    "thumbnailUrl": "//a.disquscdn.com/get?...",
                    "htmlWidth": 854,
                    "id": 81433613,
                    "thumbnailWidth": 480,
```

```
                "title": "I have no idea who this is",
                "htmlHeight": 480,
                "mediaType": "3",
                "html": ``,
                "location": "",
                "type": "2",
                "metadata": {
                    "create_method": "preview",
                    "thumbnail": "//a.disquscdn.com/get?url=..."
                },
                "urlRedirect": "https://disq.us/url?url=...",
                "description": "When Lex Luthor and The Flash
                   swap brains, Lex tries to make the best out
                   of a bad situation.",
                "post": "5347473378",
                "thumbnailURL": "//a.disquscdn.com/get?url=https
                   ..",
                "providerName": "YouTube",
                "forum": "destructoid",
                "url": "https://youtu.be/6bfBhIM5tb4?t=16",
                "resolvedUrlRedirect": "https://disq.us/url?url=
                   https...",
                "thumbnailHeight": 360
            }
        ],
```


Ispis 2.13: Prikaz dijela odgovora na POST zahtjev za komentare (drugi dio)


```
        "isSpam": false,
        "isDeletedByAuthor": false,
        "isHighlighted": false,
        "hasMore": false,
        "parent": null,
        "isApproved": true,
        "isNewUserNeedsApproval": false,
        "isDeleted": false,
        "isFlagged": false,
        "raw_message": "https://youtu.be/6bfBhIM5tb4?t=16",
        "isAtFlagLimit": false,
        "canVote": false,
        "forum": "destructoid",
        "depth": 0,
        "points": 15,
        "moderationLabels": [
            "links",
            "media"
        ],
        "isEdited": false,
        "sb": false
    },
    {
        "editableUntil": "2021-04-23T16:54:09",
        "dislikes": 1,
        "thread": "8480376159",
        "numReports": 0,
        "likes": 3,
        "message": "<p>Oh, you should. Lindsey is a really great
            essayist. Here's a few videos I very much enjoyed:</
           p><p>
        "id": "5347585397",
        "createdAt": "2021-04-16T16:54:09",
        "author": {
                ...
    }
...
```

