Automatizirano prikupljanje podataka pomoću automatizacije preglednika

Barić, Ante

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ante Barić

Automatizirano prikupljanje podataka pomoću automatizacije preglednika

DIPLOMSKI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ante Barić

Matični broj: 45312/16–R

Studij: ***Informacijsko i programsko inženjerstvo***

Automatizirano prikupljanje podataka pomoću automatizacije preglednika

DIPLOMSKI RAD

Mentor:

Doc. dr. sc. Tonimir Kišasondi

Varaždin, srpanj 2018.

Ante Barić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Cilj rada je prikazati primjenu automatizacije preglednika za automatizirano prikupljanje podataka sa stranica u realnom vremenu. Ispitat će se da li je ovakav način prikupljana podataka prikladan i koliko je primjenjiv u produkcijskim okruženjima i unutar postojećih tehnologija. Primjena automatiziranih preglednika je korisna kod razvoja OSINT platformi (eng. Open Source Intelligence) ili razvoja API-a za sustave koji ne podržavaju sučelje s trećim stranama. Dodatni naglasak rada je na metode koje se koriste u interakciji sa stranicama tijekom prikupljanja podataka i metode za sprječavanje takvih metoda. U sklopu rada su razvijene dvije aplikacije. Jedna je web aplikacija koja poslužuje REST servis, a druga je klijentska koja prikazuje korištenje servisa.

Ključne riječi: automatizacija; javascript; nodejs; servisi; preglednik; twitter; podatci;

Sadržaj

1. Uvod

Moderne aplikacije i stranice poslužene na internetu često rade s velikim brojem podataka. Unatoč tome što su sučelja tih aplikacija vrlo napredna i omogućavaju razne načine pregleda sadržaja često aplikacije pružaju i sučelja u obliku servisa. Najčešće su servisi dani u svrhu razvoja dodatnih funkcionalnosti ili pružanja usluga izvan onih integriranih unutar aplikacije. Internet servisi se mogu bazirati na nekoliko različitih arhitektura od koji će najpopularniji bili spomenuti u kasnijim poglavljima ovoga rada. Standardizirane arhitekture omogućuju laku konzumaciju takvih servisa kroz metode korištene danas u modernom razvoju programskih proizvoda. Iako su internetski servisi popularni oni nizu i nužni. Tako se onda često može dogoditi da neka bitna ili korisna stranica nema izložene servise pa je u teoretski teže doći do podataka programskim putem, a svakako teže ako to želimo napraviti automatizirano ili napisati program koji automatski to radi. Popularna metoda prikupljanja podataka je svakako programiranje automatiziranih robota (eng. Crawler) koji posjećuju stranice i prikupljaju određene podatke na određenoj vremenskoj bazi. Sličnu metodu su u početku koristili i djelomično i danas koriste internetske tražilice kao npr. Google i razni indeksi podataka. Jedna manja mana takvog načina je vremensko odstupanja podataka gdje roboti ne mogu u realnom vremenu prikupljati podatke i ažurirati bazu podataka tražilice. Naravno danas postoje i nove metode koje se koriste te to odstupanje svedeno na minimum, a svakim danom se napreduje u tom cilju. Internet preglednik je glavni način konzumacije Internet stranica i aplikacija dostupan prosječnom korisniku. Isto tako programeri Internet stranica koriste preglednike za testiranje stranica koje razvijaju. U tu svrhu autori Internet preglednika integriraju razvojne alate koji pružaju pomoć u pronalaženju i otklanjanju neispravnosti u funkcionalnostima i prikazu stranice tijekom razvoja. Takvi alati su često otvoreni za samostalnu nadogradnju. Tako Internet preglednik Google Chrome (Google, 2018a) pruža protokol pod nazivom DevTools. Protokol pruža način za integraciju i pristup alatima za razvojne programere. Google u zadnjih par godina razvija biblioteku Puppeteer za programske jezike od kojih je jedan JavaScript (Mozilla, 2018). Preko te biblioteke je moguće automatizirati i simulirati ponašanje normalnog korisnika koji pregledava Internet stranice kroz preglednik, ali samo što to u ovom slučaju radi program samostalno. Najbolja i najlošija stvar u tome je da takav pristup stranici i serveru koji je poslužuje izgleda isto kao i normalan (ljudski) korisnik. Tako je onda moguće zaobići razne mehanizme koji raspoznaju ponašanje robota od čovjeka. Loša strana je da navigacija kroz stranicu tako uključuje više zahtjeva prema serveru pa je potrebno uložiti vrijeme u optimizaciju kako bi pristup bio što efikasniji. Ovakav način isto tako

izlaže sve podatke prikazane na stranici i možemo im pristupiti kroz HTML DOM i lagano spremiti u varijable i obrađivati tijekom izvršavanja programa. Kada imamo podatke u memoriji možemo ih lako izložiti na drugim mjestima pa ovakav način pruža mogućnost da iste proslijedimo kroz programsko sučelje i poslužimo dalje kao svaki normalan Internet servis. Iako ovakav način prikupljanja podataka nije u produkcijskoj fazi daljnjim razvitkom i razvojem standarda kao što su HTTP2 (Wikipedia, 2018a) koji smanjuju broj zahtjeva prema servisima ovakav način automatiziranog prikupljanja podataka bi mogao vrlo dobro riješiti probleme navedene na početku ovog poglavlja. Cilj ovog rada je izraditi programski okvir koji bi automatizirano prikupljao podatke neke Internet stranice te ih posluživao kroz standardni REST (eng. Representational State Transfer) Internet servis. Drugi cilj je analizirati trenutne implementacije, zaštitu od ovakvih metoda pristupa i moralnu i legalnu stranu ovakvog načina prikupljanja i posluživanja podataka.

2. Podaci na internetu

Osim Internet stranica i HTTP protokola (eng. Hypertext Transfer Protocol) podaci na internetu mogu biti posluživani i na druge načine i kroz druge protokole. Većina tih protokola je dostupa kroz programske jezike i postoje biblioteke i metode za slanje i primanje podataka ovisno o protokolu. Nisu svi podatci na internetu javno dostupni svima i za pristup resursima nekada je potrebna neka vrsta ovjere identiteta korisnika. Ovjera autentičnost ne pomaže u pristupu podatcima u pogledu automatiziranju prikupljanja, ali ako posjedujemo podatke i pristupamo računima koje posjedujemo moguće je automatizirati i dio taj korak.

2.1. Vrste dokumenata

Internet stranice su posebna vrsta dokumenta i pisane su u HTML (eng. Hypertext Markup Language) jeziku, ali mogu sadržavati ugrađene druge dokumente kao što su video formati (.mp4, .mkv i slični), slike (.jpg i slični) a isto tako tekstualni dokumenti (.pdf, .docx i slični) . Dokumenti iz stranice su dostupni na odvojenoj adresi koja je dohvatljiva iz HTML dokumenta. Dokumenti mogu biti posluženi i preko FTP (eng. File Transfer Protocol) protokola ili dediciranog CDN-a (eng. Content Delivery Network) .

2.2. Protokoli

Kako je već prije spomenuto HTTP protokol je glavni protokol za posluživanje dokumenata na internetu. Trenutno se nalazimo u prelaznom periodu gdje sljedeća verzija protokola HTTP2 stupa na snagu i povećava zastupljenost. HTTP2 (Lexy Mayko, 2015) pruža optimizacije u pogledu bolje kompresije sadržaja, kontinuirane veze za razmjenu datoteka i podataka i slično. Protokoli na internetu su podijeljeni na više slojeva:

- Aplikacijski sloj
- Transportni sloj
- Mrežni sloj
- Podatkovni sloj

U pogledu ovog rada zanimljivu su nam protokoli aplikacijskoj sloja kojemu pripadaju i HTTP i HTTP2 protokoli. Neki od ostalih protokola koji se koriste u razmjenu podataka i komunikaciju na internetu su:

- IMAP (eng. Internet Message Access Protoco) i POP3 (eng. Post Office Protocol) – Email i slanje poruka
- LDAP (eng. Lightweight Directory Access Protocol) – Ovjera autentičnost
- Torrent – P2P (eng. Peer to Peer) prijenos podataka
- FTP – prijenos podataka
- SSH (eng. Secure Shell) – udaljena prijava na terminal udaljenog računala

2.3. Ovjera autentičnost

Postoje razni način prijave korisnika u Internet stranice ili aplikacije. Svaki način varira u razini sigurnosti i jednostavnosti implementacije i popularnosti. Neki od popularnijih vrsta ovjere autentičnosti su:

- Basic ovjera autentičnost – jednostavni način ovjere autentičnosti integriran usko uz HTTP protokol. Preglednik može zatražiti korisničko ime i lozinku kod pristupanja određenoj stranici ili resursu.
- Digest ovjera autentičnosti – Slično kao Basic ovjera autentičnost, ali koristi jednostavno Base64 kodiranje te nije sigurna ako se ne koristi zajedno uz TLS (eng. Transport Layer Security) protokol.
- Session ovjera autentičnost – Predstavlja vrstu ovjere autentičnosti gdje umjesto korisničkog imena i lozinke server identificira korisnika preko jedinstvenog identifikatora. Identifikator se obično šalje sigurnim kanalom i sprema na klijentu npr. Internet preglednik.
- Token ovjera autentičnost – Token ovjera autentičnost je slična Session ovjeri, ali se tokeni izdaju na određeni vremenski rok i predstavljaju jedinstvenog korisnika kod svih zahtjeva.
- OAuth ovjera autentičnost – Zadržava tokene kao način predstavljanja, ali oni dolaze od autoriziranog pružatelja gdje onda s tim tokenom korisnik može pristupati resursima koji podržavaju prijavu preko istog poslužitelja.

3. Konzumacija sadržaja na internetu

Glavnina ljudi sadržaj na internetu konzumira kroz internet preglednike. Internet preglednici su programi koji su namijenjeni pregledavanju HTML dokumenata, a i onih drugih ugrađenih u Internet stranice ili podržani od strane preglednika npr. video, pdf i slično.

3.1. Internet preglednici

Danas korisnici računala i laptopa na izbor imaju veliki broj Internet preglednika. Većina tih preglednika je dostupna i na mobilnim uređajima. Sljedeća lista se ipak tiče Internet preglednike za računala i laptope koji su redom prema zastupljenosti (Browser Market Share, 2018) :

- Google Chrome – 60.64%
- Microsoft Internet Explorer – 12.28%
- Mozilla Firefox – 11.73%
- Microsoft Edge – 4.13%
- Safari (Apple) – 3.69%
- Opera (Opera Software) – 1.52%
- Ostali – 6.01%

Korisnici danas od Internet preglednika očekuju poznato sučelje, poznati set funkcionalnosti kao što su knjižne oznake, povijest, pamćenje lozinki, sinkronizacija postavki i podataka i slično. Osim zahtjeva prosječnih korisnika Internet preglednici su danas bitan alat u razvoju bilo koje Internet aplikacije ili stranice, a isto tako bilo kojeg sadržaja na internetu. Razvojni programeri očekuju razne alate koji im pomažu u razvoju proizvoda. Zadnja bitna stvar je sigurnost. Internet preglednici su vrata u Internet za većinu korisnika te je bitno da su korisnici dobro zaštićeni od raznih prijetnji i opasnosti koje dolaze s druge strane. Preglednici implementiraju više razina zaštite i regularnim nadogradnjama održavaju standarde na visokoj razini. Posebna grana informacijske sigurnosti, Sigurnost preglednika (eng. Browser Security) (Wikipedia, 2018b) , se bavi sigurnošću preglednika. Razni projekti kao što su npr. OWASP (eng. Open Web Application Security Project) se bave definiranjem pravila i uputa za izradu sigurnih aplikacija i stranica na internetu. Preglednici neke od tih pravila implementiraju unutar svojih funkcionalnosti kako bi preventivno zabranili posjetu stranica koje potencijalno nisu sigurne. Najbitnija stavka je enkripcija tj. HTTPS protokol koji će ubrzo postati i jedini protokol preko kojega će preglednici omogućavati pregledavanje Internet stranica. Velika tvrtka i

proizvođač najkorištenijeg Internet preglednika Google je već u nekoliko navrata najavila, a od siječnja prošle godine i uvela HTTPS u svim svojim servisima i uslugama na internetu te unutar Chrome preglednika. (SEO Hacker, 2017) .