```
    }
    ...
}]
```

Ispis 2.14: Prikaz dijela odgovora na POST zahtjev za komentare (treći dio)

Opširnost informacija o komentarima i autorima uzrokovala je da ispisi prikazuju samo malen dio odgovora na GET zahtjev iz ispisa 2.10, no struktura je objašnjena i ponavlja se kroz ostatak rezultata. Vidljivo je da *Disqus* šalje značajno više informacija u odgovorima na zahtjeve nego li dodatak *Facebook comments*.

2.3. 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 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 [12]. Kada preglednik šalje zahtjev na URL, obično postavlja određena zaglavlja. Tipična zaglavlja koja preglednik postavlja su `Accept`, `Accept-Encoding`, `Accept-Language`, `Connection`, `Cookie`, `Host` ali i brojna druga. 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 vjerojatnije će biti detektiran kao nelegitiman. Vjerojatnost detekcije dalje se može smanjiti nasumičnim odabirom ključeva u zaglavlju, ali i njihovih vrijednosti [12]. Najbolja obrana od detekcije je promjena IP adrese [12], 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 odmakom. 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. Tako se osigurava da je između svakog pokušaja rastuća vremenska razlika što povećava vjerojatnost uspješnog ponovnog pokušaja.

2.4. Promjene u sustavima

Sustavi za komentiranje ne mijenjaju se često, čak toliko rijetko da se tijekom istraživanja i pisanja rada dogodila samo jedna detektirana promjena. Unatoč tomu što su rijetka pojava, promjene koje nastupe moraju se shvatiti ozbiljno jer imaju potencijal onemogućiti ispravan rad sustava. Na primjeru prethodno objašnjenog *Disqus* sustava komentiranja, malena promjena URL-a za dohvat inicijalnog skupa komentara uzrokovat će da se ne može parsirati identifikator niza. Bez identifikatora ne može se izgraditi

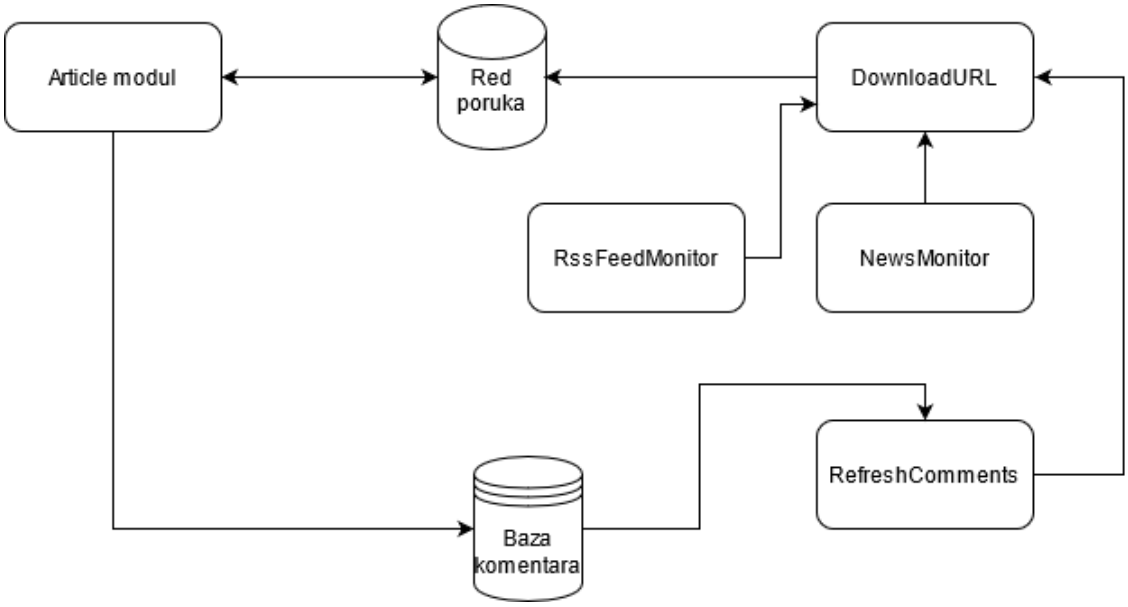
ispravan zahtjev koji će dohvatiti sljedeći skup komentara. Na takav tip promjena sustav ne može imati spremno rješenje zato što je nemoguće predvidjeti u koju vrijednost će se ispravan URL promijeniti. Zbog toga je osmišljen sustav testiranja i hijerarhija iznimnih slučajeva koji pomažu dijagnostici i brzom razrješavanju problema kada se pojave promjene. 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 razriješiti. 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. Implementacijski detalji sustava testiranja i sustava rukovanja iznimkama bit će objašnjeni u poglavlju 4.

3. 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 prvo objasniti arhitekturu sustava uz pregled najbitnijih modula i razreda te njihovih odnosa, a zatim će objasniti klasifikaciju osmišljenog sustava. Moduli i razredi koji će biti objašnjeni su:

- `article`
- `config_loader`
- `utils`
- `checks`
- `validate.py`
- `run.py`

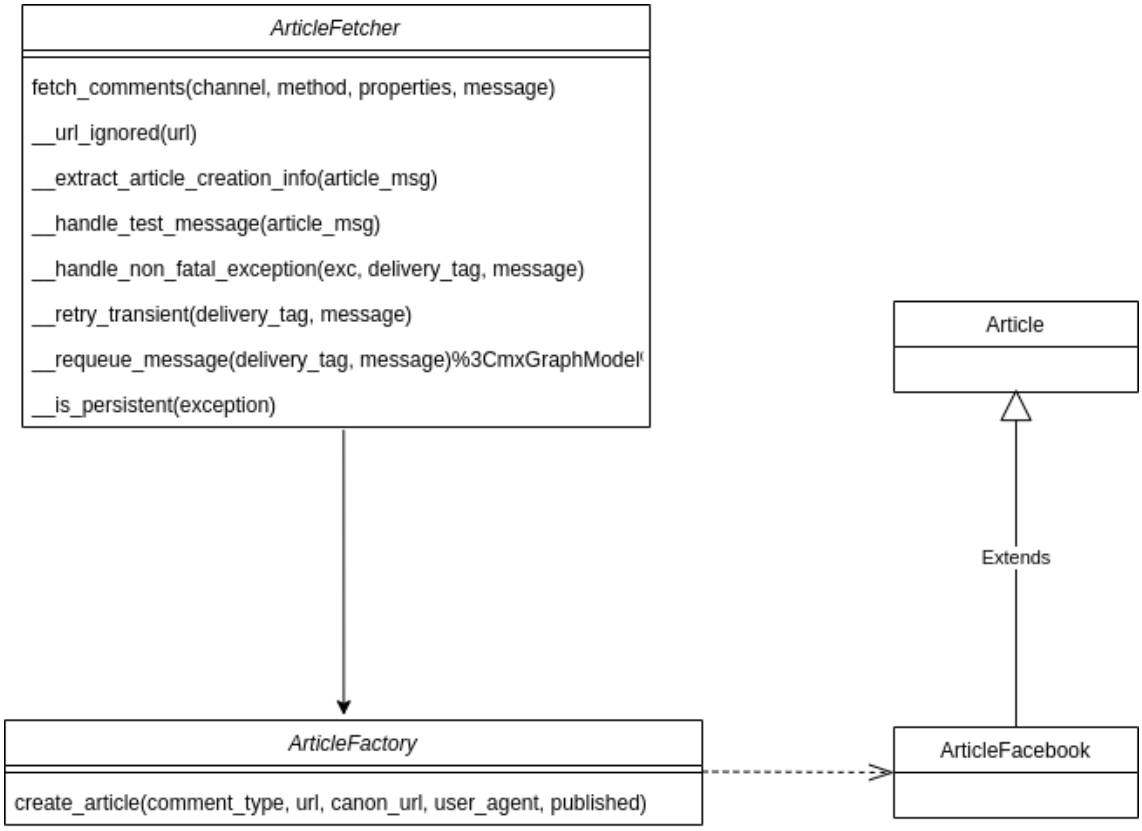
Arhitektura osnovnog sustava sastoji se od `article` modula, reda poruka, baze podataka te razreda `NewsMonitor`, `RssFeedMonitor` te `RefreshComments`. Drugim riječima, radi se o osnovnom skupu bez kojih sustav ne bi mogao raditi. Prikaz opisane arhitekture nalazi se na slici 3.1.



Slika 3.1: Arhitektura dijela sustava za dohvat komentara

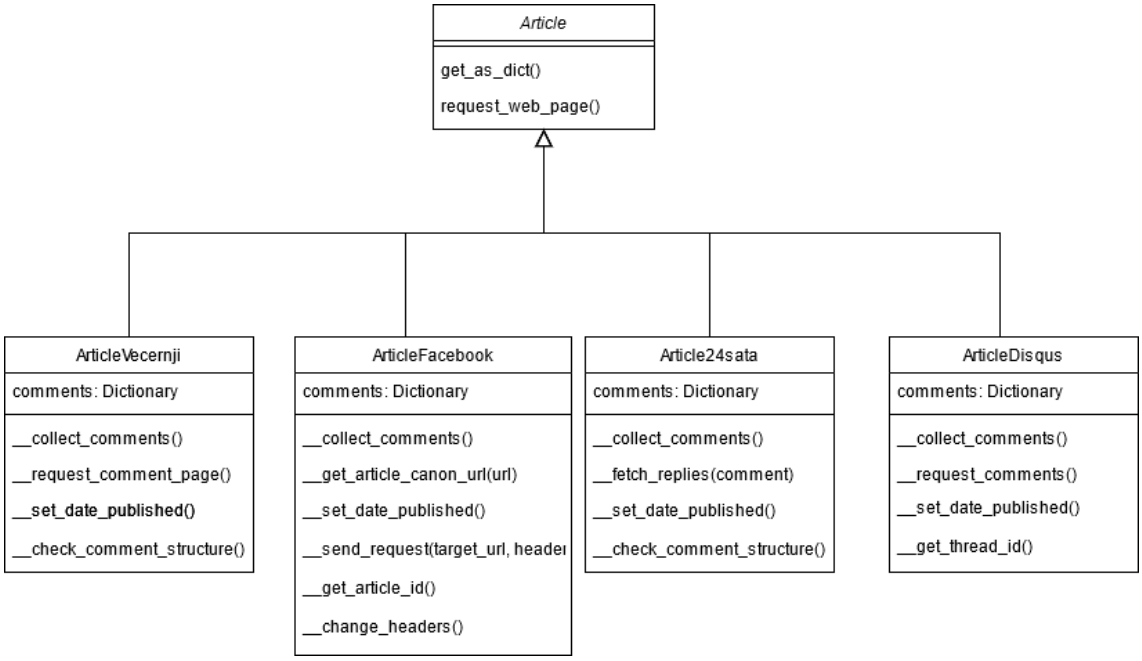
Razredi `NewsMonitor`, `RssFeedMonitor` te `RefreshComments` služe stvaranju poruka koje će se slati u RabbitMQ red. Razred `NewsMonitor` poruke šalje nezavisno o bazi podataka zajedno s `RssFeedMonitor`, dok `RefreshComments` razred mora komunicirati s bazom podataka kako bi saznao koje članke treba ponovno preuzeti. Poruke koje se šalju u red su u stvari zadaci koji nose informaciju o članku kojeg treba preuzeti. Razredi `NewsMonitor` te `RefreshComments` poruke ne šalju direktno u red, već potrebne informacije šalju `DownloadURL` razredu koji će ih formatirati i poslati u red. Poruke iz reda preuzima `article` modul te nakon obrade zadatka preuzeti članak i komentare sprema u bazu podataka. Osim prihvaćanja poruka, `article` modul može i vraćati poruke u red te je zbog toga upisana i povratna veza prema redu poruka.

Modul `article` 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, ali i vraćanje naredbi u red. `ArticleFetcher` će po primitku valjane naredbe putem `ArticleFactoryja` instancirati odgovarajući razred koji će znati rukovati tom naredbom. Razredi koji znaju rukovati naredbama zapravo su razredi koji odgovaraju jednom tipu sustava komentiranja. Na slici 3.2 prikazani su odnosi između razreda `ArticleFetcher`, `ArticleFactory` te `Article` razreda. Naime, po primitku poruke `ArticleFetcher` delegirat će posao instanciranja `ArticleFactory` razredu.



Slika 3.2: Dijagram razreda `article` modula za obradu poruka iz reda

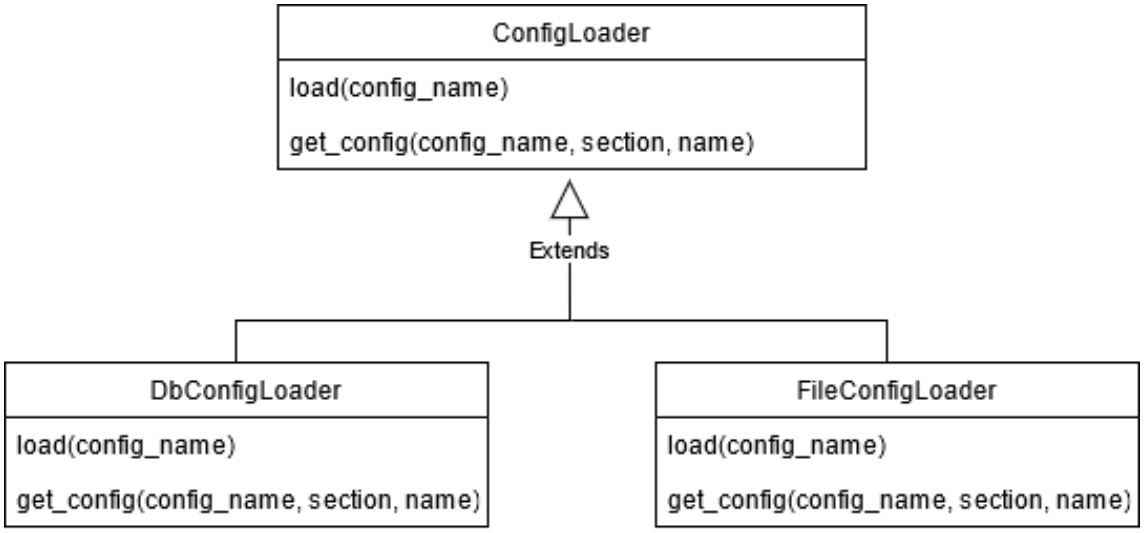
Svaki sustav komentiranja ima vlastit razred izveden iz `Article`, odnosno u sustavu postoje `ArticleFacebook`, `ArticleDisqus`, `ArticleVecernji`, `Article24sata` te bazni `Article` razred. Razlog postojanju više zasebnih razreda goleme su implementacijske razlike zbog kojih je ujedinjavanje koda u jedan razred nepraktično. Više o tome, ali i o ulozi baznog `Article` razreda bit će u poglavlju 4. Slika 3.3 prikazuje dio razreda `article` modula bez pripadajućih iznimki. Prikazani su razredi koji sadrže logiku dohvata komentara i članaka. Prikazane su njihove metode, ali ne i atributi. Razlog tomu je što atributa ima mnogo i velika većina služi internoj logici razreda, dok samo `comments` atribut kojeg dijele svi izvedeni `Article` razredi ima značaj van hijerarhije prikazane na slici 3.3. `fetch_comments` metoda `ArticleFetcher` razreda zapravo je *callback* metoda koja se poziva po primitku poruke iz reda. Osim te metode, `ArticleFetcher` sadrži i privatne metode koje služe raznim zadacima koje je potrebno obaviti tijekom obrade poruke. Tako, primjerice, metoda `__requeue_message` služi vraćanju poruke u red.



Slika 3.3: Dijagram razreda `article` modula za dohvat komentara i članaka

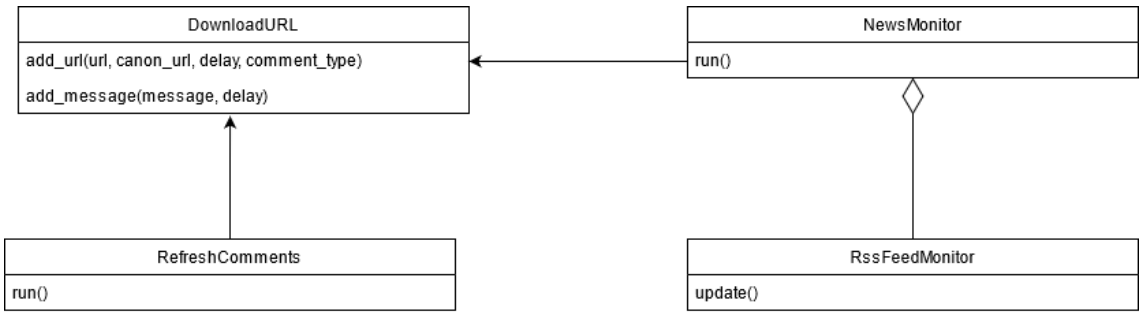
Kako bi razredi sustava mogli obaviti svoju zadaću obrade URL-a potrebni su im pomoćni `DateParser` razredi. Razreda izvedenih iz `DateParser` ima mnogo zbog toga što svaka stranica s vijestima ima različite načine pisanja kada je članak objavljen. S obzirom na to da postoji velik broj konfigurabilnih elemenata u `article` modulu, osmišljen je i modul koji će olakšati učitavanje konfiguracije.

Modul `config_loader` sadržajno je malen, međutim igra veliku ulogu u sustavu. Dijagram razreda modula prikazan je na slici 3.4. Dijagram je vrlo jednostavan zbog malenog broja razreda i metoda. Naime, bazni razred `ConfigLoader` je apstraktan te njegove metode moraju implementirati izvedeni razredi. 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 naziva `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 4.



Slika 3.4: Dijagram razreda modula `config_loader`

`Utils` modul sadrži velik broj razreda od kojih su neki jednostavni pomoćni razredi, dok su neki pak ključni za rad sustava i s `article` modulom čine osnovu sustava. Tako, posebna pažnja bit će predana razredima: `DownloadURL`, `NewsMonitor`, `RssFeedMonitor` te `RefreshComments`. Dijagram razreda koji sadrži navedene najbitnije razrede prikazan je na slici 3.5.

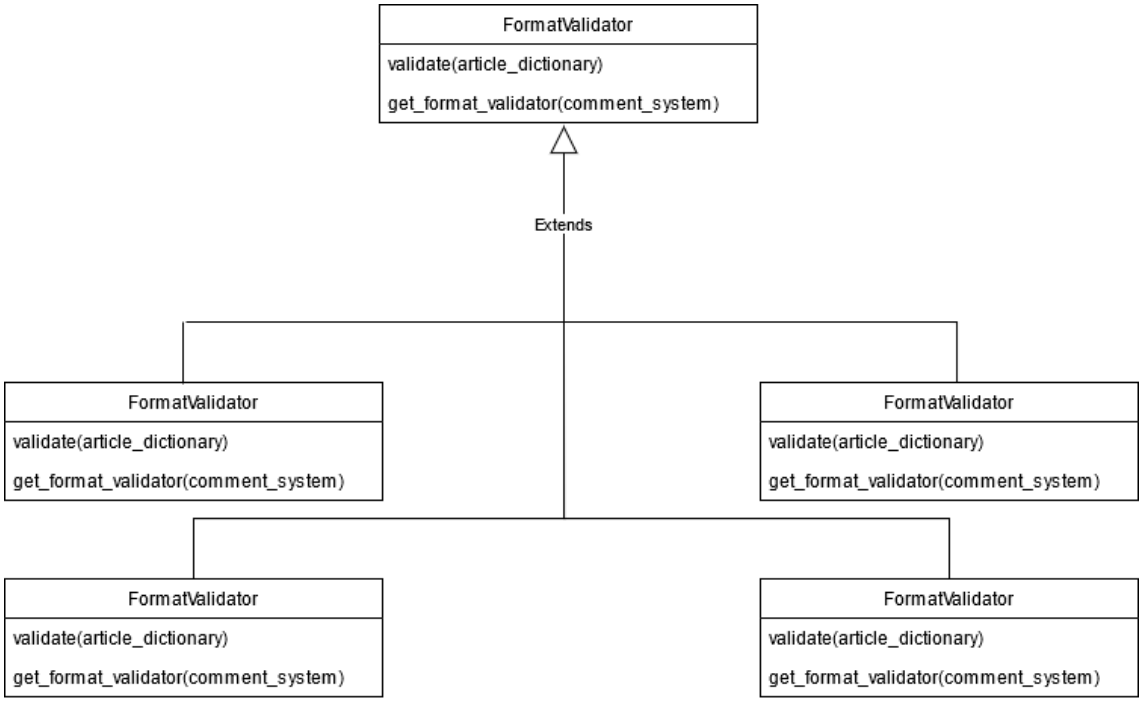


Slika 3.5: Dijagram najbitnijih razreda `utils` modula

Ta četiri 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. Pri tome koristi `RssFeedMonitor` koji prati novosti iz *RSS feedova*. Razlog takvom razlaganju na dva razreda je što dodavanjem novih izvora vijesti dovoljno je samo napisati zaseban razred i koristiti ga u `NewsMonitor` razredu bez da ostatak sustava zna za promjenu. Dalje, `DownloadURL` razred služi slanju naredbi `article` modulu, odnosno `ArticleFetcher` razredu. Njega koriste `NewsMonitor` i `RefreshComments` kako bi poslali poruke do dijela sustava koji ih može obraditi, odnosno `article` modulu. Zadnji od četiri 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 vremena 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.

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. Dijagram razreda za `checks` modul prikazan je na slici 3.6. Navedeni razredi služe provjeravanju formata preuzetih podatkovnih struktura, odnosno provjerava se jesu li strukture onakve kakvima ih ostatak sustava očekuje. `Checks` modul uparen je s `validate.py` skriptom čija je svrha objašnjena u sljedećem potpoglavlju.



Slika 3.6: Dijagram razreda u `checks` modulu

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.

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. Više o tome bit će u poglavlju o prikazu načina rada sustava.

3.1. Klasifikacija razvijenog sustava

Sustav za dohvat komentara primarno je *web scraper*, ali sadrži i svojstva web-pauka (eng. *web crawler*). Web *scraperi* su sustavi koji preuzimaju izvorne internetske stranice te ih obrađuju, odnosno ekstrahiraju podatke iz njih [10]. Web paukovi su pak sustavi koji samostalno pronalaze poveznice na internetske stranice [14]. Opisani sustavi međusobno su komplementarni te surađuju tako da jedan pronalazi poveznice, a drugi preuzima njihov sadržaj. Web pauk u sustavu relativno je jednostavan i ograničen tako 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 web pauk, no svakako sadrži svojstvo traženja novih poveznica koje je karakteristično web paukovima. Pronalazak novih stranica u potpunosti je utemeljen na praćenju *RSS feedova* koji su zadani u konfiguraciji. Drugim riječima, sustav prati obavijesti od portala koje se šalju kad god je novi članak objavljen. Poruka sadrži koji je URL novog članka tako da se vrlo lako može poslati na obradu. Na taj način sustav će obrađivati svaki novi članak svih portala koji su zadani za praćenje.

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 koje 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. Teorijska podloga izbjegavanja detekcije i rukovanja promjenama objašnjena je u potpoglavlju 2.3, dok će konkretna implementacija rukovanja promjenama biti objašnjena u poglavlju o implementaciji sustava. Konkretno, modul `article` predstavlja *scraping* dio sustava zbog toga što služi dohvatu informacija sa stranica koje dobije kroz red poruka. Dio sustava koji bi se mogao nazvati web paukom je `utils` modul, pritom misleći na razrede `NewsMonitor` te `RefreshComments`.

Scraping se u razvijenom sustavu može 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 reverznim inženjerstvom otkriti na koji način sustav komentiranja funkcionira. Preciznije, potrebno je promatrati mrežni promet i izdvojiti te analizirati zahtjeve koji su zaslužni za prijenos komentara s poslužitelja do preglednika kako bi se mogli iskoristiti programski 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 poglavlju 2. je na primjeru dodatka *Facebook comments* te sustava *Disqus* objašnjeno kako reverzati protokole koje koriste te kako iskoristiti pronađene zahtjeve i informacije da bi se dohvatili komentari, ali i kako izbjeći detekciju te koje promjene

su moguće.

4. Implementacija sustava

Ovo poglavlje dat će opširni pregled implementacije potreban da bi se bolje razumjeli problemi koje je rad riješio. Objašnjeno je koje su tehnologije izabrane za implementaciju te iz kojih razloga, na što se pazilo tijekom implementacije vezano uz dobre prakse programskog inženjerstva, a objašnjeno je i na koje konkretne načine je kod poboljšavan. Zatim su objašnjeni implementacijski detalji kao nadogradnja na teorijsku podlogu objašnjenu u prva tri potpoglavlja ovog poglavlja.

4.1. 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 [8]. 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 što 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 (eng. *queue-based communication*). Odabir komunikacije temeljene na redovima omogućuje buduće nadogradnje u smislu konkurentne obrade više zadataka, ali i omogućuje da pošiljatelj zadatka ne ovisi o implementaciji razreda koji će zadatak obraditi. Konkretna tehnologija koju se koristi je RabbitMQ. Razlog odabira su dobra dokumentacija, jednostavnost korištenja te podrška u Pythonu.

4.2. 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 tako 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 što je prošlo puno vremena [2]. Zbog tih razloga tijekom razvoja nastojalo se maksimalno pridržavati PEP 8 vodiča uz određene iznimke.

Osim stilskog vodiča, primjenjivale su se dobre 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 dugačke metode, loše imenovanje, metode s previše argumenata, kršenje jedinstvene odgovornosti, pojava magičnih brojeva te duplikacija koda. Osim problematičnog koda koji je potrebno razriješiti, nastojalo se uvesti i promjene koje poboljšavaju kod koji nije problematičan, no nije niti optimalan.
Jedan od principa koji pomažu očuvati kvalitetu koda u cijelom sustavu glasi da je potrebno prilikom svake izmjene nad kodom pogledati kod i napraviti refaktoriranje (eng. *refactor*) gdje god je moguće [7]. Refaktoriranje je restrukturiranje postojećeg koda u svrhu poboljšanja kvalitete bez promjene vanjskog ponašanja. Refaktoriranje postiže bolju čitljivost, lakšu održivost, lakšu nadogradnju i poboljšava opću kvalitetu koda [1]. Primjenom malih refaktoriranja često tijekom životnog ciklusa razvoja osigurava se da kod kroz vrijeme ne postaje sve neuredniji, već u najgorem slučaju kompleksnost stagnira.

4.3. Korištene tehnike refaktoriranja

Tehnika refaktoriranja ima mnogo, a sežu od malenih promjena poput preimenovanja metoda pa do većih promjena poput ekstrahiranja razreda. U ovom potpoglavlju bit će objašnjene sve tehnike korištene tijekom implementacije kako bi sustav bio izgrađen u skladu s dobrim praksama programskog inženjerstva. Najčešće korištene tehnike refaktoriranja u ovom radu su ekstrahiranje metode (eng. *extract method*), micanje metode (eng. *move method*), ekstrahiranje razreda (eng. *move class*), zamjena uvjeta polimorfizmom (eng. *replace conditional with polymorphism*), zamjena magične brojke konstantom (eng. *replace magic number with constant*), preimenovanje razreda i metoda (eng. *rename class / rename method*), ekstrahiranje nadrazreda (eng. *extract superclass*) te ekstrahiranje metode u nadrazred (eng. *pull up method*).

Jedna od najčešće korištenih tehnika u ovom radu je ekstrahiranje metode. Ekstrahiranje metode koristi se u slučajevima dugačkih metoda koje su teško čitljive i izazovne za mijenjati. Postupak provođenja tehnike refaktoriranja je pronalazak logičkih cjelina u dugačkoj metodi te ekstrahiranje istih u zasebne metode [1]. Tako se osnovna metoda značajno skraćuje te najčešće postaje i konceptualno čišćom, odnosno lakše je shvatiti što metoda radi bez dubljeg proučavanja.

Micanje metode je tehnika refaktoriranja u kojoj se metoda premješta iz jednog razreda u drugi i jedna je od temeljnih tehnika [1]. Najčešći slučajevi gdje se koristi su kada se pojedina metoda više koristi u drugom razredu ili metoda konceptualno ne pripada razredu iz kojeg se miče. Kao tehnika refaktoriranja komplementarna je tehnici ekstrahiranja razreda. Razlog tomu je što nakon što je metoda ekstrahirana ponekad je vidljivo da je izdvojeno ponašanje prikladnije smješteno u drugi razred, bilo postojeći ili novi. Tehnika se svodi na jednostavni premještaj metode iz jednog razreda u drugi.

Ekstrahiranje razreda koristi se u slučajevima kada razred radi više stvari, odnosno kada se krši princip jedinstvene odgovornosti [13]. Tehnika se svodi na ekstrahiranje metode te micanje metode, odnosno kombinaciju prethodno objašnjenih dviju tehnika. Nakon primjene tehnike ekstrahiranja razreda postiže se veća konceptualna čistoća razreda, odnosno razred više nije opterećen više odgovornosti već samo jednom.

Zamjena uvjeta polimorfizmom jedna je od najzanimljivijih tehnika korištenih u ovom radu. Tehnika se nastoji riješiti kompleksnih uvjeta koji osim što su teški za čitati, prilikom svake nadogradnje moraju se mijenjati. Općenito se smatra dobrom praksom izbjegavanje kompleksnih uvjeta, posebice `switch-case` struktura [7]. Provođenje se svodi na stvaranje više podrazreda od kojih svaki odgovara jednom od željenih uvjeta. Tako umjesto kompleksnog uvjeta, potreban je samo jedan poziv predanoj instanci razreda [1]. Konkretan primjer primjene tehnike nad izgrađenim sustavom bit će dan u poglavlju 6.

Zamjena magične brojke konstantom jedna je od najstarijih tehnika refaktoriranja koja se rješava jednog od najstarijih problema u kodu [7]. Naime, magične brojke su problematične zbog toga što je programeru koji čita kod teško zaključiti koje je značenje te brojke [1]. Postoje međutim iznimke, kao što je primjerice formula za izračun opsega `radius * Math.PI * 2` u kojoj je prihvatljivo ostaviti faktor `2` [7]. Međutim, u ostalim slučajevima takvo stanje pogodno je za nastanak grešaka. Jednostavno rješenje je uvođenje konstanti čija imena daju do znanja koje je značenje njihovih vrijednosti.

Preimenovanje razreda i metoda najjednostavnija je tehnika, ali i najčešće korištena u sklopu ovog rada. Tehnika se koristi u slučajevima kada ime metode ili razreda ne odgovara njihovom ponašanju ili odgovornosti [13]. Unatoč tomu što se metoda odnosi na metode i razrede, primjenjiva je i na varijable. Programer koji po prvi puta čita kod neće se htjeti zadržavati na unutarnjoj logici razreda i metoda, već će htjeti pregled cijelog sustava. Loše imenovanje u najboljem slučaju skriva ponašanje, a u najgorem slučaju obmanjuje čitatelja. Imena trebaju odgovoriti na sva glavna pitanja

- zašto postoji, što radi i na koji način se koristi [7]. Preimenovanje pomaže čitatelju da na prvi pogled može sa sigurnošću reći koja je jedinstvena odgovornost razreda ili metode. Konkretan primjer preimenovanja bit će dan u poglavlju 4.

Ekstrahiranje nadrazreda je tehnika koja se koristi kada više razreda ima nekoliko zajedničkih metoda ili atributa, odnosno kada se kod duplicira [13]. Tehnika se svodi na stvaranje nadrazreda koji sadrži zajedničko ponašanje i atribute, a ostali razredi ga nasljeđuju. Stvorena je smislena hijerarhija razreda i kod je dedupliciran. Nakon provođenja tehnike kod je lakše čitljiv zbog toga što izvedeni razredi sadrže samo specifična ponašanja, ali je i otporniji na greške jer ne postoje duplikacije koda. Ekstrahiranje metode sadržano je u tehnici ekstrahiranja nadrazreda s jednom razlikom da se odnosi i na slučajeve kada već postoji nadrazred u koji je moguće ekstrahirati metodu.

4.4. Primjeri refaktoriranja

Ovo potpoglavlje nastojat će prikazati primjere problematičnog koda te koje od tehnika refaktoriranja su korištene. Najčešći problemi koje je refaktoriranje nastojalo riješiti bili su predugačke metode, loše imenovanje, kršenje jedinstvene odgovornosti, kompleksni uvjeti te magične brojke. U rijetkim slučajevima pojavila se i duplikacija koda.

Prvi primjer odnosi se na dugačke metode i kršenje jedinstvene odgovornosti.
Primjer dugačke metode moguće je pronaći u prvoj verziji razreda `ArticleFetcher` u metodi `commentsFetchCallback`. Odsječak nad kojim će se raditi refaktoriranje prikazan je u ispisima 4.1 i 4.2.