3.2. Alati unutar Internet preglednika

Najzastupljeniji internet preglednici imaju standardiziran set alata za razvoj. Ti alati se većinom tiču pregleda mrežnog prometa stranice, pregleda DOM-a i HTML strukture, JavaScript konzole koja je ograničena na dokument stranice. Taj set alata minimalno, ali ne i nužno sadržava:

- Elementi – pregled strukture stranice, HTML elemenata, njihovih stilova, događaja, akcija i drugo
- Konzola – JavaScript konzola koja omogućava izvršavanje koda u kontekstu stranice
- Datoteke – Pregled datoteka zahtijevanih od stranice
- Mreža - Pregled mrežnih zahtjeva i odgovora koje stranica kreira i prima
- Memorija – Pregled pred memorije stranice, kolačića, sesije i slično

Osim osnovnih, neki preglednici imaju daleko naprednije funkcionalnosti. U sklopu ovoga rada fokus će biti na alatima u sklopu Chrome Internet preglednika. Osim što je najzastupljeniji on sadrži i zanimljive dodatne alate koji će biti direktno korišteni u sklopu izvedbe praktičnog dijela ovoga rada.

3.3. Programska sučelja

Kada govorimo o programskim sučeljima u sklopu ovog rada govorit ćemo o programskim sučeljima na strani poslužitelja. Takva programska sučelja su poslužena preko HTTP/HTTPS protokola i obično se sastoje od niza točaka tj. Internet adresa. Svaka točka može služiti za različitu stvar, a ovakva programska sučelja danas se najčešće definiraju i izvode prema REST arhitekturi. REST daje standard za pristup resursima i definiranje akcija koje je moguće izvršiti nad njima. Svaki resurs može imati više točaka koje ga se tiču, ali najčešće su to tako zvane CRUD metode (eng. Create Read Update Delete) . Zamislimo da imamo resurs pod nazivom članak (eng. Article) i da želimo definirati CRUD metode za isti, prema REST standardu URL-ovi točaka bi izgledali ovako:

- GET /articles - dohvaća listu resursa tipa članak
- GET /articles/1 - dohvaća 1 članak čiji je identifikator broj 1

- POST /articles - kreira jedan novi članak
- PUT /articles - uređaju ili mijenja sadržaj jednog članka
- DELETE /articles - briše ili isključuje (eng. Soft delete) članak iz baze podataka

Svaka točka je zapravo putanja na serveru koji poslužuje programsko sučelje, a ispred svake putanje napisana je HTTP metoda koja se koristi za odrađivanje pojedine radnje. HTTP metode su korisne jer možemo bez kompliciranja ili dodatnog opisivanja putanje do neke metode semantički preko imena metode implicirati što metoda radi. Tako na primjer znamo da "GET /articles" dohvaća, a ista putanja s POST metodom kreira novi resurs tipa članak. Naravno moguće je za sve koristiti GET metodu, ali takav način ne prati REST standard. Treba napomenuti da je REST standard samo primjer arhitekture programskog sučelja na internetu i postoje i druge arhitekture i tipovi sučelje. Naravno nije uvijek potrebno slijepo se držati jedne arhitekture te ovisno o aplikaciji koja će pristupati sučelju ili bazi podataka koja stoji iza sučelja možemo napraviti iznimke u svrhu jednostavnosti ili olakšavanja rada sa sučeljem. Programska sučelja mogu na razini pojedine metode ili cijelog resursa ograničiti pristup tj. zatražiti ovjeru autentičnosti korisnika. Ovjera autentičnost se implementira pomoću nekog od načina spomenutih ranije u isto imenom poglavlju. Važno je odabrati prikladan i najsigurniji način ovjere identiteta korisnika koji paše u kontekstu određenog programskog sučelja. Možemo zaključiti da programska sučelja na internetu usko prate i koriste funkcionalnosti HTTP/HTTPS protokola što je i glavna prednost ovakvog načina programiranja poslužitelja. S klijentske strane postoje razni način za pristup ovakvim sučeljima. Najčešći način je proširenje HTTP klijenta unutar programskog jezika koji se koristi. Drugi način je kreiranje istog sučelja na strani klijenta koje se onda spaja na poslužitelj i obavlja sve mrežne radnje i prosljeđuje podatke klijentskoj aplikaciji kao nativne tipove podataka u sklopu jezika koji se koristi. Popularne i globalne web aplikacije koje koristimo svaki dan kao što su na primjer Google, Facebook, Twitter, Gmail i drugi dodatno pružaju usluge i kroz programsko sučelje. Najčešće kako bi olakšali programerima integraciju nekih servisa u sklopu njihovih aplikacija. Naravno navedene web aplikacije ne izlažu sve usluge preko programskog sučelja. Uglavnom one najbitnije kao što su npr. kreiranje tweet-ova na Twitter-u ili slanje poruka na Facebook-u spadaju u tu kategoriju iz očiglednog razloga jer bi izlaganjem tih servisa omogućili drugim stranicama pružanje usluga i tako smanjili promet i posjećenost svojih stranica. Kako mnoge internetske stranice, a pogotovo društvene mreže žive od prodaje oglasa i marketinga to bi za njih bio veliki udarac. Postoje i Internet stranice koje svoje usluge pružaju isključivo ili većinskim dijelom preko servisa. Primjer takvih stranica su sustavi za plaćanje (eng. Payment Gateways) koji preko raznih plaćenih pretplata pružaju servise za kreiranje i pregled transakcija, kupovine proizvoda i slično. Postoje iznimke gdje se i najbitnije funkcionalnosti

izlažu preko API-a, ali u tom slučaju pristup ide preko aplikacija koje često imaju određene limite i opet ne pružaju totalnu kontrolu nad takvim sadržajem. Programska sučelja su trenutno prisutna u mnogim aspektima razvoja programskih proizvoda, a kako većina tih proizvoda ima neki "online" aspekt sigurno koriste i programska sučelja koja su opisana u ovom poglavlju. U tu svrhu svaki novi način razvoja programskih sučelja bi trebao biti ispitan u smislu korisnosti te novih načina razvoja.

4. Alati za razvojne programere (Chrome)

U prijašnjim poglavljima je spomenuto da svaki preglednik ima niz alata namijenjenih za olakšavanje razvoja Internet aplikacija i stranica. Razvojni alati preglednika Chrome su srž izvedbe praktičnog dijela ovoga rada. Kroz ovo poglavlje ćemo pregledati te alate i što oni pružaju te kako se koriste u sklopu razvoja i integracije kroz već prije spomenuti DevTools protokol.

4.1. DevTools protokol

Protokol je razvijen kako bi se programskim putem moglo pristupiti značajkama razvojnih alata unutar Chrome preglednika. Protokol je podijeljen u više domena i svaka od njih predstavlja jedan do značajki spomenutih u prijašnjoj analizi alata unutar preglednika. Chrome nije jedini preglednik koji podržava ovaj protokol. Tako onda jezgra Chrome preglednika pod nazivom Chromium isto podržava ovaj protokol. Unutar Chromium projekta sadržan je Blink engine (Wikipedia, 2018c) . Prema izvoru Blink je kreiran nad WebCore komponentom WebKit engine-a koji je originalno razvijen od strane KDE-a. Prema tome DevTools protokol je teoretski podržan na svim preglednicima koji koriste Blink kao na primjer Opera preglednik. U sklopu ovoga rada DevTools protokol će se koristiti preko instance Chromium preglednika kroz Puppeteer sučelje koje je zapravo omotač oko DevTools protokola i moguće ga je koristiti u JavaScript i sličnim jezicima, a dostupan je i u sklopu NPM-a (eng. Node Package Manager) i kompatibilan s ostalim NodeJS bibliotekama. Nešto više o Puppeteer-u i tehnologijama u kasnijem poglavlju.

4.2. Funkcionalnosti

Sljedeća poglavlja sadrže opis funkcionalnosti i njihovo korištenje preko DevTools protokola. Navedene funkcionalnosti se tiču elemenata koje će biti korištene u sklopu praktičnog dijela ovoga rada. Za ostale funkcionalnosti i domene pristupa može se konzultirati oficijelna dokumentacija projekta (Google, 2018b) .

4.2.1. HTML DOM

Ova domena izlaže operacije pisanja i brisanja u lokalni DOM web stranice. Za svaki element stranice (HTML tag) postoji zrcaljeni objekt s jedinstvenim identifikatorom. Preko

identifikatora je moguće dohvatiti sve dodatne informacije o elementu kao na primjer atributi, sadržaj ili njegove ugrađene elemente. Protokol ovdje ne sadrži detalje o elementima nego ih on samo prosljeđuje klijentu, a klijent (preglednik na primjer) je zadužen za to da svaki primljeni element zapiše i vodi zapis o njemu. Protokol nikada ne šalje dva ista elementa niti ponavlja slanje. Ugrađeni HTML dokumenti npr. U "iframe" elementu se isto obrađuju samo što je njihov direktni roditeljski element ugrađeni dokument, a ne globalni dokument. Svaka operacija nad elementima se onda radi na klijentu s metodama unutar protokola. Unutar DOM-a je moguće postavljati i oznake za traženje grešaka gdje će se izvršavanje ili pro paginacija na primjer događaja na stranici zaustaviti na postavljenoj oznaci tj. elementu. Uz DOM događaje pauzira se i izvršavanje JavaScript-a ako postoji metoda dodana na element, a više o JavaScript debug-u u sljedećem poglavlju.

4.2.2. JavaScript konzola (Runtime)

Domena daje pristup lokalnom JavaScript okruženju. Preko njega je moguće evaluirati objekte, izvršavati metode, dodavati nove i drugo. Kako stranice danas aktivno koriste JavaScript svi zahtjevi se izvršavaju zrcaljenjem originalnih objekata. Svaki objekt zadržava originalnu referencu u memoriji osim ako se objekt eksplicitno ne obriše JavaScript naredbama kao što su “delete” i druge dostupne u JavaScript programskog jeziku. Ova domena pruža najviše mogućnosti jer praktički dozvoljava da na lokalnoj instanci stranice koju smo dobili na klijentu (Internet pregledniku) izvršavamo vlastiti JavaScript kod i manipuliramo i upravljamo stranicom kako želimo, skupljamo podatke i slično. Kako je spomenuto u prošlom poglavlju moguće je postavljati oznake unutar koda kako bi se lokalni JavaScript kod ispitivao ili da se pronađe greška tijekom izvršavanja na stranici. Neki IDE-i (eng. Integrated Development Environments) koriste dodatke za preglednike kao što su Chrome kako bi iskoristili ovaj aspekt DevTools protokola i omogućili svojim korisnicima bolju podršku za testiranje koda izvršavanog u pregledniku.