```
if isFacebookArticle(articleMsg):
    try:
        article = ArticleFacebook(articleMsg['url'], cannonicalURL=
           articleMsg['cannonicalURL'], userAgent = self.user_agent,
            published = published)
    except requests.exceptions.ReadTimeout:
            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:
            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 4.1: Problematičan odsječak koji instancira `Article` razrede u `commentsFetchCallback` metodi

```
elif articleMsg['commentType'] == 'vecernji':
    article = VecernjiComments(articleMsg['url'], cannonicalURL=
       articleMsg['cannonicalURL'], userAgent = self.user_agent,
       published = published)
    self.channel.basic_ack(method.delivery_tag)
    return

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']))
        self.channel.basic_ack(method.delivery_tag)
    else:
        print("Unknown/unsupported commenting type {} for URL {}.
           Ignoring".format(articleMsg['commentType'], articleMsg['
           cannonicalURL']))
        self.channel.basic_ack(method.delivery_tag)
```

Ispis 4.2: Problematičan odsječak koji instancira `Article` razrede u `commentsFetchCallback` metodi (nastavak)

Ispisi 4.1 i 4.2 prikazuju najproblematičniji odsječak `commentsFetchCallback` metode razreda `ArticleFetcher`. Problem s metodom je što sadrži kompleksan uvjet za instanciranje `Article` razreda koji ju čini teško čitljivom i krši princip jedinstvene odgovornosti. Naime, svrha metode `commentsFetchCallback` je obrada poruka koje dolaze iz RabbitMQ reda, odnosno mora pokrenuti dohvaćanje i spremanje članka i njegovih komentara. Međutim, dodavanje logike instanciranja `Article` razreda ne bi trebalo biti u toj metodi. Uvjet prikazan u ispisu služi instanciranju odgovarajućeg `Article` razreda u ovisnosti o sadržaju poruke koju je dobio iz reda. Primijenjene su dvije tehnike refaktoriranja, a to su ekstrahiranje razreda te zamjena uvjeta polimorfizmom.

Primjenom tehnike ekstrahiranja razreda stvoren je novi razred `ArticleFactory` po uzoru na oblikovni obrazac tvornice s metodom `create_article` čija je jedina odgovornost instanciranje odgovarajućeg `Article` razreda. Rezultat primjene tehnike ekstrahiranja razreda malen je napredak, međutim i dalje je potrebno riješiti kompleksan uvjet. Rješenje je uvođenje polimorfizma korištenjem refleksije u Pythonu. `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 `article.ArticleDisqus` razred. U ispisu 4.3 prikazana je implementacija razreda tvornice. Sve se odvija unutar `try-catch` bloka s namjerom da se uhvati iznimka `ModuleNotFoundError` koja daje do znanja da je ili naziv kriv ili taj sustav komentiranja još nije podržan. Po hvatanju iznimke, diže se nova iznimka tipa `NotYetSupportedError` koja je definirana u `Article` razredu.

```
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
```

Ispis 4.3: Kod implementacije tvornice

Rješenje prikazano u ispisu 4.3 lakše je čitljivo, a predstavlja i poboljšanje u održivosti u odnosu na izvorni kod. Ovakvo rješenje omogućava nadogradnju bez promjene zbog toga što se instanciranje vrši dinamički, odnosno ne instancira se prema predefiniranom imenu razreda. Kada je potrebno dodati novi sustav komentiranja, `ArticleFactory` ne mora se mijenjati. U ispisu 4.4 prikazan je novi oblik problematičnog odsječka iz ispisa 4.1 te 4.2.

```
        try:
            article = ArticleFactory.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 4.4: Poboljšani odsječak koda iz `commentsFetchCallback`

Osim očigledne razlike u čitljivosti, kod iz ispisa 4.4 otporan je na promjene u slučaju dodavanja novih sustava komentiranja zbog toga što ne ovisi niti o jednom specifičnom sustavu komentiranja, već ovisi samo o jednoj metodi tvornice.

Primjera dugačkih metoda bilo je mnogo tijekom razvoja sustava. 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 tako da ih obavlja pozivima drugim metodama ili razredima. Ispisi 4.5 te 4.6 prikazuju problematičan kod s puno uvjeta koji služe parsiranju datuma kojeg je potrebno razriješiti primjenom tehnika refaktoriranja.

```
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(date_regex, 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()
```

Ispis 4.5: Problematičan kod parsiranja datuma iz `__init__` metode `ArticleFacebooka`

```
    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 4.6: Problematičan kod parsiranja datuma iz `__init__` metode `ArticleFacebooka` (nastavak)

Ispisi 4.5 i 4.6 prikazuju samo dio kompleksne logike parsiranja datuma koja zauzima velik dio koda u `__init__` metodi `ArticleFacebook` razreda kako bi se vjerno dočarala veličina i kompleksnost metode. Vidljivi su `if-elif` uvjeti od kojih svaki odgovara jednoj stranici, odnosno svaki pokriva specifičnosti parsiranja datuma jednog portala kojeg sustav prati. Dva su problema koja treba riješiti. Prvi je kompleksan uvjet koji bi trebao biti zamijenjen polimorfizmom. Drugi je sama veličina metode koja otežava čitanje i održavanje, ali i krši princip jedinstvene odgovornosti.
Kako bi se razriješio problem, korištene su tehnike refaktoriranja ekstrahiranje metode, micanje metode te zamjena uvjeta polimorfizmom.

Prvo se ekstrahira razred, odnosno uvodi se niz razreda `DateParser` koji sadrže logiku parsiranja za svaku od pojedinih stranica. Bitno je napomenuti da je metoda ekstrahiranja razreda zapravo kombinacija tehnika izdvajanja metode i micanja metode samo sa stvaranjem novog razreda. 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`. Poboljšani odsječak koda nalazi se u ispisu 4.7.

```
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 4.7: Poboljšani odsječak koda iz ispisa 6.4

Ispis 4.7 prikazuje poboljšani odsječak koda prikazanog u ispisima 4.5 i 4.6 koji dohvaća instancu odgovarajućeg `DateParser` razreda za pojedinu stranicu te pokreće parsiranje datuma. 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`. Konkretno, odsječak koda iz 4.7 prvo dohvaća ime portala pozivom na metodu `get_host_from_url` čija je implementacija prikazana u ispisu 4.8. Implementacija dohvata imena portala se svodi na korištenje `urllib` biblioteke kako bi se iz predanog URL-a dohvatilo domensko ime. Iz tog imena se zatim parsira sam naziv portala.

Ispis 4.8: Metoda za dohvat imena portala iz URL-a captionpos

```
def get_host_from_url(url):
    netloc = urllib.parse.urlparse(url).netloc
    if netloc.find('www') != -1:
        netloc = netloc[3:]
    if netloc.find('com') != -1:
        netloc = netloc[:-3]
    if netloc.find('.hr') != -1:
        netloc = netloc[:-3]
    if netloc.find('.net') != -1:
        netloc = netloc[:-3]
    if netloc.find('.org') != -1:
        netloc = netloc[:-3]
    return netloc.replace(".", "")
```

Nakon što je dohvaćeno ime portala, šalje se metodi `get_date_parser` iz `DateParser` modula.

Implementacija metode za dohvat odgovarajuće instance `DateParser` razreda prikazana je u ispisu 4.9.

```
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 4.9: Prikaz dinamičkog instanciranja `DateParser` razreda

Ispis 4.9 prikazuje metodu `get_date_parser` koja instancira i vraća odgovarajući `DateParser` razred. Metoda radi na isti način kao i tvornička metoda u `ArticleFactory` , odnosno pokušava instancirati objekt prema dobivenom argumentu koji predstavlja naziv portala, a samo ime razreda gradi iz prefiksa 'DateParser' te naziva stranice.

Još jedan čest problem u sustavu bilo je loše imenovanje varijabli i metoda. Pri refaktoriranju, 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 redundancije u imenovanju te da se izbjegavaju općenita imena poput `list` ili `data` [7]. Tijekom pisanja sustava jedno od prvih refaktoriranja bilo 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 `Comments` sufiks zamijenjen s `Article`. Drugi razlog 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.