4.2.3. Memorija preglednika i lokalni spremnik

Domene u pogledu memorije preglednika i lokalnog spremnika kao što su LocalStorage ili SessionStorage su u trenutnoj verziji još uvijek u eksperimentalnoj fazi. Ono što je trenutno podržano su osnovne operacije nad pred memorijom i spremnikom. Razlika je samo to što je njihova stabilnost upitna. Što se tiče sesije, kolačića i identifikacijskih podataka oni su tretirani kroz domenu preglednika koja omogućava upravljanje instancom preglednika. Svi spremljeni podatci su lako dohvatljivi i u pravilu se brišu nakon zatvaranja instance, ali je moguće postaviti

drugu lokaciju za lokaciju spremnika ili spremiti objekt na drugo mjesto pa potencijalno upotrijebiti u više navrata.

4.2.4. Mrežni promet

Stranice su poslužene preko HTTP/HTTPS protokola i većina zahtjeva koje rade je za resursima i datotekama na poslužitelju s kojega dolaze. U budućnosti će HTTP2 omogućiti otvaranje sigurnog kanala i spajanje resursa kako bi klijent to jest preglednik radio što manje zahtjeva. Nadziranje mrežne aktivnosti stranice nam omogućava znanje o tome da li su svi resursi preuzeti i da li postoje otvoreni zahtjevi i onda ovisno o tome izvršavati radnje iz drugih domena. Na primjer možemo pričekati da se HTML učita kako bi onda tek dohvatili neki element i njegov sadržaj i DOM-a. Kako su današnje stranice često SPA (eng. Single Page Application) tipa često koriste asinkrone servise za dohvat podataka pa onda nekada treba pričekati da stranica dobije odgovor i osvježi svoj sadržaj prije nego izvršimo nekakvo prikupljanje podataka ili pritisnemo neki gumb na stranici.

4.3. User Agents

Korisnički agenti (eng. User Agents) u kontekstu Internet preglednika služe kako bi pristupanoj web stranici ili aplikaciji dali podatke o pregledniku, operacijskom sustavu i nekada o korisniku. Ove informacije su važne stranici kako bi mogla prikazati odgovarajući sadržaj korisniku ovisno o npr. Internet pregledniku preko kojega pristupa, rezoluciji ekrana i slično. Korisnički agenti dolaze s klijentske strane i u većini slučajeva klijent (preglednik ili nešto drugo) je zadužen za slanje ispravnog agenta kako bi se ispravno predstavio poslužitelju. Praksa u informacijskoj sigurnosti se u ovom kontekstu svodi najčešće na frazu "Nikada ne vjeruj klijentu" (Hacker News, 2016) . Upravo preko protokola ili programskih sučelja je moguće vrlo lako lažirati to jest postaviti korisničkog agent-a na što god želimo. Takve se metode koriste u praksi za testiranje mobilnih uređaja gdje ne posjedujemo fizički uređaj, ali isto tako za maskiranje radnji na internetu. Ovo drugo dovodi do ilegalnih radnji, ali to nije direktna tema ovoga rada, a u kasnijem poglavlju će biti rečeno nešto više o posljedicama i moralnim pogledima ovakvog načina pristupa sadržaju. Puppeteer sučelje koje će biti korišteno u nastavku emitira poseban korisnički agent koji ga identificira, ali ga je isto tako vrlo lako moguće postaviti na nešto drugo, a čak i promijeniti tijekom izvođenja skripte to jest instance preglednika.

5. Automatizacija programskih proizvoda

Automatizacija kao pojam u izradi programskih proizvoda, održavanja sustava i testiranja ima znatnu ulogu. Potreba za automatizacijom se pojavila kao rješenje za repetitivne zadatke ili iste zadatke koje je potrebno odraditi u nekom određenom periodu ili specifičnog datuma. Cron (Wikipedia, 2018d) je na primjer alat za vremensko odrađivanje poslova unutar UNIX operativnih sustava. Često je korišten za automatizaciju raznih poslova koji se tiču poslova administratora sustava, ali i bilo čega drugog unutar sustava koji drže aplikacije ili web servera. Cron onda može u određenom vremenu ili vremenskom intervalu odraditi niz radnji koje se mogu definirati skriptom ili dati putanja na gotovu skriptu koju će sustav izvršiti u namješteno vrijeme. Cron i slični alati su terminalski ili programski pokretani iz komandne linije. Kako su preglednici programi kojima pristupamo preko grafičkog sučelja automatizacija akcija unutar takvog sučelja je više prilagođena ljudima nego programima. U tu svrhu su razvijeni razni programi ili okruženja koja omogućuju programiranje radnji koje računalo treba izvršiti unutar grafičkog sučelja. Programi se najčešće oslanjaju na prepoznavanje slika (eng. Image recognition) ili raščlanjivanje (eng. Parsing) npr. U slučaju internet preglednika HTML koda stranice i onda razumijevanjem strukture stranice znaju gdje se nalazi element koji treba pritisnuti u okviru odrađivanja akcije. Testiranje grafičkog sučelja tijekom razvoja programskog proizvoda je repetitivan zadatak koji je proizveo nekoliko programskih okvira baš za tu svrhu. Jedan od njih je i Puppeteer koji će biti korišten u kontekstu praktičnog dijela ovoga rada (samo u malo drugačije svrhe) . Ostali poznatiji su:

- Selenium – alat otvorenog koda namijenjen za automatizaciju testiranja web aplikacija
- Katalon Studio – alat za automatizaciju testiranja izgrađen na Selenium-u, podržava web, mobilne aplikacije i Internet servise
- Unified Functional Testing (UFT) - komercijalni alat za funkcijsko testiranje. Podržava više platformi i uz ostalo i desktop aplikacije
- Watir – alat otvorenog koda pisan u Ruby programskom jeziku, namijenjen za testiranje web aplikacija
- IBM Rational Functional Tester – alat za testiranje raznih aplikacija (web i desktop) koristi "Storyboard" koncept za planiranje akcija
- TestComplete – komercijalni alat namijenjen za testiranje web, mobilnih i desktop aplikacija, koristi prepoznavanje objekata u procesu automatizacije
- TestPlant eggPlant - još jedan alat, ali prilagođen krajnjem korisniku, više otvoren za ljude koji ne poznaju ili nisu upućeni u programiranje i razvoj programskih proizvoda

- Tricentis Tosca – alat za testiranje s dobrom podrškom za analizu podataka skupljenih tijekom testiranja
- Ranorex – alat za automatizaciju testiranja web aplikacija isto podržava kreiranje testova bez i jedne napisane linije koda
- Robot framework - alat otvorenog koda, odvaja se od drugih jer ga je moguće lako proširiti s drugim alatima ili bibliotekama za testiranje, tako na primjer može biti proširen sa Selenium-om kako bi testirao web aplikacije.

5.1. Primjene

Kako je spomenuto u uvodu ovog poglavlja postoji nekoliko primjena automatizacije u kontekstu Internet preglednika i razvoja programskih proizvoda. Osim testiranja postoji još nekoliko situacija gdje se automatizacija aktivno koristi i dalje će biti pokriven još jedan aspekt koji je bliže vezan uz tematiku i kontekst ovog rada. Prikupljanje ili generiranje podataka je često implementirano automatizacijom. Na primjer generiranje kompliciranih izvještaja, slanje izvještaja email-om ili prebacivanje lista podatka iz jednog formata u neki drugi. Automatiziranje se može postići na glavna dva načina:

- Programiranjem bot-a i zadavanje radnji koje će obaviti u kontekstu sučelja (operacijski sustav, preglednik ili nešto treće)
- Snimanjem radnji koje obavlja čovjek, a pomoću tehnologije prepoznavanja objekata radnje se slijedno spremaju i kasnije mogu biti ponovljene.

Obje metode imaju isti rezultat, ali proces automatizacije u krajnosti ovisi o programskom okviru koji koristimo. U slučaju ovog rada Puppeteer će biti programski okvir i on će biti korišten za programiranje botova koji će onda posjećivati stranice i kupiti određene podatke te ih vračati na klijenta (instancu Chromium preglednika na računalu), a ti podatci će onda biti dostupni kroz REST servis kojemu će se moći pristupiti preko internog web server-a.

5.2. Prednosti

Najveća prednost automatizacije je uvijek bila ušteda vremena. Automatizacijom skraćujemo vrijeme obavljanja kompleksnih, ali i jednostavnih radnji. Isto tako uštedimo puno vremena gdje na primjer ne moramo biti fizički prisutni za računalom kako bi u određeno vrijeme na primjer izradili i poslali izvještaj. Još jedna prednost je to što se programski okviri i načini na koje možemo automatizirati radnje na računalu stalno razvijaju i tek u zadnjih

nekoliko godina smo dobili stvarne primjere automatizacije koja je dostupna krajnjim ili običnim korisnicima. Dosta popularni su tako zvani osobni pomoćnici (Siri, Google Assistant, Bixby itd.) koji u sklopu operacijskog sustava pružaju korisniku da im direktno zada zadaće koje će obaviti, podsjetiti ga na neki događaj ili jednostavno u pozadini odraditi neku radnju kao što je postavljanje alarma ili slanje email-a. Kada govorimo o automatizaciji možemo jako brzo doći u područje AI-a (eng. Artificial Intelligence) . AI podrazumijeva neku vrstu samo zaključivanja i učenja na temelju okoline. Zato treba reći da iako su ova dva pojma slična i mogu pomagati u sličnom kontekstu treba reći da automatizacija u kontekstu ovog rada nema veze s AI-em i strojnim učenjem. Naravno AI bi se mogao iskoristiti za dodatno proširenje tijekom prikupljanja podataka što se zapravo direktno veže na neke mane koje su trenutno najveći problem kod ovakvog tipa automatizacije.

5.3. Mane

Automatizacija je vrlo korisna, ali sama definicija već prikazuje njenu glavnu manu. Naime automatizacija se oslanja na predefinirani niz radnji u konstantnom okruženju. Konstantno okruženje u kontekstu da se radnje uvijek izvode na isti način, a okruženje se jedino mijenja u smislu onoga što okolina sadrži i prikazuje, ali ne i kako prikazuje. Na primjer ako postoji tablični prikaz nekih podataka ovakav bot očekuje da će ta tablica imati istu ili vrlo sličnu strukturu. Ako odjednom podatci budu prikazani drugačije npr. preko grid-a velika je vjerojatnost da zadatak neće biti uspješno odrađen. Druga mana se veže na to da je kod promjene okoline potrebno ponovno programirati bot-a. Zapravo kod bilo kakve promjene niza koraka potrebno je promijeniti i doraditi bot-a. Postoji nekoliko metoda koje mogu ovo ublažiti. Te metode ovise o okolini u kojoj bot odrađuje radnje pa nema potrebe konkretno spominjati svaku. Neke od tih metoda u kontekstu Internet preglednika će biti opisane u kasnije dijelu rada kada će se objašnjavati logika iza bot-a izrađenog za prikupljanje podataka za krajnji REST servis.