Duplikacija koda nije se često pojavljivala u sustavu zbog pažnje pri implementaciji. 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 i prikazan je u ispisu 4.10.

```
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. Requieing.')
        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 4.10: Primjer dupliciranog koda za rukovanje iznimkom

Ispis 4.10 prikazuje dupli cirani 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. Prikaz implementacije metode za vraćanje u red nalazi se u ispisu 4.11. Logika metode svodi se na slanje potvrde o primitku poruke, ali i slanju istog sadržaja nazad u red kako bi se primio nakon perioda kojeg definira `DELAY` konstanta.

```
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 4.11: Implementacija metode za vraćanje poruke u red

U ispisu 4.12 prikazana je poboljšana implementacija koda iz ispisa 4.10.

```
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. Requeing.')
    requeue_msg(method.delivery_tag, message)
    return
```

Ispis 4.12: Poboljšana implementacija bez duplikacije koda

Ispis 4.12 prikazuje novo stanje u `ArticleFetcher` koje više nema duplikacije koda. Sada umjesto da se logika duplicira, pozivom na istu metodu je osigurano da nema duplikacije. U slučaju da će se logika vraćanja u red mijenjati, mogućnost da se uvede *bug* u sustav je smanjena.

Zadnji primjer primjene tehnika refaktoriranja u sustavu je zamjena magičnih brojeva konstantama. U ispisu 4.13 prikazan je primjer magičnih brojeva u kodu razvijenog sustava.

```
if response.status_code in [301, 302, 307]:
    self.log.error('Redirected or moved.')
    raise RedirectError('Redirected or moved.')
```

Ispis 4.13: Primjer statusnih kodova kao magičnih brojeva

Ispis 4.13 prikazuje provjeru statusnog koda odgovora na zahtjev u `__save_html_page` metodi `ArticleFacebook` razreda. Unatoč tomu što su neki statusni kodovi poput `200` ili `404` dobro poznati, pretpostavka je da većina programera napamet neće znati koji kod nosi koje značenje. Poboljšanje bi bilo koristiti konstante iz `requests` biblioteke Pythona. Ispis 4.14 prikazuje poboljšanje koda nakon uvođenja konstanti. Sada postoje imena koja programeru daju do znanja što se provjerava u uvjetu.

```
if response.status_code in [requests.codes.moved_permanently,
                            requests.codes.temporary_redirect,
                            requests.codes.permanent_redirect]:
    self.log.error('Redirected or moved.')
    raise RedirectError('Redirected or moved.')
```

Ispis 4.14: Zamjena magičnih brojeva konstantama za statusne kodove

Ekstrahiranje nadklase kao tehnika refaktoriranja primijenjena je jednom u sustavu unatoč tomu što postoji više hijerarhija razreda, pogotovo ako se uzmu u obzir iznimke. Više o tome bit će u sljedećim potpoglavljima. Razredi odgovorni za dohvat komentara `ArticleFacebook, ArticleVecernji, ArticleDisqus, Article24sata` srodni su te dijele neke zajedničke metode. Primjer toga je zajednička logika dohvata članka prije dohvata komentara. Zbog toga je u novi apstraktni razred `Article` izdvojena metoda `request_web_page(url, headers)`. Ispis 4.15 prikazuje implementaciju apstraktnog razreda `Article`.

```
class Article(ABC):
    """ Abstract base class for all article classes"""

    def get_as_dict(self):
        """ Derived classes must implement this function"""
        return 0

    def request_web_page(self, url, headers):
        try:
            response = requests.get(url, params=headers, timeout=5)
            return response
        except requests.exceptions.ReadTimeout:
            raise ArticleFetchError("Timeout while archiving HTML page")
        except requests.exceptions.TooManyRedirects:
            raise ArticleFetchError("Too many redirects while archiving HTML
                page")
        except requests.exceptions.MissingSchema:
            raise InvalidUrlError("The url is most likely not valid. Check
                it and try again")
        except requests.exceptions.ConnectionError:
            raise ArticleError("The url could not be connected to.")
```

Ispis 4.15: Poboljšana implementacija bez duplikacije koda

Primjenom postupka ekstrahiranja nadklase, ali i metode, dedupliciran je kod i olakšano je čitanje ostalih izvedenih razreda. Metoda za dohvat HTML koda stranice nalazi se na jedinstvenom mjestu i dostupna je svim postojećim i budućim razredima izvedenima iz `Article`. Jedan zanimljiv implementacijski detalj je osiguravanje apstraktnosti razreda tako što `Article` nasljeđuje razred `ABC` iz Pythonove biblioteke `abc`.

Osim prikazanih primjera, pažnja je posvećena i da se određena refaktoriranja izvrše čim nastaje potreba. Primjer toga je uklanjanje nekorištenih metoda, varijabli te *importova*. Unatoč tomu što ostavljanje nekorištenog koda ne uzrokuje neželjene posljedice

u radu sustava, ono otežava čitanje koda. Posebno bitan primjer je micanje nepotrebnih argumenata iz metoda. Vodeći se principom da treba težiti metodama bez argumenata, a maksimalno tolerirati dva argumenta u metodi, metoda `__change_headers` iz `ArticleFacebook` koja je imala jedan argument izmijenjena je tako da ne prima niti jedan argument. Prednost micanja argumenata je olakšavanje čitanja koda. Što je veći broj argumenata, to je potreban veći napor pri čitanju, ali i testiranju [7]. Ispis 4.16 prikazuje primjer metode s jednim argumentom.

```
def __change_headers(self, headers):
    headers['Cookie'] = "c_user=" + random.randint(self.random_min, self.
        random_max)
    headers['User-Agent'] = random.choice(self.user_agent_values)
```

Ispis 4.16: Implementacija metode za promjenu zaglavlja

Ispis 4.17 prikazuje metodu `__change_headers` kojoj je maknut argument `headers` tako da se referencira na atribut instance.

```
def __change_headers(self):
    self.headers['Cookie'] = "c_user=" + random.randint(self.random_min,
        self.random_max)
    self.headers['User-Agent'] = random.choice(self.user_agent_values)
```

Ispis 4.17: Metoda za promjenu zaglavlja bez argumenata

Prednost novog oblika je što testiranje ne mora uzimati u obzir kombinacije ulaznog argumenta. Osim toga, olakšana je čitljivost koda zbog toga što se ne mora pratiti otkuda dolazi argument već je poznato da je to atribut instance.

4.5. Sustav rukovanja iznimkama

U poglavlju 2. koje govori o teorijskoj podlozi rada spomenuto je da se sustavi komentiranja mijenjaju i da to u nekim slučajevima znači da razvijeni sustav ne može normalno raditi. Kako bi sustav pravilno reagirao na različite vrste promjena, bilo je potrebno osmisliti sustav rukovanja iznimkama koji će olakšati taj posao. Prvi korak bio je nadogradnja postojećih iznimki u sustavu. S obzirom na to da različite promjene uzrokuju različite pojave tijekom rada sustava, potrebno je osmisliti hijerarhiju iznimki koja vjerno reflektira vanjsku promjenu koja je uzrokovala iznimku. Iznimke koje su povezane sa sustavima komentiranja nalaze se u `article` modulu.

4.5.1. Iznimke article modula

Osnovne iznimke definirane su u apstraktnom baznom razredu `Article` i to su sljedeće: `ArticleError`, `ArticleFatalError`, `InvalidUrlError`, `ArticleFetchError`, `NotYetSupportedError`, `ArticleURLParseError` te `DateNotFoundError`.

Iznimke `ArticleError` i `ArticleFatalError` nisu namijenjene instanciranju, već predstavljaju bazne razrede iz kojih se specifične iznimke mogu izvesti. `ArticleError` bazni je razred za sve iznimke koje ne ukazuju na trajnu promjenu sustava komentiranja. `ArticleFatalError` iznimke su koje ukazuju na to da se vrlo vjerojatno jedan ili

više sustava komentiranja promijenio te da sustav više ne može normalno raditi.

`InvalidUrlError` i `ArticleURLParseError` iznimke su koje koriste svi izvedeni razredi u `article` modulu za slučajeve kada URL članka ili nije valjan ili se iz njega ne može parsirati potrebna informacija, kao što je primjer za Index portal.

`ArticleFetchError` izveden je direktno iz `ArticleError` te predstavlja širok skup slučajeva u kojima iz nekog razloga nije moguće dohvatiti komentare ili sam članak.

`NotYetSupported` iznimka služi kad se pokuša dohvatiti članak koji koristi sustav komentiranja koji nije podržan u sustavu.

Svaki od `Article` razreda izvodi vlastite iznimke kako bi iznimka odgovarala specifičnostima pojedinog sustava komentiranja. Tako `ArticleFacebook` sadrži hijerarhiju iznimki s dvije vršne iznimke. To su vršne iznimke za asinkroni dohvat naziva `AsyncFetchError` te iznimka za sve ostale probleme prilikom dohvata komentara naziva `FacebookCommentFetchError`. `AsyncFetchError` predstavlja sve slučajeve u kojima asinkroni dohvat komentara nije uspio. Konkretan opis postupka na koji se iznimka odnosi nalazi se u poglavlju 2. `FacebookCommentFetchError` predstavlja sve slučajeve u kojima dohvat komentara nije uspio, osim asinkronog dohvata. Iz razreda `FacebookCommentFetchError` izvedena je specifična iznimka `FacebookArticleIdParseError` koja ukazuje na to da se nije uspio dohvatiti identifikator članka. Osim iznimki vezanih uz specifičnosti sustava komentiranja, postoje i iznimke vezane uz pojedinosti portala. Primjerice, Index ima poseban oblik URL-a koji se mora izvesti iz originalnog URL-a članka kako bi se dohvatili komentari. Tako postoji `IndexUrlParseError` iznimka izvedena iz osnovne iznimke `ArticleURLParseError` iz `Article` razreda.

Ostali razredi izvedeni iz `Article` razreda prate isti koncept definiranja iznimki kao i `ArticleFacebook`. Jedine razlike su u broju i namjeni iznimki što je direktan rezultat različitosti načina rada sustava za komentiranje.

4.5.2. Trajne i privremene iznimke

Jedna od specifičnosti u razvijenom sustavu rukovanja iznimkama je dodatna podjela na trajne, odnosno perzistentne te privremene, odnosno tranzijentne iznimke. Ideja iza podjele je da se pojedine iznimke mogu konfigurabilno zadati kao tranzijentne i perzistentne s obzirom na njihov značaj. Kada se u konfiguraciji zada da je pojedina iznimka tranzijentna, a uhvaćena je u kodu, obrada tog članka ponovit će se dokle god dohvat komentara ne uspije ili se ne dosegne maksimalni broj ponovnih pokušaja za taj članak. U slučaju da je iznimka perzistentna, članak će biti spremljen u bazu podataka bez sadržaja sa statusom da je nedostupan. Sa statusom nedostupnosti bit će zabilježeno koja iznimka je uzrokovala da se komentari i članak nisu mogli dohvatiti. Konfigurabilna podjela omogućuje da se bez promjene izvornog koda nosi s promjenama u sustavima komentiranja.

4.5.3. Hvatanje iznimki

Iznimke definirane u prethodnom potpoglavlju hvata `ArticleFetcher`. Dobrom organizacijom koda u `ArticleFetcher` razredu, odnosno metodi `fetch_comments`, postignuto je da postoji jedinstvena linija koda koja može uzrokovati iznimke vezane uz sustave komentiranja. Radi se o liniji koja instancira razred sustava komentiranja pomoću tvornice. Rukovanje iznimkama u `ArticleFetcher` izvedeno je tako da se diferencira prvenstveno između iznimki izvedenih iz `ArticleFatalError` i ostalih iznimki, ali i između tranzijentnih i perzistentnih. Bitno je napomenuti da se iznimke izvedene iz `ArticleFatalError` zbog svoje naravi smatraju uvijek perzistentnima. Za iznimke izvedene iz nefatalnih iznimki vrši se provjera jesu li tranzijentne ili perzistentne ovisno o tome što piše u konfiguraciji. Ispis 4.18 prikazuje logiku rukovanja iznimkama u `ArticleFetcher` razredu.

```
try:
    article = ArticleFactory.create_article(comment_type, current_url,
                                  canonical_url, self.user_agent, published)

    article_dict = article.get_as_dict()
    article_dict['fetch_status'] = self.fetch_status_ok

    self.mongo_collection.insert_one(article_dict)
    self.log.info("Saved article %s. Status: %s",
                  article_msg['url'], article_dict['fetch_status'])

except ArticleError as article_error:
    self.handle_non_fatal_exception(article_error,
                                    method.delivery_tag,
                                    message)
    return

except ArticleFatalError as article_fatal_error:
    self.log.error("Could not fetch article with URL %s. Error message: %s",
                           article_msg['url'], article_fatal_error.message)

    article_dict = {'fetch_status': self.fetch_status_fatal,
                    'url': article_msg['url'],
                    'comment_type': comment_type,
                    'exception': article_fatal_error.message}

    self.mongo_collection.insert_one(article_dict)
    self.channel.basic_ack(method.delivery_tag)

    return
```

Ispis 4.18: Logika rukovanja iznimkama u ArticleFetcher razredu

Bitan dio logike je da se iznimke izvedene iz `ArticleError` dalje obrađuju, dok se sve iznimke izvedene iz `ArticleFatalError` zapisuju u bazu kao nedostupne i ne šalju se na ponovni pokušaj. Ispis 4.19 prikazuje metodu koja rukuje nefatalnim iznimkama.

```
def handle_non_fatal_exception(self, exception, delivery_tag, message):
    article_msg = json.loads(message.decode('utf-8'))

    url, canon_url, comment_type = self.extract_article_creation_info(
        article_msg)

    if self.is_persistent(exception):
        self.log.error("Persistent error detected: %s.\nSaving as
            unavailable", exception.message)
        article_dict = {'fetch_status': self.fetch_status_pers,
                        'url': canon_url,
                        'comment_type': comment_type,
```

```
                    'exception': exception.message
                    }
        self.mongo_collection.insert_one(article_dict)
        self.channel.basic_ack(delivery_tag)
    else:
        could_retry = self.retry_transient(delivery_tag, message)
        if not could_retry:
            self.log.error("Article could not be fetched with retries.")

            self.retry_log[article_msg['url']] = 0

            article_dict = {'fetch_status': self.fetch_status_trans,
                            'url': canon_url,
                            'comment_type': comment_type,
                            'exception': "Maximum retries reached."
                            }
            self.mongo_collection.insert_one(article_dict)
            self.channel.basic_ack(delivery_tag)
```

Ispis 4.19: Logika rukovanja nefatalnim iznimkama

Prvo se provjerava je li iznimka perzistentna te ako jest, sprema se zapis za taj članak u bazu i označuje kao nedostupan s odgovarajućim informacijama zbog čega je to tako. U slučaju da je iznimka tranzijentna, šalje se metodi za ponovni pokušaj. Metoda za ponovni pokušaj vraća istinitu vrijednost ako je uvjet za ponovni pokušaj zadovoljen. Ako vrati neistinitu vrijednost, sprema se zapis za taj članak u bazu te označuje kao privremeno nedostupnim. Ispis 4.20 prikazuje metodu za ponovni pokušaj.

```
def retry_transient(self, delivery_tag, message):
    article_msg = json.loads(   message.decode('utf-8'))

    if article_msg['url'] not in self.retry_log:
        self.retry_log[article_msg['url']] = 0

    self.log.debug("Retrying URL %s.", article_msg['url'])

    if self.retry_log[article_msg['url']] < self.MAX_RETRIES:
        delay = self.initial_delay * (self.delay_multiplier ** self.
            retry_log[article_msg['url']])

        self.requeue_msg(delivery_tag, message, delay)
        self.retry_log[article_msg['url']] += 1
        return True
    else:
        return False
```

Ispis 4.20: Metoda za ponovni pokušaj obrade članka

U metodi za ponovni pokušaj prvo se provjerava je li već dosegnut maksimalni broj pokušaja za taj članak te ako jest, vraća se neistinita vrijednost. Ako nije, vraća se poruka za obradu tog članka u red s vremenskim odmakom koji eksponencijalno raste svakim novim pokušajem.

4.6. Sustav automatskog testiranja

Osim što sustav ima razvijen mehanizam detekcije promjena tijekom normalnog rada, potrebno je osigurati i testove koji će u odnosu na poznate članke provjeriti je li se išta promijenilo ili u načinu dohvata komentara ili u formatu dohvaćenih komentara. Naime, u slučaju uspješnog dohvaćanja komentara potrebno je osigurati da su komentari u formatu kojeg ostatak sustava očekuje. Kako bi se ta dva zahtjeva zadovoljila,

implementiran je sustav automatskog testiranja. Testiranje dohvata članka i komentara izvedeno je u skripti `validate.py`. Njena odgovornost je da pročita URL-ove iz konfiguracije i pokuša instancirati odgovarajuće `Article` razrede. U slučaju uspješnog instanciranja, odnosno ako se nije pojavila iznimka, komentari su uspješno preuzeti. Ispisi 4.21 i 4.22 prikazuju ključni odsječak izvornog koda skripte za testiranje. Preskočen je dio čitanja konfiguracije.

```
tests = {'date': datetime.now()}

status = 'OK'
message = None

passed_tests = 0
total_tests = 0
for section in config:
    if len(comment_systems) > 0 and section not in comment_systems:
        continue

    log.info("Validating comment system - %s", section)

    format_validator = get_format_validator(section)

    section_tests = {}
    counter = 0

    for key in config[section]:
        val = config[section][key]
        if counter > MAX_URLS:
            break

        url_test = {'url': val}

        log.debug("Validating URL %s", val)
        total_tests += 1

        web_page = requests.get(val)
        if web_page.status_code != 200:
            log.error("validation: ERROR url " + val + "\n returns code
                other than 200")
            status = 'FAILED'
            message = 'Could not access URL - request response code {}'.
                format(web_page.status_code)
            url_test['status'] = status
            url_test['message'] = message

            section_tests[key] = url_test

            continue
```

Ispis 4.21: Izvorni kod skripte validate.py

```
        try:
            article = ArticleFactory.create_article(section, val)
        except BaseException as exc:
            log.error("Article creation failed: %s", exc.message)
            status = 'FAILED'
            url_test['status'] = status
            url_test['message'] = exc.message

            section_tests[key] = url_test

            continue

        try:
```

```
            format_validator.validate(article)
        except FormatValidationError as fvErr:
            log.error("Format validation failed: %s", fvErr.message)
            status = 'FAILED'
            url_test['status'] = status
            url_test['message'] = fvErr.message

            section_tests[key] = url_test

            continue

        url_test['status'] = 'OK'
        section_tests[key] = url_test
        passed_tests += 1

    tests[section] = section_tests


log.info("Testing completed.")
log.info("Test results: %s/%s", passed_tests, total_tests)
mongo_collection.insert_one(tests)

log.info("Test results saved to database.")
```

Ispis 4.22: Izvorni kod skripte validate.py (nastavak)

Skripta iterira nad sekcijama konfiguracije pri čemu svaka sekcija odgovara jednom sustavu komentiranja. Svaka sekcija sadrži jedan ili više URL-ova na članke za koje je poznato da sustav uredno dohvaća komentare. Zatim se pripremi struktura podataka za rezultate testiranja trenutnog sustava komentiranja, a i instancira se odgovarajući `FormatValidator` o čemu će više riječi biti kasnije.
Sljedeće se iterira nad svakim od zapisa u sekciji, odnosno nad svim URL-ovima za taj sustav komentiranja. Za svaki od URL-ova se dohvaća HTML članka, pokušava se instancirati `Article` razred te se na kraju poziva instancirani `FormatValidator`.

Instanciranje odgovarajućeg `FormatValidator` razreda izvršava se na isti način kao i za `Article` razrede. Ispis 4.23 prikazuje metodu `get_format_validator` koja služi vraćanju odgovarajuće instance `FormatValidator` razreda u odnosu na tip komentara predan kao argument metodi.

```
def get_format_validator(comment_type: str):
    format_validator_module = "checks.comment_validators.FormatValidator" +
        comment_type.capitalize()
    format_validator_cname = "FormatValidator" + comment_type.capitalize()
    format_validator_class = getattr(importlib.import_module(
        format_validator_module), format_validator_cname)

    return format_validator_class()
```

Ispis 4.23: Metoda za dohvat instance FormatValidator razreda

`FormatValidator` je apstraktni razred koji predstavlja osnovu za više konkretnih razreda koji odgovaraju jednom sustavu komentiranja. Tako postoji `FormatValidatorFacebook`, `FormatValidatorDisqus`, `FormatValidatorVecernji` te `FormatValidator24sata`. Apstraktni razred `FormatValidator` ima jednu metodu `validate(self, article)` koja prima članak s komentarima u obliku rječnika. Konkretna implementacija ostavlja se izvedenim razredima kako bi mogli testirati specifičnosti pojedinih sustava komentiranja. Ispis 4.24 prikazuje konkretnu implementaciju metode `validate` u razredu `FormatValidatorDisqus`.

```
def validate(self, article):
    article_dict = article.get_as_dict()
    users = article.get_commenting_users()
    if article_dict['published'] is None:
        self.log.error("Published property is None.")
        raise DateNotFoundError("Published property is None.")
    elif article_dict['comments'] is None or len(article_dict['comments'])
        == 0:
        self.log.error("No comments collected.")
        raise NoCommentsError("No comments have been collected.")
    elif users is None or len(users) == 0:
        self.log.error("No users parsed.")
        raise NoUsersError("No users could be parsed.")
    else:
        self.log.debug("Parsing users... ")
        for user in users:
            try:
                for attr in self.user_attrs:
                    self.log.debug("[ %s : %s ]", attr, user[attr])
            except KeyError:
                self.log.error("Key error occurred. No key %s in user.",
                    attr)
                raise UserAttributesError("Key error for {} {}".format(attr,
                     user))
```

Ispis 4.24: Implementacija metode validate

U implementaciji metode `validate` prikazane u ispisu 4.24 prvo se poziva metoda `get_as_dict` nad dobivenim člankom. Time se postiže da je članak u obliku rječnika kojim se može dalje baratati. Zatim se testiraju ključevi koje sustav očekuje u rukovanju s člancima tipa Disqus. U slučaju da nešto nije u redu, diže se odgovarajuća iznimka kako bi dijagnostika problema bila lakša.
Nakon što skripta `validate` završi s postupkom, rezultati testiranja spremaju se kao rječnik u bazu podataka. Ispis 4.25 prikazuje oblik u kojem se spremaju rezultati testiranja.

```
{
    'date': ...,
    'facebook': {
        "https://jutarnji.hr/...." : {
            status: 'PASSED'
        },
        "https://index.hr/"..." : {
            status: 'FAILED',
            message: 'Could not parse Date...'
        }
    },
    'disqus': {
        ...
    },
    'vecernji': {
        ...
    },
    ...
}
```

Ispis 4.25: oblik rezultata testiranja

Bilježi se datum testiranja te rezultat za svaki od URL-ova. Skripta međutim bilježi u *log* i broj izvršenih testova s brojem prolaza.