6. Tehnologije

6.1. NodeJS

NodeJS je odvojeno JavaScript okruženje koje se temelji na V8 engine-u razvijenom za Google Chrome od strane Google-a. Razvio ga je Ryan Dahl 2009. godine. NodeJS je podržan na svim platformama (u početku samo na macOS i Linux) i do sada je predstavljan kao alat otvorenog koda. Upravo to što je od početka predstavljen kao alat otvorenog koda NodeJS je brzo skupio popularnost među programerima (Hámori, 2017) . Koristi se u razne svrhe, a nije ograničen samo na web aplikacije i one koje bi inače povezali ili izrađivali s JavaScript jezikom. NPM (eng. Node Package Manager) je upravitelj paketa za NodeJS i ujedno i CLI alat (eng. Command Line Interface). Preko njega je vrlo brzo moguće postaviti osnovni NodeJS projekt, uključiti potrebne vanjske biblioteke i krenuti s razvojem. Popularnost NPM-a dokazuje i to da je NPM postao najveći registar programskog koda i proizvoda u svijetu, a svakoga dana bilježi više miliona preuzimanja paketa iz repozitorija. Ono što odvaja NodeJS od običnog razvoja JavaScript koda je fokus na programiranje pogođeno događajima (eng. Event Driven Programming) . Ovakav način programiranja je dosta popularan i danas kod mobilnih i nativnih aplikacija gdje mnoge radnje trebaju čekati odgovore ili notifikacije operacijskog sustava da bi izvršile neki zadatak i gdje naravno nije sve sinkrono. (Wikipedia, 2018e) . NodeJS se najčešće koristi za razvoj serverskih aplikacija ili aplikacija koje u nekom pogledu komuniciraju preko interneta i mreže. NodeJS u svojoj jezgri daje pakete i biblioteke za upravljanje mrežnim protokolima, pristup datotečnom sustavu, kriptografiji i drugim. Najveća razlika u izradi web servera na ovakav način u usporedbi s na primjer razvojem servera u PHP-u je u više nitnost tj. paralelnom procesiranju preko događaja. Na primjer PHP funkcije i akcije moraju većinom čekati odrađivanje radnji prije da bi se one izvele tj. blokiraju ostale funkcije koje možda i ne ovise o njima. Za razliku NodeJS vrlo lako omogućuje izvršavanje funkcija i akcija ne ovisno o redoslijedu ili čak paralelno. Treba napomenuti da paralelni rad nije riješen preko više nitnosti kao na primjer u operacijskim sustavima ili jezicima kao što je Java. Paralelnost je riješena preko događaja gdje svaka funkcija može određenim odaslati događaj kako bi javila da je završila i na taj događaj može početi nova funkcija i tako dalje. Kako je Chrome-ovo V8 okruženje pisano za JavaScript vrlo je efikasno, a ujedno su svi web programeri sa znanjem JavaScripta mogli početi pisati NodeJS aplikacije. Na NPM repozitoriju postoje razni projekti od kojih nekih čak nisu povezani s web serverima ili web-om nego s nativnim aplikacijama i drugim primjenama. Na primjer postoje biblioteke kao što su

ReactNative i Electron koji nisu direktno dio NodeJS-a, ali podržavaju platformu preko NPM repozitorija što dodatno pojačava cijelu zajednicu koja koristi ovaj pristupačan i brz način razvoja.

6.2. Puppeteer

Puppeteer biblioteka je bila spomenuta u nekoliko ranijih poglavlja. To je biblioteka za NodeJS koja omogućuje rukovanje DevTools protokolom unutar okruženja. Puppeteer omogućava da skoro sve radnje koje možemo napraviti preko normalnog grafičkog sučelja preglednika možemo napraviti i preko programskog koda. Ovo otvara mogućnosti za već navedene automatizacije aspekata preglednika, Internet stranica itd. Neka od glavnih funkcionalnosti koje Google ističe (Google, 2018c) za Puppeteer su:

- Generiranje slika i PDF dokumenata stranica
- Prikupljanje podatak iz SPA aplikacija i generiranje dinamičkog sadržaja
- Automatizirano slanje obrazaca
- Testiranje korisničkog sučelja
- Unos podatka
- Kreiranje testnog okruženja nad instancom Chrome preglednika
- Snimanje mrežnog prometa i rada aplikacije u svrhu optimizacije

Puppeteer biblioteka je razvijana i održava ju Chrome DevTools tim. Ovo ujedno osigurava da svaka nova verzija radi i više-manje podržava sve funkcionalnosti trenutne verzije Chrome tj. Chromium preglednika. Uz svaku instalaciju Puppeteer biblioteke dolazi integrirana inačica Chromium preglednika koja je testirana i preporučuje ju se koristiti radi stabilnosti. Moguće je konfigurirati Puppeteer i s bilo kojom drugom instalacijom Chrome-a na računalu/okruženju, ali mogu postojati određene greške. Chrome DevTools tim je predan i daljnjem razvoju ove biblioteke što je dovoljna garancija da će projekt biti podržan u budućnosti. Neki od daljnjih ciljeva za razvoj su:

- Povećati efikasnost i kompaktnost biblioteke
- Pokazati najbolji način implementacije kako bi druge biblioteke mogle koristiti Puppeteer kao jezgru
- Povećati popularnost “headless” preglednika
- Unaprijediti automatizaciju preglednika i testiranje

Konkretno korištenje Puppeteera će biti prikazano u sljedećim poglavljima. Sljedeće točke će se bazirati na osnovno objašnjenje kako Puppeteer funkcionira.

- Preko biblioteke kreiramo instancu preglednika koja može biti s grafičkim sučeljem (vizualno možemo vidjeti što se događa) ili bez što je brža i efikasnija inačica i preporuča se za produkcijske ili memorijski teže i intenzivnije radnje
- Kreiranje instance se provodi asinkrono preko događaja i sustav nakon podizanja instance vraća odgovor te je moguće dalje raditi s njom kao i inače s preglednikom
- Puppeteer kreira zasebni Chrome korisnički profil i uz njega šalje posebnog korisničkog agenta kojega je moguće zamijeniti s bilo kojim drugim zapisom
- Kod zatvaranja stranice potrebno je pravilno ugasiti instancu preglednika ako više neće biti korištena ili sačuvati referencu unutar memorije kako bi se mogla kasnije ponovno iskoristiti

6.3. Express

Pod drugim nazivom Express.js je programski okvir za web aplikacije u NodeJS okruženju. Razvijen je i pušten u javnost kao alat otvorenog koda. Danas je on zapravo standard za bilo kakve web aplikacije pisane u NodeJS okruženju. Autor programskog okvira je TJ Holowaychuk (Wikipedia, 2018f) . Autor je rekao kako je glavni cilj bio razviti kompaktni, ali proširiv okvir koji bi omogućio da se sve dodatne funkcionalnosti lako dodaju bez diranja izvornog koda ili jezgre okvira. Proširenja se dodaju kao tako zvani “Middleware-i” gdje se koristi Express baza, ali se bilo koji aspekt može programski prepisati i dodati funkcionalnost. Ako je pravilno napisano, proširenje može dalje biti prošireno novim i tako dalje. Nekoliko zanimljivih projekata koristi ovu biblioteku u ne običajne svrhe kao što su testiranje ili razvoj demo proizvoda. Osnovno Express može biti podignut nad HTTP bibliotekom unutar NodeJS okruženja gdje se onda kao web aplikacija poslužuje na zadanoj adresi i opcionalno nekom drugom port-u osim 80 (standardni HTTP port) . Osnova aplikacije je usmjerivač (eng. Router) unutar kojega definiramo putanje koje mogu biti pristupane kroz web aplikaciju. Ovakav način je pristupačan za kreiranje REST servisa koji ne zahtijevaju veliku obradu odgovora nego vračaju JSON formatiran tekst. Express pruža i početne 404 i stranice s pogreškama koje je moguće zamijeniti vlastitima. Router isto radi na principu događaja i podržava asinkrone metode u JavaScript jeziku. Važno je u slučaju korištenja asinkronih metoda postaviti limit za odgovor kako bi aplikacija tj. server ipak nakon nekog vremena vratio odgovor klijentu.

7. Praktični dio – prikupljanje i prikaz podataka

7.1. Twitter platforma

Twitter je društvena mreža na kojoj korisnici objavljuju kratke poruke zvane “Tweet” . Originalno su Tweet-ovi bili ograničeni na 140 znakova, ali od 11. studenog 2017. godine broj znakova je produljen na 280. Twitter su kreirali Jack Dorsey, Noah Glass, Biz Stone i Evan Williams u ožujku 2006 godine, a službeno je počeo s radom u srpnju iste godine. Twitter trenutno broji više od 100 milijuna članova (Wikipedia, 2018g). Twitter ima službeni API koji sadrži više metoda za dohvat i objavljivanje korisničkog sadržaja. U sklopu ovoga rada bit će implementiran sličan servis, ali korištenjem automatizacije preglednika preko koje će se ostvariti komunikacija s Twitter platformom. Programsko sučelje Twitter API razvijeno u sklopu ovoga seminarskog rada sadrži sljedeće metode:

- Login - Prijavi se u Twitter web aplikaciju kao određeni korisnik
- Follow - prati odabranog korisnika
- Unfollow - prestani pratiti odabranog korisnika
- Like tweet - favoriziraj određeni tweet odabranog korisnika
- Like recent tweet - favoriziraj sve nedavno objavljene tweet-ove odabranog korisnika
- Like last tweet - favoriziraj zadnji tweet odabranog korisnika
- Retweet - Objavi određeni tweet odabranog korisnika
- Retweet last - Objavi zadnji tweet odabranog korisnika
- Followers - Dohvati sve korisnike koji prate odabranog korisnika
- Interests - Dohvati sve korisnike koje odabrani korisnik prati
- Follow network - Prati sve korisnike koji prate odabranog korisnika
- Follow interests - Prati sve korisnike koji odabrani korisnik prati
- Logout - Odjavi se iz Twitter web aplikacije kao određeni korisnik

Za realizaciju ovih metoda koristi se programsko sučelja za Puppeteer koje kontrolira Twitter web aplikaciju kao da to radi stvarni korisnik.

7.2. Implementacija programskog okvira

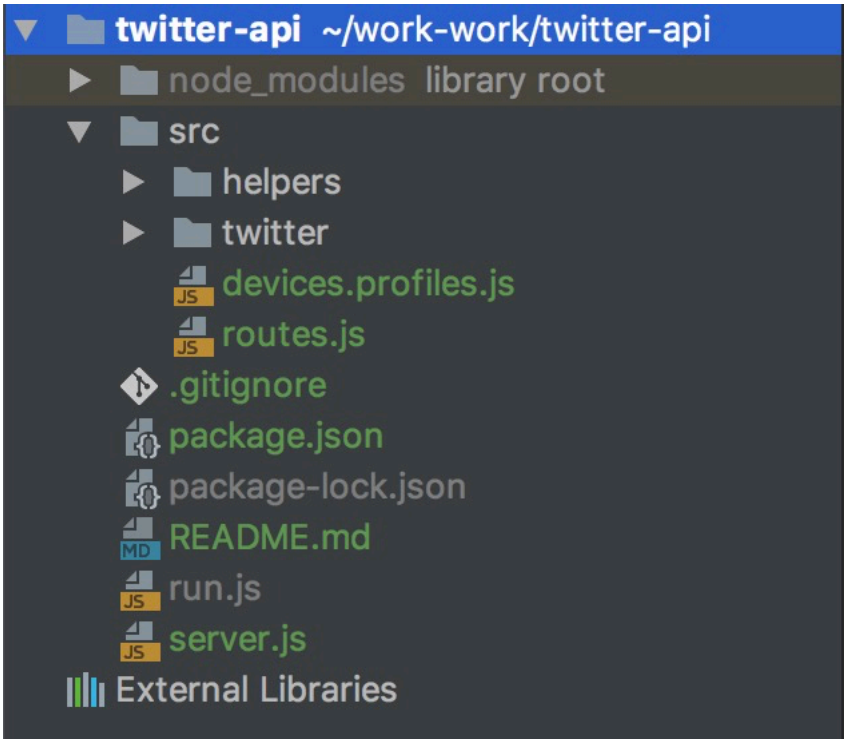
Implementacija programskog okvira je rađena u JavaScript programskom jeziku. Korištena je ES2017 sintaksa. Sučelje je podijeljeno u tri zasebna dijela od kojih svaki predstavlja jedan dio funkcionalnosti Twittera:

- Ovjera autentičnost – prijava i odjava korisnika i korisnička sesija.
- Korisnik – dohvaćanje podatak o korisniku, pratiteljima, objava sadržaja kao određeni korisnik.
- Poruke – slanje i pregled poruka kao određeni korisnik

Svaki dio je napisan kao zasebna klasa kojoj je moguće pristupiti kroz glavnu klasu koja ujedno i sadrži metode za prijavu i odjavu i sadrži podatke o sesiji korisnika. Za svakoga korisnika je potrebno kreirati novi objekt Twitter klase i proslijediti mu korisničke podatke.

7.2.1. Arhitektura i struktura projekta

Struktura projekta je relativno jednostavna i bazira se na klasičnoj strukturi NodeJS projekta. Slika 1. prikazuje datotečnu strukturu projekta.



Slika 1 Datoteke projekta

Datoteka "package.json" sadrži sve meta i opisne podatke za projekt i služi kao identifikator za projekt ako bi se postavljao na NPM repozitorij. Datoteka još sadrži skripte koje se mogu pokrenuti u kontekstu projekta i vanjske biblioteke projekta. Kod ispod prikazuje datoteku "package.json" projekta:

```
{
 "name": "twitter-api",
 "version": "1.0.0",
 "description": "",
 "dependencies": {
  "express": "^4.16.3",
  "puppeteer": "^1.4.0",
  "request": "^2.87.0"
 },
 "scripts": {
  "test": "echo \"Error: no test specified\" && exit 1"
 },
 "repository": {
  "type": "git",
  "url": "git+https://github.com/capJavert/twitter-api.git"
 },
 "keywords": [],
 "author": "Ante Barić (capJavert)",
 "license": "MIT",
 "bugs": {
  "url": "https://github.com/capJavert/twitter-api/issues"
 },
 "homepage": "https://github.com/capJavert/twitter-api#readme"
}
```

Mapa "helpers" sadrži pomoćne klase korištene u projektu. Jedna sadrži pomoćne metode za uvijete, a druga za filtriranje i procesiranje teksta. Datoteka device.profiles.js sadrži profile koji će instanca Chromium preglednika koristiti tijekom izvršavanja. Datoteka u ovom slučaju sadrži samo jedan profil, ali može sadržavati i više pa se ovisno o situaciji oni mogu izmjenjivati.

```
module.exports = {
  desktop: {
    'name': 'Desktop',
    'userAgent': 'Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/41.0.2228.0 Safari/537.36',
    'viewport': {
      'width': 1280,
      'height': 800,
    }
  }
};
```

Profil ujedno predstavlja korisničkog agenta i informacije koje će dobiti stranica kada bude otvarana unutar preglednika. Datoteke routes.js i server.js će biti objašnjene u poglavlju Implementacija REST servisa. Mapa "twitter" sadrži datoteke klasa koje su odgovorne za automatizaciju preglednika. One se nadovezuju jedna na drugu i prosljeđuju podatke preko data atributa tijekom kreiranja pojedinih objekata i kod promjene stanja.

7.2.2. Otvaranje instance preglednika

Svaka klasa je zasebni modul kojemu se pristupa iz roditeljske instance Twitter klase. Svaki modul koristi Puppeteer biblioteku za automatiziranje preglednika. To rade tako da izvode metode nad instancom stranice. Kako bi objasnili funkcioniranje biblioteke prvo ćemo objasniti kako se dolazi do instance stranice. Prvo je potrebno kreirati instancu preglednika i u slučaju da želimo ovdje definiramo parametre kao što su način rada, korisnički agent i drugi. Sljedeći kod prikazuje kreiranje instance preglednika:

```
(async () => {
  const userData = {
    username: 'USERNAME',
    password: 'PASSWORD'
  }

  await puppeteer.launch({headless: headless, timeout: 0}).then(async browser => {
    let page = await browser.newPage()
    await page.emulate(DevicesProfiles.desktop)

    const twitter = await new Twitter(page)
    let zeusFoi = twitter.user()
    await twitter.login(userData.username, userData.password)
  })
})()
```

Kako je već spomenuto sve metode su asinkrone pa se koristi ključna riječ "await" kako bi se egzekucija programa zaustavila dok u ovom slučaju operacijski sustav ne podigne instancu preglednika. Metodom "launch" objekta Puppeteer kreiramo instancu preglednika. Unutar konteksta metode prosljeđujemo metodu koja će biti izvršena nakon što metoda "launch" bude izvršena tj. dok se događaj kreiranja preglednika izvrši. Ovakav način ugnježđivanja metoda će biti često korišten u daljnjim prikazima koda pa ga je važno ovdje razumjeti. Kao rezultat "launch" metoda će proslijediti instancu preglednika kroz browser varijablu i ta instanca će živjeti sve dok se nalazimo unutar konteksta. Objekt tipa Browser sadrži metodu "newPage()" koja otvara novu stranicu (u slučaju Chrome-a nova kartica), a kao rezultat vraća objekt tipa "Page". Opet koristimo ključnu riječ "await" kako bi program pričekao da nova kartica bude

otvorena u instanci preglednika. Svakoj stranici možemo dati drugačijeg korisničkog agenta. U ovom slučaju dajemo jedinog agenta iza atributa desktop. Sada kada se nalazimo u kontekstu preglednika i imamo instancu stranice tu instancu možemo proslijediti Twitter klasi koja kreira novi objekt tipa Twitter i otvara metode dostupne preko našeg API sučelja. Sljedeće dvije linije prikazuju korištenje klase "User" i prijavu u Twitter pomoću korisničkog imena i lozinke, ali na taj dio ćemo se vratiti nakon što detaljnije objasnimo funkcioniranje Twitter API-a.

7.2.3. Twitter modul

Kreirani Twitter objekt je jedinstven unutar konteksta preglednika. Slično kako kada ste jednom ulogirani u svoj profil na nekoj stranici kada god otvorite novu karticu u pregledniku s tom istom stranicom neće biti potrebe za ponovnom prijavom jer već postoji sesija koja se čuva u pregledniku. Twitter objekt kroz konstruktor prima instancu stranice kako bi se njome moglo automatizirano upravljati programskim putem.

```
class Twitter {
  /**
     * Init page and data from puppeteer Page
   *@param page
   */
  constructor(page) {
    this.page = page
    this.data = {
      'baseurl':'https://twitter.com',
      'session':null
    }

    this.page.on('dialog',async dialog => {
      await dialog.accept()
    })
  }
```

Unutar konstruktora se sprema referenca na objekt stranice i kreira se data atribut koji je tipa "object" i u koji će se spremati svi podatci tijekom izvršavanja programa. Spremamo odmah adresu početne stranice Twitter aplikacije kako bi mogli pravilno kreirati adrese koje će preglednik posjećivati. Kako stranice često prikazuje tako zvane skočne prozore zadnja linija koda postavlja metodu koja će se izvršiti ako stranica koja pošalje bilo kakav skočni prozor. Možemo vidjeti da opet koristimo ugnježđivanje gdje dobivamo varijablu "dialog" i u kontekstu skočnog prozora postavljamo zadanu radnju preko metode "accept()". To znači da će preglednik automatski potvrditi svaki skočni prozor kako bi mogao nastaviti dalje. Ovdje je

moguće postaviti bilo koju kompliciraniju logiku, ali za potrebe ovoga projekta ovo će biti dovoljno.

7.2.3.1. Prijava korisnika

Metoda "login(username, password)" prima parametre korisničkog imena i lozinke.

```
async login(username, password) {
  this.data.username = username
```

Korisničko ime se sprema u data atribut kako bi API kasnije znao o kojem se korisniku radi prema imenu. Lozinka se sprema jer nakon prijave više nije potrebno znati lozinku korisnika te se sve rješava preko sesije i kolačića unutar preglednika kao i kod normalnog otvaranja web stranice u pregledniku. Nakon spremanja korisničkog imena slijedi niz metoda kojima automatiziramo preglednik da odradi prijavu kroz Twitter aplikaciju. Kako je ovo prvi puta da prikazujemo ove metode one će biti detaljnije opisane. Ostale se baziraju na istom konceptu ili se ponavljaju na drugim mjestima.

```
try {
  return true
} catch(e) {
    console.log(e)

  return false
}
```

Za početak svaki niz naredbi koje upravljaju preglednikom su ugniježđene u "try-catch" blok za hvatanje greška. Ovo je napravljeno u svrhu da ako dođe do greške tijekom automatizacije ta greška ne sruši cijelu instancu preglednika nego da se sljedeće naredbe ili novi pokušaj mogu ponoviti u istoj instanci kao da se ništa nije dogodilo.

```
await this.page.goto(this.data.baseurl+"/login", {waitUntil: 'networkidle2'})

if(this.data.session === null){
  await this.page.waitForSelector('button.submit')

  await this.page.type(".js-username-field", username)
  await this.page.type('.js-password-field', password)
  await this.page.click('button.submit')

    console.log("Submit clicked")
```

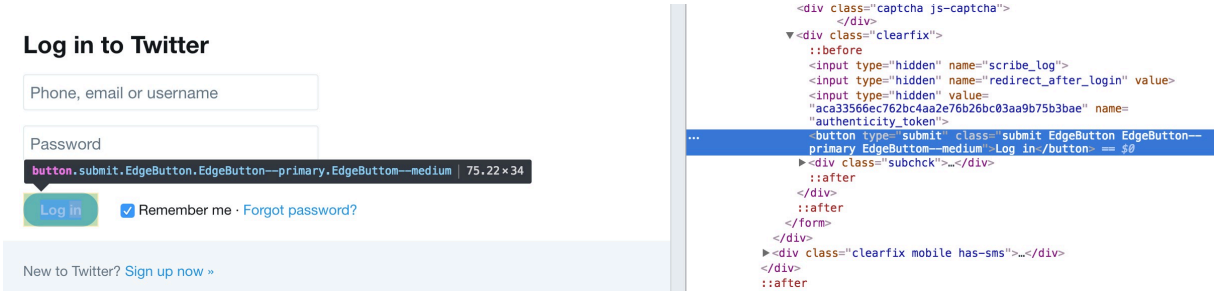
```
  await this.page.waitForSelector('.dashboard-left')

  this.data.session = "SESSION_KEY"

  console.log("Logged in")
} else {
  await this.page.evaluate(session => {
    for (let key in session) {
      if (session.hasOwnProperty(key)) {
                sessionStorage.setItem(key, JSON.stringify(session[key]))
      }
    }
  }, this.data.session)

    console.log("Logged from session")
}
```

Još jednom treba istaknuti da metode pozivane nad instancom stranice imaju ključnu riječ "await" ispred sebe jer je svaka asinkrona. Metoda "goto()" omogućava kretanje na novu stranicu u trenutno otvorenoj aktivnoj kartici preglednika (ekvivalent upisivanja adrese u adresnu traku preglednika). Nakon što otvorimo stranicu za prijavu provjeravamo da li već postoji spremljena sesija tj. ako je metoda “login()“ već bila pozvana onda se ne trebamo prijavljivati ponovno. Ako nismo prijavljeni koristimo posebnu metodu "waitForSelector(selector) " koja pauzira izvršavanje tj. vraća stanje “true” tek kada se HTML element s određenim selektorom pojavi na stranici. Ovo je najbolja metoda za detekciju da li se neki dio stranice učitao jer se sadržaj u današnjim modernim aplikacijama često dinamički generira ili čega na dodatne HTTP zahtjeve. Bilo kakvo fiksno čekanje određenog broja sekundi može rezultirati u grešci da element ne postoji nakon određenog vremena. Isto tako ako se element ne pojavi u određenom vremenu ova metoda vraća dediciranu grešku "timeOutException" kroz koju možemo lako shvatiti o čemu se radi. Nakon što znamo da je gumb za prijavu prikazan na stranici možemo biti sigurni i da je forma za prijavu prikazana. Kako znamo sve to? Programiranje automatiziranih skripti preko Puppeteer-a uključuje određenu igru probe i promašaja. Naime moramo otvoriti stranicu, pogledati njen DOM i analizirati koje elemente želimo i kako im pristupiti što možemo vidjeti na slici 2.