4.7. Sustav učitavanja konfiguracije

Uloga modula `config_loadera` objašnjena je u poglavlju o arhitekturi sustava. Međutim, postoji implementacijski dio koji je zanimljiv, a to je uporaba ubacivanja za-

visnosti (eng. *dependency injection*) kako bi svaki od modula koji trebaju učitavati konfiguraciju lako mogli dobiti jedan od dva objekta, ovisno o tome što je zadano kao parametar pri pokretanju sustava. Kako bi se to postiglo, koristi se radni okvir `dependency_injector`. Prvo se definira *container* koji će sadržavati definicije objekata koji se moraju dohvaćati. Tu je moguće definirati je li riječ o kreiranju objekata, odnosno tvornici, ili jedinstvenom objektu, odnosno *singletonu*. Ispis 4.26 prikazuje razred `Container` koji sadrži definiciju dva singletona koji odgovaraju dvaju `ConfigLoader` razreda.

```
class Container(containers.DeclarativeContainer):
    dbConfigLoader = providers.Singleton(
        DbConfigLoader
    )
    fileConfigLoader = providers.Singleton(
        FileConfigLoader
    )
```

Ispis 4.26: Implementacija containera

Kako bi se mogla dohvatiti instanca jednog od `ConfigLoadera`, potrebno je samo pristupiti `Containeru` s nazivom objekta. Ispis 4.27 prikazuje isječak iz `__init__` metode `ArticleDisqus` razreda kojom se dohvaća instanca `ConfigLoadera` koja odgovara onom koji je postavljen prilikom pokretanja sustava.

```
self.config = getattr(Container, ConfigLoader.loader_name)().load(self.
    __class__.__name__)
```

Ispis 4.27: Poziv za dohvat instance `ConfigLoadera`

4.8. Vođenje dnevnika

Vođenje dnevnika (eng. *logging*) je način praćenja svih događaja u sustavu. Služi praćenju rada te dijagnostici pri kvarovima. Pomaže pronaći odgovor na pitanje u kakvom je stanju sustav bio prije kvara i u trenutku kvara [9]. U slučaju ovog rada, vođenje dnevnika predstavlja veliku pomoć pri praćenju brzine dohvata članaka i komentara, ali i pri praćenju kvarova. Vođenje dnevnika je implementirano korištenjem ugrađene (eng. *built-in*) biblioteke u Pythonu nazvane `logging`. Vođenje dnevnika razlikuje poruke prema razinama bitnosti, a Python implementacija sadrži sljedeće razine: `DEBUG`, `INFO`, `WARNING`, `ERROR` te `CRITICAL` [5].

Bitan dio vođenja dnevnika u Pythonu je poznavanje pojmova *handler* i *logger*. `Logger` je objekt nad kojim se pozivaju metode vođenja dnevnika, odnosno `debug(), info()` i ostale [11]. U ovom radu napravljeno je da svaki modul ima vlastiti logger. Razlog tomu je što se bitnost događaja u pojedinim modulima međusobno razlikuje te je potrebno održavati različite postavke vođenja dnevnika za različite module. Dalje, handleri su objekti odgovorni za vođenje poruka do definiranih odredišta. Postoje predefinirani handleri za nekoliko čestih obrazaca korištenja. Tako postoji `ConsoleHandler` koji ispisuje poruke u konzoli, `FileHandler` koji ispisuje u datoteku te `SMTPHandler` koji šalje email s porukama [11]. Postoje i mnogi drugi. U ovom radu korišteni su

`FileHandler` i `ConsoleHandler`. Razlog tomu je što bi korištenje isključivo ispisa u konzoli rezultiralo nepreglednim nizom informacija. Korištenje dodatnog `FileHandlera` dopušta da se manje bitne informacije ne gube već zapisuju u odgovarajuće datoteke, odnosno da svaki razred ima vlastitu datoteku s opširnim logovima.

Tijekom pisanja implementacije sustava na velikom broju mjesta pronašli su se jednostavni ispisi na standardni izlaz poput naredbe `print`. Problem s tom praksom je što ne postoji utjerivanje jedinstvenog formata ispisa te ispis nije konfigurabilan. Kako bi se to riješilo, sve što je trebalo davati ispis je pretvoreno u log. Velika prednost toga je što je ispis uniforman i značajno lakši za pratiti vizualno.

4.9. Implementacijski detalji dohvata komentara

U ovom potpoglavlju bit će objašnjeno na koji način je implementiran postupak opisan u poglavlju 2.1. Metoda koja je srž logike dohvata komentara u razredu `ArticleFacebook` je `__collect_comments(self)`. Metoda radi sve što je opisano u poglavlju 2.1, odnosno dohvaća identifikator članka iz početnog GET zahtjeva, parsira ga te gradi POST zahtjev koji će dohvatiti JSON s komentarima. Ispis 4.28 prikazuje kod metode `__collect_comments`.

```
def __collect_comments(self):

    target_id = self.get_article_id()

    target_url = 'https://www.facebook.com/plugins/comments/async/' +
        target_id + '/pager/time/'
    r = self.__send_request(target_url=target_url, data={'__a': 1, 'limit':
        MAX_COMMENTS}, method="POST")
    if r is None or r.status_code != requests.codes.ok:
        self.log.error("Cannot retrieve comment JSON as response to URL: %s"
            , target_url)
        raise AsyncFetchError("Cannot retrieve comment JSON.")

    comments = json.loads(r.text[COMMENT_START_INDEX:])
    self.id_map = comments["payload"]["idMap"]
    self.article_id = target_id
```

Ispis 4.28: Metoda za dohvat komentara

Prvi korak je dohvat i parsiranje identifikatora članka pozivom metode `get_article_id (self)`. Ispis 4.29 prikazuje metodu za dohvat i parsiranje identifikatora članka.

```
def get_article_id(self):
    initial_comment_url = self.article_id_url.format(quote(self.
        canonical_url))

    article_id_list = []

    for i in range(self.article_id_max_retries):

        response = requests.get(initial_comment_url, headers=self.headers)
        if response is None or response.status_code != requests.codes.ok:
            self.log.error("Cannot fetch initial comment page.")
            raise FacebookInitialRequestError("Cannot fetch initial comment
                page.")

        article_id_list = re.findall(self.reg_fbid, response.text)

        if len(article_id_list) == 1:
```

```
            break

        self.__change_headers()

        self.log.info("Could not fetch article id. Sleeping and retrying in
            %s.", self.article_id_retry_sleep)
        time.sleep(self.article_id_retry_sleep)

    if len(article_id_list) != 1:
        self.log.error("Cannot parse article ID for URL %s. "
                       "Looking for %s in response text of request URL: %s."
                       ,
                       self.url, self.reg_fbid, initial_comment_url)
        raise FacebookArticleIdParseError("Cannot parse article ID.")

    return article_id_list[0]
```

Ispis 4.29: Metoda za dohvat i parsiranje identifikatora članka

Metoda prikazana u ispisu 4.29 implementirana je tako da pokušava dohvatiti sadržaj URL-a za dohvat identifikatora članka, odnosno inicijalni skup komentara. U slučaju neuspjeha, mijenja zaglavlja i čeka određeno vrijeme prije ponovnog pokušaja. U slučaju uspješnog parsiranja vraća identifikator članka. Nakon što je uspješno vraćen identifikator članka metoda `__collect_comments` šalje POST zahtjev kojim se dohvaćaju komentari te se u slučaju neuspjeha diže iznimka. U suprotnom se postavljaju atributi instance `id_map` te `article_id`. Slanje POST zahtjeva prikazano je u ispisu 4.28.

5. Prikaz rada sustava

Ovo poglavlje pokazat će na koji način se sustav pokreće te će prikazati rad sustava.

5.1. Način pokretanja sustava

Sustav se primarno pokreće pokretanjem skripte `run.py` koja prima opcije. Moguće je zadati opcije `-A`, `-M` te `-R` koje služe pokretanju određenih dijelova sustava, odnosno `ArticleFetcher`, `NewsMonitor` te `RefreshComments`. Pokretanje samo `ArticleFetcher` ima smisao samo ako je u prethodnim pokretanjima barem jednom bio pokrenut `NewsMonitor` koji će poslati naredbe za obradu članaka u red. Naravno, pretpostavka je da se sadržaj reda nije obrisao. U sljedećim ispisima bit će pokazan komandno-linijski ispis generiran tijekom rada sustava u nekoliko načina. Ispis 7.1 prikazuje logove stvorene u prvih nekoliko sekundi rada sustava sa samo `ArticleFetcherom` uključenim.

```
2021-06-18 10:48:27,229 - ArticleFetcher - INFO - Waiting for tasks.
2021-06-18 10:48:35,226 - ArticleFetcher - INFO - Saved article
    https://www.novilist.hr/sport/majstorica-za-naslov-premijer-lige-
    zadar-i-split-igraju-odlucujucu-petu-utakmicu-za-prvaka-hrvatske
    /. Status: 'available'
2021-06-18 10:48:38,955 - ArticleFetcher - INFO - Saved article
    https://www.novilist.hr/sport/braci-sinkovic-se-u-finalu-dvojaca-
    na-svjetskom-kupu-pridruzila-i-braca-patrik-i-anton-loncaric/.
    Status: 'available'

2021-06-18 10:48:43,408 - ArticleFetcher - INFO - Saved article
    https://www.novilist.hr/novosti/hrvatska/koronavirus-u-hrvatskoj-
    danas-211-novih-slucajeva-preminulo-jos-13-osoba/. Status: '
    available'
2021-06-18 10:48:47,402 - ArticleFetcher - INFO - Saved article
    https://www.novilist.hr/novosti/svijet/starica-je-docekala-pravdu
    -vlasti-su-jutros-pocele-s-rusenjem-crkve-koja-je-nelegalno-
    izgradena-u-dvoristu-fate-orlovic/. Status: 'available'
2021-06-18 10:48:52,072 - ArticleFetcher - INFO - Saved article
    https://www.novilist.hr/sport/uefa-odlucila-smanjiti-nagradni-
    fond-za-ovogodisnji-euro-no-i-dalje-je-najveci-u-povijesti-
    natjecanja/. Status: 'available'
```