Slika 2 - Forma za prijavu i njen DOM prikaz

Identičan prikaz kao na slici 2. postoji i u instanci preglednika koju smo pokrenuli, a kako Chrome razumije DevTools protokol, a Puppeteer mu šalje komande preko baš toga protokola on zna što treba odraditi. Na sličan način nađemo i selektore za polja u koja upisujemo korisničko ime i lozinku te na sljedeći način iskoristimo metode "type()" i "click()" kako bi upisali podatke i potvrdili formu.

```
await this.page.type(".js-username-field",username)
await this.page.type('.js-password-field',password)
await this.page.click('button.submit')

console.log("Submit clicked")

await this.page.waitForSelector('.dashboard-left')

this.data.session = "SESSION_KEY"

console.log("Logged in")
```

Metode "type" i "click" i ostale koje se izvršavaju u kontekstu DOM-a stranice primaju CSS (eng. Cascading Style Sheet) selektore kojima možemo jednoznačno identificirati svaki element na stranici. Metoda "type" upisuje tekst u HTML element (u ovom slučaju je to <input> polje) . Metoda "click" odrađuje događaj pritiska na gumb kao da je to napravio korisnik i sva logika u web aplikaciji se izvršava kao da joj pristupa normalni korisnik. Na kraju opet koristimo metodu "waitForSelector()" kako bi pauzirali automatizaciju dok se stranica ne učita i ne pojavi se glavna korisnička ploča koju korisnik vidi kada se prijavi u Twitter web aplikaciju. Pozivi console.log su dobri za brzo i kasnije otkrivanje grešaka jer kreirana instanca preglednika pokrenuta u NodeJS okruženju ostavlja iza sebe JavaScript log koji možemo spremiti ili zapisivati u neku datoteku kroz izlazni tok podataka u terminalskom sučelju. Nakon što se ploča prijavi to znači da je korisnik prijavljen i metoda login() vraća "true". U ovom trenutku

preglednik čeka s otvorenom stanicom koja se zadnja pojavila. To znači da sada da teoretski možemo samo nastaviti od tamo i automatizirati neku drugu akciju.

7.2.3.2. Odjava korisnika

Kako je gumb za odjavu korisnika većinom dostupan u navigaciji Twitter aplikacije ovo je dobar primjer da se prikaže dobra praksa kada želimo za redom izvršiti više radnji nad jednom instancom Twitter API-a. Nakon prijave smo stali na početnoj korisničkoj stranici. Možemo odmah probati kliknuti gumb za odjavu i to je to odjavili smo se, ali to nije uvijek najbolje rješenje. Što ako smo u međuvremenu odradili neke druge radnje i na stranici smo s postavkama profila koja nema gumb za odjavu na istom mjestu? Zato je bolje uvijek tijekom novog niza akcija ponovno "goto()" metodom otići na stranicu koja nam treba jer ako već jesmo na toj stranici nećemo napraviti novi HTTP zahtjev, a ako nismo izbjegavamo potencijalne pogreške. Opcija "waitUntil" u metodi "goto()" se koristi još za odgodu događaja ako na mreži postoje aktivni HTTP zahtjevi. Na primjer ako stranica osim inicijalnog učitavanja radi dodatne zahtjeve. Metoda prima niz znakova ili polje niza znakova predefinirane vrijednosti:

- "load" - navigacija na stranicu se smatra završenom kada je inicijalno učitavanje završeno
- "domcontentloaded" - navigacija na stranicu se smatra završenom kada je DOM stablo stranice učitano
- "Networkidle0" - navigacija na stranicu se smatra završenom kada u zadnjih 500 milisekundi nije bilo novih zahtjeva ili konekcija
- "networkidle2" - navigacija na stranicu se smatra završenom kada u zadnjih 500 milisekundi nije bilo više od dva zahtjeva ili konekcije

  Metoda za odjavu sadrži sljedeći niz naredbi i prikazana je u kodu ispod:

```
await this.page.goto(this.data.baseurl+"/logout",{waitUntil: 'networkidle2'})

await this.page.waitForSelector('button.js-submit')
await this.page.click("button.js-submit")
this.data.session = null

console.log("Logged out")

return true
```

Odlazimo na stranicu za odjavu, čekamo da se prikaže gumb za odjavu, kliknemo ga i ispraznimo sesiju. Nakon toga smo se odjavili. Sesiju brišemo zbog provjere u login() metodi da li sesija postoji, a slična postoji i na početku ove metode:

```
async logout(){
  if (this.data.session === null){
        console.log("Not logged in")

    return false
  }
```

Ako sesija nije postavljena odmah vraćamo "false" jer se ne možemo odjaviti ako nismo prijavljeni. Možemo još primijetiti da je "logout()" metoda isto kao i login() označena sa "async "ključnom riječi. To je iz razloga da bi mogli usporedno kada pozivamo metodu login() iz roditeljske funkcije kao u kodu na početku poglavlja koristiti ključnu riječ "await" za pauziranje izvršavanja dok cijeli proces prijave ili odjave ne bude gotov. To je u svrhu što su ove i ostale metode unutar Twitter klase izolirane i predstavljaju jedan automatizirani niz radnji nad Twitter web aplikacijom.

7.2.4. Modul korisnika

Nakon što je korisnik prijavljen i postoji sesija preko metode "user()" moguće je dohvatiti objekt korisnika koji otvara novi set metoda. Metoda "user()" unutar klase Twitter vraća objekt tipa "User". Svakim pozivom metode se generira nova instanca pa je preporučljivo u implementaciji sačuvati referencu na objekt korisnika nakon kreiranja, a ne stalno tražiti novi objekt prije poziva neke od metoda. Klasa "User" sadržava veći broj metoda od kojih neće sve biti detaljno objašnjene. Ostale je moguće s dokumentacijom pregledati na službenom repozitoriju (Barić, 2018a) . Korisnici su na Twitteru jednoznačno identificirani preko korisničkog imena. Korisnička imena nisu osjetljiva na velika i mala slova tako da je korisničko ime "capJavert" i "capjavert" isto i vodi na istu poveznicu. Metode klase User rukuju korisničkim imenima korisnika i odrađuju akcije vezane uz korisničke interakcije unutar Twitter web aplikacije.

7.2.4.1. Praćenje korisnika

Bazna interakcija između korisnika na Twitter platformi je praćenje drugih korisnika i njihovih kanal objava. Korisnici mogu odlučiti da je njihov račun privatan ili javan. To znači da zahtjevi za praćenje ako je račun javan automatski budu prihvaćeni i korisnik odmah postaje

pratitelj. Privatan račun znači da korisnik koji posjeduje račun mora odobriti zahtjev za praćenjem. Što se tiče logike praćenja ona je neovisna o računu i zahtjev se šalje na isti način. Unutar "User" klase postoje dvije metode:

- "follow(username)" – određeni korisnik prati korisnika s navedenim korisničkim imenom
- "unfollow(username)" – određeni korisnik prestaje pratiti korisnika s navedenim korisničkim imenom

Ove dvije metode imaju sličnu logiku što se tiče niza naredbi za automatizaciju samo su obrnute jedna od druge.

```
async follow(username) {
  try {
    await this.page.goto(this.data.baseurl+'/'+username, {waitUntil:
                  'networkidle2'})

    await this.page.waitForSelector('.ProfileNav-list .user-actions .js-follow-
                  btn')

    if (await this.page.$('.ProfileNav-list .user-actions.not-following')
                  !== null) {
      await this.page.waitForSelector('.ProfileNav-list .user-actions .js-
                  follow-btn')
      await this.page.click('.ProfileNav-list .user-actions .js-follow-btn')
    } else {
            console.log('Already Following')
    }

    return true
  } catch(e) {
        console.log(e)

    return false
  }
}
```

Follow metoda na početku odlazi na stranicu profila korisnika koja je formata "twitter.com/:username". Nakon toga s metodom "waitForSelector()" čeka dok se profil korisnika ne učita do kraja provjerom DOM elemenata na stranici. Sljedeća klauzula ispituje stanje gumba za praćenje korisnika. Gumb u stanju praćenja korisnika ne sadržava klasu “not-following” . Ako gumb ne sadrži tu klasu izvršava se pritisak na gumb. Nakon izvršavanja akcije metoda vraća uspjeh. Ako korisnik već prati korisnika u log se zapisuje notifikacija o tome da korisnik već prati korisnika i metoda isto vraća uspjeh. Metoda "unfollow()" se razlikuje samo u "if-else" klauzuli:

```
if (await this.page.$('.ProfileNav-list .user-actions.following') !== null) {
  await this.page.waitForSelector('.ProfileNav-list .user-actions .js-follow-btn')
  await this.page.click('.ProfileNav-list .user-actions .js-follow-btn')
} else {
```

```
    console.log('Not following')
}
```

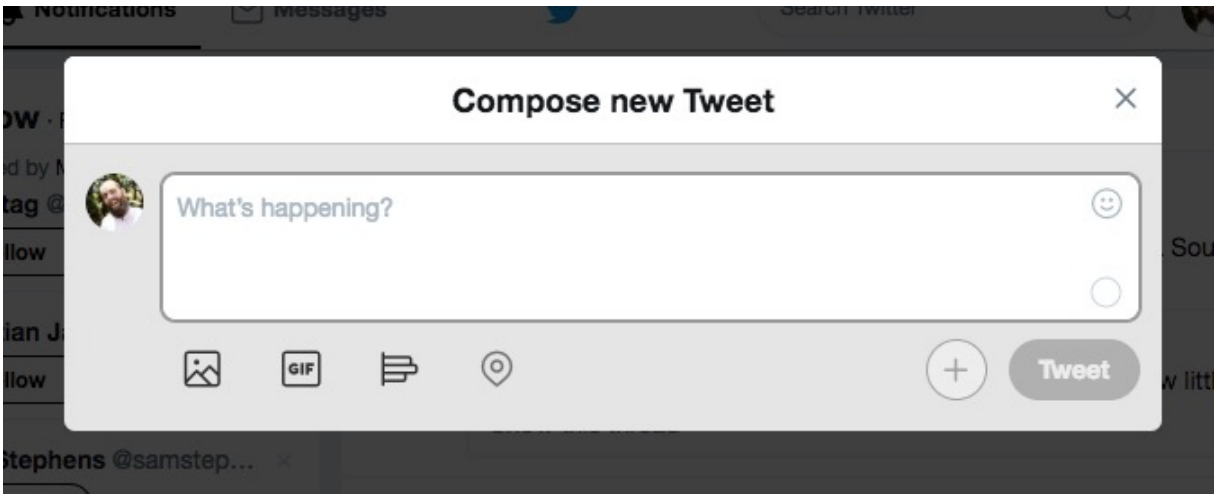
Dok je gumb u stanju praćenja sadrži klasu “following” i ovisno o tome se odradi pritisak na gumb i korisnik se više ne prati. Ove dvije metode mogu biti automatizirano pozvane naizmjenično više puta. Kako počinju na istoj stranici (ako se radi o istom korisniku) onda se u instanci preglednika zapravo samo automatski naizmjenično pritišće gumb i status se mijenja. Ipak nije poželjno puno puta u kratkom vremenu izvršiti ove metode jer je moguće pogoditi limit zahtjeva na Twitter aplikaciju što je i inače mogu s klasičnim API-ima i zahtjevima.