Ispis 5.1: Ispis na konzoli prilikom pokretanja sustava s opcijom `-A`

Ispis 7.2. prikazuje logove stvorene u prvih nekoliko sekundi rada sustava sa samo `NewsMonitorom` uključenim. U logovima je vidljiv ispis samo `DownloadURL` razreda. Razlog tomu je što `NewsMonitor` ima ispise samo za iznimke.

```
2021-06-18 10:52:25,641 - DownloadURL - INFO - Adding URL https://
    www.index.hr/magazin/clanak/fotke-iz-arhive-ovako-je-vanja-na-
    euru-navijala-za-modrica-dok-nisu-bili-u-braku/2284408.aspx (
    comment type facebook) with delay 14400.0 s
2021-06-18 10:52:25,641 - DownloadURL - INFO - Adding URL https://
    www.index.hr/vijesti/clanak/rastu-cijene-kulena-kilogram-bi-mogao
    -biti-skuplji-za-100-kuna/2284397.aspx (comment type facebook)
    with delay 14400.0 s
2021-06-18 10:52:25,641 - DownloadURL - INFO - Adding URL https://
    www.index.hr/vijesti/clanak/sefa-u-gskgu-i-gradjevinara-koji-mu-
    je-davao-mito-spojio-je-bandicev-tjelohranitelj/2284413.aspx (
    comment type facebook) with delay 14400.0 s
2021-06-18 10:52:25,641 - DownloadURL - INFO - Adding URL https://
    www.index.hr/sport/clanak/thiem-propusta-olimpijske-igre/2284412.
    aspx (comment type facebook) with delay 14400.0 s
2021-06-18 10:52:25,641 - DownloadURL - INFO - Adding URL https://
    www.index.hr/vijesti/clanak/zagrebacka-zupanija-ima-sedam-novih-
    slucajeva-zaraze/2284410.aspx (comment type facebook) with delay
    14400.0 s
2021-06-18 10:52:25,641 - DownloadURL - INFO - Adding URL https://
    www.index.hr/vijesti/clanak/grcki-pilot-priznao-ubojstvo-mlade-
    britanke-udarila-me-i-rekla-da-ce-me-ostaviti/2284406.aspx (
    comment type facebook) with delay 14400.0 s
```

Ispis 5.2: Ispis na konzoli prilikom pokretanja sustava s opcijom `-M`

Način rada sa samo uključenim `RefreshComments` neće biti prikazan zbog toga što je ispis identičan ispisu koji je dobiven pokretanjem samo `NewsMonitora`.

5.2. Pokretanje testiranja

Testiranje se može pokrenuti izvršavanjem skripte `validate.py` koja prima argumente. Kao argument moguće je poslati naziv sustava komentiranja koji se žele testirati. Ako se izostave argumenti, svi sustavi bit će testirani. Ispis 7.3 prikazuje rezultat pokretanja validacijske skripte bez argumenata.

```
2021-06-18 10:54:33,011 - Validate - INFO - Validating comment
    system - Facebook
2021-06-18 10:54:38,736 - Validate - INFO - Validating comment
    system - Vecernji
2021-06-18 10:54:45,630 - Validate - INFO - Validating comment
    system - 24sata
2021-06-18 10:54:51,231 - Validate - INFO - Validating comment
    system - Disqus
2021-06-18 10:54:53,549 - Validate - ERROR - Article creation failed
    : The request produced no viable response. Try again later or
    check manually.
2021-06-18 10:54:56,106 - Validate - ERROR - Article creation failed
    : Cannot parse date published from HTML code.
2021-06-18 10:54:57,526 - Validate - ERROR - Article creation failed
    : The request produced no viable response. Try again later or
    check manually.
2021-06-18 10:54:57,541 - Validate - INFO - Testing completed.
2021-06-18 10:54:57,541 - Validate - INFO - Test results: 7/10
2021-06-18 10:54:57,587 - Validate - INFO - Test results saved to
    database.
```

Ispis 5.3: Ispis na konzoli prilikom pokretanja testiranja bez argumenata

Vidljivi su rezultati testiranja, ali i iznimke koje su se dogodile tijekom testiranja. Prema ovim rezultatima vidljivo je da se moraju bolje pogledati logovi vezani uz

Disqus sustav kako bi se sustav prilagodio promjeni ako je potrebno. Slika 7.4 prikazuje rezultat pokretanja validacijske skripte s argumentima.

```
2021-06-18 10:59:14,120 - Validate - INFO - Validating comment
   system - Facebook
2021-06-18 10:59:19,350 - Validate - INFO - Validating comment
   system - Vecernji
2021-06-18 10:59:26,724 - Validate - INFO - Testing completed.
2021-06-18 10:59:26,724 - Validate - INFO - Test results: 5/5
2021-06-18 10:59:26,751 - Validate - INFO - Test results saved to
   database.
```

Ispis 5.4: Ispis na konzoli prilikom pokretanja testiranja za sustave Facebook i Vecernji

6. Zaključak

Izgradnja sustava za dohvat članaka i komentara zahtjevan je zadatak iz gledišta teorije, ali i implementacije. Teorijski dio rada uključivao je primjenu reverznog inženjerstva kako bi se otkrili unutarnji načini rada sustava za komentiranje na webu, a zatim i obradu te primjenu saznanja kako bi se teorija prilagodila zahtjevima implementacije. Izazov navedenog zadatka je u nedostatku izvora koji bi ubrzali proces. Sve što je otkriveno o sustavima tijekom rada otkriveno je samostalnim istraživanjem i eksperimentiranjem nad sustavima za komentiranje tijekom njihova rada. Vrijednost dobivenih saznanja je u njihovoj rijetkosti i spremnosti za korištenje u budućim implementacijama ili nadogradnjama na postojeći sustav.

Implementacija sustava za dohvat članaka i komentara predstavlja izazov jer nastoji pretočiti teorijska saznanja o sustavima komentiranja u kod uz težnju da izgrađeni sustav prati najbolje prakse programskog inženjerstva. Iterativno tijekom implementacije zaustavljao se rad na novim značajkama sustava te se napor usredotočio na poboljšavanje robusnosti sustava i refaktoriranje u svrhu lake nadogradnje i bolje čitljivosti. Svaka opcija implementacije nove mogućnosti sustava odvagnuta je u odnosu na ostale te se odabir provodio tako da sustav nakon implementacije ne bude kompleksniji nego li mora biti. U tome se nalazi velik izazov koji je implementacijski dio rada nastojao riješiti.

Krajnji rezultat rada je ostvarenje sustava koji ispunjava svoju svrhu u potpunosti, a uz to je lako nadogradiv, robusan i jednostavan za održavati.

LITERATURA

[1] Martin Fowler. *Refactoring - Improving the Design of Existing Code*. Addison-Wesley Professional, 1 izdanju, 1999. ISBN 9780201485677,0201485672.

[2] 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-352ba5d [Online; pristup ostvaren 15.6.2021].

[3] Daniel Ha, 2011. URL https://blog.disqus.com/the-numbers-of-disqus. [Online; Pristup ostvaren 22.6.2021.

[4] Nick Huss. How many web sites are there around the world, Svibanj 2021. URL https://siteefy.com/how-many-websites-are-there/. [Online; Pristup ostvaren 17.6.2021].

[5] Son Nguyen Kim. *Python Logging: An In-Depth Tutorial*, 2021. URL https://www.toptal.com/python/in-depth-python-logging. [Online; Pristup ostvaren 17.6.2021].

[6] Ying Lin. 10 internet statistics every marketer should know in 2021, 2021. URL https://www.oberlo.com/blog/internet-statistics. [Online; Pristup ostvaren 17.6.2021].

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

[8] 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].

[9] Sid Panjwani. *The Pythonic Guide To Logging*. URL https://timber.io/blog/the-pythonic-guide-to-logging/. [Online; Pristup ostvaren 17.6.2021].

[10] Martin Perez. What is web scraping and what is it used for?, 2019. URL https://www.parsehub.com/blog/what-is-web-scraping. [Online; Pristup ostvaren 18.6.2021].

[11] Vinay Sajip. *Logging HOWTO*, 2021. URL https://docs.python.org/3/howto/logging.html. [Online; Pristup ostvaren 17.6.2021.

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

[13] Alexander Shvets. Refactoring guru: Extract class. URL https://refactoring.guru/extract-class. [Online; Pristup ostvaren 16.6.].

[14] Unknown. What is a web crawler bot? URL https://www.cloudflare.com/learning/bots/what-is-a-web-crawler/. [Online; Pristup ostvaren 18.6.2021].

Biblioteka za dohvat novinskog članka s portala i pridruženih komentara na društvenim mrežama

Sažetak

U sklopu ovog rada otkriven je unutarnji način rada nekolicine sustava za komentiranje na webu te je na osnovu tih saznanja implementiran sustav za dohvat članaka i njihovih komentara. Krajnji cilj rada je stvoriti web *scraping* sustav s elementima web pauka koji ispunjava svoju svrhu, a da je lako nadogradiv, robusan te lak za održavati. Odnosno, potrebno je osigurati da je implementacija provedena u skladu s dobrim praksama programskog inženjerstva. Kako bi se moglo reći da sustav ispunjava svoju svrhu, mora moći samostalno preuzimati članke i komentare u odnosu na definirana ograničenja te održavati rad bez obzira na vanjske promjene sustava komentiranja na webu. U radu je prikazan postupak reverznog inženjerstva sustava komentiranja, arhitektura razvijanog sustava te implementacijski detalji bitni za razumijevanje najvećih problema koje je ovaj rad riješio.

Ključne riječi: reverzno inženjerstvo, web pauk, web scraper, programsko inženjerstvo, dobre prakse

Library for retrieving news articles and associated comments on social networks

Abstract

This paper serves to explain the inner-workings of comment systems used on the web, and based upon that theoretical background goes through the process of building a system which retrieves articles and associated comments from news sites. The end goal of this paper is to build an easily-upgradeable, robust and easy-to-maintain web scraping and crawling system which fulfills its purpose. In other words, good practices must be enforced during the whole implementation life-cycle. The criteria which tells us if the system has fulfilled it's purpose is when it can automatically download news articles and comments and keep working regardless of outside changes of comment systems it relies on. The paper contains the reverse engineering process required to build a base of knowledge of comment systems, the architecture of the built system, and most importantly the implementation details which are needed to fully understand the problems this paper solved.

Keywords: reverse engineering, web crawler, web scraper, software engineering, best practices