7.2.4.2. Objava statusa

Objava statusa (tako zvanih Tweet-ova) je glavni način na koji korisnici mogu kreirati sadržaj na Twitter-u. Statusi uz tekst mogu imati dodatne sadržaje:

- Slike
- Video
- GIF animacije
- Ankete
- Lokaciju

Linkovi se unutar Tweet-a broje kao tekstualni znakovi i većinom se smanjuju na 22 znaka preko servisa za smanjivanje URL-ova.



Slika 3 - Sučelje za kreiranje objava na Twitter-u

Slika 3. prikazuje sučelje za kreiranje Tweet-ova koje se automatizira u sklopu API-a. Za potrebe ovoga rada omogućeno je samo slanje tekstualnih Tweet-ova s linkovima i

emotikonima. Vrlo lako bi se funkcionalnost mogla proširiti i na dodavanje ostalog sadržaja na sličan način kako će biti prikazano u sljedećim odjeljcima koda. DOM stablo ove forme sa Slike 3. za unos Tweet-ova je prikazano na Slici 4.

```
▼<div class="modal" id="Tweetstorm-dialog-dialog" role="dialog" aria-labelledby="Tweetstorm-
dialog-header" style="top: 5%; left: 197px;">
    <div class="js-first-tabstop" tabindex="0"></div>
...  ▼<div class="modal-content TweetstormDialog-content" role="document"> == $0
      ▶<div class="modal-header">…</div>
      ▶<div class="modal-body">…</div>
      ▶<div class="modal-footer">…</div>
      </div>
    ▶<button type="button" class="modal-btn modal-close js-close" aria-controls="Tweetstorm-
    dialog-dialog">…</button>
      <div class="js-last-tabstop" tabindex="0"></div>
    </div>
</div>
```

Slika 4 - DOM stablo forme za unos statusa

Metoda "tweet()" klase "User" prima jedan parametar, a to je tekst Tweet-a. Tekst je tipa "String" i može sadržavati tekst, linkove i kodove emotikona. Gumb za objavu statusa se nalazi na svakoj stranici u gornjem desnom kutu Twitter web aplikacije. Zato na početku metode postoji provjera da li se nalazimo na stranici koja sadržava gumb. Ako se ne nalazimo idemo na početnu stranicu aplikacije koja sigurno sadrži gumb. Metodom "waitForSelector()" nakon navigacije provjeravamo da li se gumb i prikazao.

```
async tweet(text) {
  try {
    if (!Helpers.urlsEqual(this.page.url(), this.data.baseurl)) {
      await this.page.goto(this.data.baseurl, {waitUntil: 'networkidle2'})
      await this.page.waitForSelector('.js-tweet-btn')
    }

    await this.page.waitForSelector('#tweet-box-home-timeline')
    await this.page.click('#tweet-box-home-timeline')
    await this.page.waitForSelector('#tweet-box-home-timeline.is-showPlaceholder')
    await this.page.type('#tweet-box-home-timeline', text)
    await this.page.waitForSelector('.js-tweet-btn')
    await this.page.click('.js-tweet-btn')

    return true
  } catch(e) {
        console.log(e)

    return false
  }
}
```

Sljedeće naredbe redom automatiziraju popunjavanje teksta Tweet-a u polje za unos te pritisak na gumb za objavu. U slučaju da preglednik automatizirano pritisne gumb za objavu, ali zahtjev za objavu Tweet-a još nije završio može se pojaviti skočni prozor koji pita korisnika da li je

siguran da želi otići na drugu stranicu. Tu se okida metoda prikvačena na događaj skočnog prozora. Na njoj je definiran automatski klik na gumb potvrde tako da automatizacija preglednik ne zapne na tom prozoru. Metoda ovdje može vračati podatke o novom Tweet-u, ali za prezentacijske svrhe ovdje je samo stavljeno da vraća potvrdan odgovor tj. "true". Osim objave Tweet-a koje korisnik sam izradi moguće je ponovno objaviti Tweet drugog korisnika. Kada objavimo takav Tweet on ima poveznicu na originalni Tweet, ali se vodi i sadržan je na kanalu određenog korisnika. Metode "retweet()" i "retweetLastTweet()" unutar "User" klase služe za objavu tuđih Tweet-ova.

```
async retweet(tweetId, username) {
  try {
    await this.page.goto(this.data.baseurl+'/'+username+'/status/'+tweetId, {waitUntil:
'networkidle2'})

    await this.page.waitForSelector('.PermalinkOverlay-modal div.stream-item-footer
.ProfileTweet-actionButton.js-actionRetweet')

    try {
      await (await this.page.$('.PermalinkOverlay-modal div.stream-item-footer
.ProfileTweet-actionButton.js-actionRetweet')).click()
      await (await this.page.$('.RetweetDialog-retweetActionLabel')).click()
    } catch (e) {
              console.log('Already retweeted')
    }

    return true
  } catch(e) {
        console.log(e)

    return false
  }
}
```

Metoda "retweet()" prima parametre identifikator Tweet-a koji dobijemo nakon što je objavljen i korisničko ime korisnika koji ga je objavio. Korisničko ime i ID sačinjavaju URL svakog Tweet-a u Twitter web aplikaciji. Automatizirano preglednik odlazi na URL Tweet-a i čeka metodom "waitForSelector()" da se stranica učita do kraja. Unutar ove metode prvi puta možemo vidjeti korištenje $() metode koja kao parametar prima HTML/CSS selektor elementa. Metoda pretražuje DOM stablo stranice i vrača rezultat kao objekt tipa "ElementHandle" . Dobivene elemente možemo obrađivati i pozivati metode slično kao i nad objektom tipa "Page" . Na primjer metode "click()","type()" i isto tako "$()" metoda može biti izvršena nad elementom, samo se ovaj puta gleda samo sadržaj i pod elementi unutar njega. Postoji i metoda $$() koja podrazumijeva da vrača više rezultata. Na primjer kada pretražujemo elemente prema CSS klasi koja može biti na više elemenata pa dobivamo jedan ili više elemenata. Odgovor ove

metode je uvijek lista elemenata koja može sadržavati i samo jedan ili niti jedan element. Metoda sadrži sličnu provjeru kao kod metoda "follow()" i "unfollow()" gdje gumb za radnju ima različitu klasu određeno o stanju.

Slika 5 - Retweet gumb

Metoda "retweetLastTweet()" ne prima Tweet ID kao parametar nego samo ime korisnika. Zbog toga su koraci u automatizaciji malo drugačiji. Metoda odlazi na profil korisnika i nakon učitavanja onda na njegovom kanalu traži zadnji Tweet koji je korisnik objavio (obično onaj na vrhu kanala) . Na tom Tweet-u klikne gumb za retweet akciju.

```
async retweetLastTweet(username) {
  try {
    await this.page.goto(this.data.baseurl+'/'+username, {waitUntil: 'networkidle2'})

    await this.page.waitForSelector('.ProfileTweet-action--retweet')

    try {
      await (await this.page.$('.ProfileTweet-actionButton.js-actionRetweet')).click()
      await (await this.page.$('.RetweetDialog-retweetActionLabel')).click()
    } catch (e) {
             console.log('Already retweeted')
    }

    return await this.page.$eval('div[data-tweet-id]', element => {
      return element.getAttribute('data-tweet-id')
    })
  } catch(e) {
        console.log(e)

    return false
  }
}
```

Retweet metoda sadrži istu provjeru da li je korisnik već objavio ovaj Tweet. U prijašnjem kodu možemo vidjeti i korištenje metode "$eval()" . Ova metoda je slična metodi "$()" samo što

umjesto "ElementHandle" tipa objekata može vratiti bilo koje podatke koje mi kreiramo ili obradimo kroz ugniježđenu metodu. Metoda "$eval()" vraća DOM element, a "$$eval()" uvijek vraća listu DOM elemenata koji se prosljeđuju funkciji. Unutar ove funkcije metodom "getAttribute()" nad DOM elementom dohvaćamo “data-tweet-id” atribut i vračamo ga kao rezultat "retweetLastTweet()" metode. Na sličan način se kupi ID novog Tweet-a nakon kreiranja metodom tweet() .

7.2.4.3. Sviđanje sadržaja

Svaki sadržaj (svoj ili tuđeg korisnik) korisnik može pratiti ili označiti da mu se sviđa. Sve što korisnik označi da mu se sviđa se grupira kronološki u "likes" odjeljak na korisničkom profilu u Twitter web aplikaciji. API podržava metode za sviđanje sadržaja te metodu za dohvat sadržaja koji je korisnik označio da mu se sviđa. Prva metoda je metoda "like(tweetId, username) " preko koje korisnik može "lajkati" određeni Tweet drugog korisnika. Naravno možemo proslijediti i vlastito korisničko ime pa "lajkati" vlastiti Tweet.

```
async like(tweetId,username) {
  try {
    await this.page.goto(this.data.baseurl+'/'+username+'/status/'+tweetId,{waitUntil:
'networkidle2'})

    await this.page.waitForSelector('.PermalinkOverlay-modal div.stream-item-footer
.ProfileTweet-actionButton.js-actionFavorite')

    try {
      await (await this.page.$('.PermalinkOverlay-modal div.stream-item-footer
.ProfileTweet-actionButton.js-actionFavorite')).click()
    } catch (e) {
              console.log('Already liked')
    }

    return true
  } catch(e) {
        console.log(e)

    return false
  }
}
```

Unutar Twitter aplikacije svaki Tweet ima svoj jedinstveni URL formata "/username/status/tweetId" . Unutar metode se prelazi na stranicu prema proslijeđenim parametrima. S metodom "waitForSelector()" se čeka na učitavanje HTML elementa koji sadrži Tweet. Sljedeći "try-catch" blok je zadužen za “like-anje” Tweet-a. Pokušava se “lajkati” Tweet i ako klik akcija ne uspije to znači da je gumb već pritisnut. Na slične načine rade i metode "likeLastTweet()" i "likeRecentTweets()" .

7.2.4.4. Interesi korisnika

Na Twitteru korisnici iskazuju svoje interese praćenjem drugih korisnika i korisničkih profila koji predstavljaju stranice, poznate osobe, servise i slično. U sklopu ovog API-a postoji nekoliko metoda za praćenje korisnika i praćenje interesa drugih korisnika. Metode za praćenje korisnika su objašnjene u prijašnjem poglavlju, ali se na njima baziraju metode za praćenje interesa drugih korisnika te prikupljanje podataka o interesima drugih korisnika. Prva od metoda je "followers()" . Metoda se koristi za dohvaćanje liste pratitelja nekoga korisnika ili svojih pratitelja.

```
async followers(username = ''){
  try {
    if (ConditionsUtil.isNotNullNorEmpty(username)) {
      await this.page.goto(this.data.baseurl+'/'+username+'/followers', {waitUntil:
'networkidle2'})
    }else {
      await this.page.goto(this.data.baseurl+'/followers', {waitUntil: 'networkidle2'})
    }

    await this.page.waitForSelector('.AppContent-main')

    return await this.page.$$eval('.AppContent-main .username.u-dir .u-linkComplex-
target', elements => {
      let followers = []

      for (let element of elements) {
                followers.push(element.innerText)
      }
              followers.shift()

      return followers
    })
  }catch(e) {
        console.log(e)

    return Helpers.wrapError(e)
  }
}
```

Prvi dio metode je sličan kao i do sada. Odlazi se na stranicu i provjerava se da li je učitana do kraja. Ugniježđena metoda koja se poziva unutar "$$eval" poziva je zadužena za prikupljanje podataka iz DOM-a stranice. Konkretno uzimaju se elementi s određenom klasom te se njihov tekstualni sadržaj sprema u polje. Svaki element u ovom slučaju sadrži korisničko ime korisnika koji je ispisan na stranici pratitelja na kojoj se nalazimo. Popunjeno polje se vraća kao povratna vrijednost "$$eval()" funkcije i taj rezultat vraća i API metoda. Slično radi i "interests()" metoda koja vraća popis korisnika koje određeni korisnik prati tj. popis njegovih

interesa na Twitter-u. U kontrastu s ove dvije, metode "followNetwork()" i "followInterests()" se koriste za praćenje korisnikovih pratitelja i korisnika koje prati određeni korisnik. Metode imaju sličnu logiku, a za primjer ćemo pogledati prvu.

```
try {
  if (ConditionsUtil.isNotNullNorEmpty(username)) {
    await this.page.goto(this.data.baseurl+'/'+username+'/followers', {waitUntil:
'networkidle2'})
  } else {
    await this.page.goto(this.data.baseurl+'/followers', {waitUntil: 'networkidle2'})
  }

  let response = {}
     response.username = username === "? 'me' : username
     response.status = "Network followed"

  await this.page.waitForSelector('.AppContent-main')

  for (let element of await this.page.$$('.AppContent-main .js-follow-btn')) {
    await element.click()
  }

     response.users = await this.page.$$eval('div[data-screen-name]', elements => {
    let users = []

    for (let element of elements) {
              users.push(element.getAttribute('data-screen-name'))
    }

    return users
  })

  return response
} catch(e) {
     console.log(e)

  return Helpers.wrapError(e)
}
```

Ova metoda prikazuje i dedicirano vraćanje odgovora u obliku objekta. Objekt odgovora se tipično sastoji od atributa povezanih sa zahtjevom i dodatno vraća status što je tekstualna poruka o stanju odgovora. Metoda koristi istu logiku za povlačenje podataka iz DOM-a stranice kao i metoda followers() . Razlika je u tome što se umjesto teksta HTML elementa dohvaća atribut koji u ovom slučaju sadrži ime korisnika. Podatci se vraćaju kao polje i pridodaju objektu odgovora. U slučaju uspjeha metoda vraća taj objekt, a u slučaju ne uspjeha vraća “error” objekt. Pomoću sljedeće metode uzima podatke o JavaScript pogrešci i prepisuje ih u objekt koji se vraća kao odgovor u slučaju pogreške.

```
static wrapError(error) {
  return {
    name: error.name,
    message: error.message
  }
}
```

Metoda koristi atribute name i message JavaScript objekta. Ime predstavlja tip pogreške, a poruka predstavlja opis pogreške. Ideja iza ove metode je standardizirati pogreške i na strani API-a.

7.2.5. Modul poruka

Modul poruka sadrži metoda za upravljanje direktnim porukama u sustavu Twitter web aplikacije. Metode su sljedeće:

- List – Dohvati sve razgovore određenog korisnika
- Messages – Dohvati sve poruke unutar odabranog razgovora
- Create – Kreiraj novi razgovor i pošalji prvu poruku odabranom korisniku
- Reply – Odgovori na poruku tj. pošalji novu poruku u odabrani razgovor
- Delete – Obriši odabrani razgovor

Sve metode zahtijevaju prijavu. Metode su pisane na isti način kao i ostale metode unutar programskog sučelja. Za razumijeva metode je dovoljno pogledati opisane primjere ranijih metoda. Programski kod metoda je moguće pregledati u repozitoriju projekta.

7.3. Implementacija REST servisa

Internet servis (API) koji je implementira u sklopu ovog rada služi kao praktični prikaz da se programski API na razini automatizacije preglednika može koristiti u svrhu prikupljanja i pružanja podataka. Servis se bazira na REST arhitekturi i poslužuje sadržaj u JSON formatu.

7.3.1. Arhitektura

REST servis je implementiran kao Internet aplikacija u Express programskom okviru. Kao bazu koristi aplikacijski “router” koji je zadužen za posluživanje sadržaja Internet aplikacije korisnicima kroz razne URL-ove. Router osim što sadrži definirane URL-ove za aplikaciju sadrži i podatke o sesiji. Express aplikacija se definira na sljedeći način:

```
const app = express();

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

require("./src/routes.js")(app);

const server = app.listen(3000, function () {
    console.log("Listening on port %s...", server.address().port);
});
```

Aplikacija prima razne parametre kroz konfiguracijski objekt. Na primjer u kodu iznad se prosljeđuje bodyParser varijabla koja dodaje funkcionalnost procesiranja povratnih aplikacija kao JSON. Pozivom metode listen() aplikacija se pokreće na određenom HTTP port-u i moguće je proslijediti funkcija koja će se izvršiti nakon pokretanja aplikacije. U ovom slučaju se samo ispisuje poruka i port aplikacije u konzolu nakon pokretanja.

```
const apiService = new ApiService();
const sessions = [];

/**
 * Run web app for Twitter API
 *
 * @param app
 */
const appRouter = function (app) {
  // URL mappings
}
```

“Router” se definira kao funkcija koja kao parametar prima kreiranu Express aplikaciju. Unutar funkcije idu pozivi metoda koje definiraju URL-ove aplikacije. Aplikacija ima niz funkcija od kojih svaka predstavlja ove četiri HTTP metode koje su spomenute u opisu REST arhitekture:

- GET
- POST
- PUT
- DELETE

Osim ovih sadrži i funkcije za ostale HTTP metode. “Router” radi na principu ulančavanja metoda i događaja. Na primjer odgovor koji bi metoda trebala vratiti može biti procesirana nekoliko puta i svaki puta metoda može proslijediti sljedeću funkciju. Svaka funkcija prima parametre "request", "response" i "next". Kod svakoga HTTP zahtjeva se popunjava "request" parametar, a "response" se stvara kada "router" mapira URL na metodu i pozivom metode

"send()" nad "response" parametrom možemo vratiti odgovor na HTTP zahtjev. Primjer koda koji izvršava taj postupak je sljedeći:

```
app.get('/api/register',function (req,res) {
    res.wrap({
    scope:['twitter-api'],
    key: apiService.createKey()
  })
})
```

Razlika je što umjesto "send()" metode ovdje se koristi metoda "wrap()" kojom je proširen objekt odgovora u sklopu ove aplikacije. Metoda ugnježđuje svaki odgovor u data atribut roditeljskog objekta.

```
app.response.wrap = function (data) {
  return this.send({data: data})
}
```

Pristup metodama servisa je ograničen i nije im moguće pristupiti bez autorizacije. Autorizacija se odvije preko API ključa koji je moguće registrirati preko metode za registraciju ključeva na servisu. Autorizacija je provedena kroz oAuth2 standard i ključ koji služi kao "token" se prosljeđuje kroz “Authorization” HTTP zaglavlje. Metoda zadužena za provjeru autorizacije koristi instancu klase "ApiService" . Ova klasa služi za upravljanje ključevima unutar sesije server-a. Kod inicijacije router-a kreira se instanca "ApiService" klase koja sadrži sve ključeve unutar za vrijeme rada servera. Klasa sadrži sljedeće metode:

- "createKey()" – kreira novi jedinstveni UUID identifikator koji će predstavljati API ključ i dodaje ga kao validnog u polje ključeva
- "deleteKey(key)" – briše određeni API ključ iz polja ključeva
- "isKeyValid(key)" – provjerava da li je određeni ključ ispravan tj. da li je sadržan u polju ključeva

Metoda koja koristi "ApiService" instancu i koristi se za provjeru identiteta korisnika izgleda ovako:

```
const isAuth = (request,response) => {
  const auth = request.header("authorization");

  if (!ConditionsUtil.isNullOrEmpty(auth)) {
    const key = auth.replace('Bearer ',")
```

```
    return apiService.isKeyValid(key)
  }else {
    return false
  }
}
```

Metoda provjerava "isKeyValid()" metodom da li je ključ validan i vraća "Boolean" vrijednost ovisno o rezultatu provjere. Metoda se koristi unutar direktive za autorizaciju koja se dodaje u inicijalizaciji svake metode servisa koji zahtjeva autorizaciju preko ključa. Metoda vraća HTTP status 401 što je standardni odgovor za korisnika u slučaju manjka prava za pristup nekom Internet resursu.

```
const restricted = (request, response) => {
  if (!isAuth(request, response)) {
    return response.status(401).wrap({
      status: 401,
      name: 'Unauthorized',
      message: 'Authorization Required'
    })
  }
}
```

Ako autorizacija nije uspješna direktiva na sebe preuzima slanje odgovora korisniku umjesto konkretne metode. Iz toga razloga "restricted()" metoda prima "request" i "response" parametre. Ovo je jedan konkretni primjer korištenja ulančavanja u sklopu Express aplikacije. Kako Express aplikacija nema trenutno stanje, barem ne u svom izvornom obliku bilo je potrebno na neki način omogućiti identifikaciju korisnika i dodijeliti svakomu njegovu sesiju. Upravo iz toga razlika svaki ključ identificira jednu sesiju koja predstavlja jedan kontekst preglednika koji odrađuje automatizaciju. Ovo možda nije idealno ako imamo veliki broj korisnika, ali za potrebe dokazivanja koncepta dobro balansira funkcionalnost i iskorištenost resursa koji su na raspolaganju. Za kreiranje instanci preglednika unutar Internet servisa se koristi klasa "BrowserService". Instanca ove klase se isto kreira u sklopu sesije servera. Klasa je zadužena za upravljanje instancama Chromium preglednika koji odrađuju automatizaciju. Puppeteer preko DevTools protokola komunicira s preglednikom na određenom WebSocket adresi. Kako bi se više različitih skripti moglo povezati na istu instancu bilo je potrebno negdje sačuvati listu tih adresa. Za to se brine "BrowserService" klasa. Klasa radi jednostavnosti i ne ovisnosti o datotečnom sustavu sprema listu u JSON formatu na GIST servis. "BrowserService" sadrži sljedeće metode:

- "saveSessionKey(sessionKey)" – sprema novu WebSocket adresu na GIST servis

- "loadSessionKey(index)" – uzima određenu po redu WebSocket adresu. Zadano uzima zadnje dodanu adresu. Adresa se lokalno sprema i dostupna je bez potrebe ponovnog pristupa GIST servisu kroz "sessionKey" atribut
- "clearAllKeys()" – briše sve spremljene adrese s GIST servisa

7.3.2. Metode

REST servis sadrži sve metode kao i programski Twitter API uz iznimke za metode koje se tiču autorizacije, dohvaćanja ključeva i provjere statusa. Metode su sljedeće:

- "/keys/register" - metoda za registraciju novog ključa za pristup servisu
- "/" - root metoda ispisuje korisnikov trenutno status, to sadrži prava pristupa, status ključa itd.
- "/keys/remove" - metoda briše ključ i uništava sesiju na serverskoj strani (gasi instancu preglednika za kontekst koji je pridružen ključu

Preporučljivo je ako bi se ovakvo rješenje koristilo u produkcijskom ili drugom ozbiljnijem kruženju redovito koristiti metodu za brisanje ključeva. To može biti u smislu da se korisnika upozori da bi bilo dobro da obriše ključeve nakon što mu više ne trebaju. Druga opcija je postavljanje zadatka preko Cron alata koji bi periodički brisao ključeve i sesije koje nisu korištene određeni vremenski rok. Metoda za brisanje sesije koristi i Puppeteer-ovu metodu za zatvaranje konteksta koji predstavlja sesiju iz njegove perspektive. Na ovaj način se osigurava da su ključevi i sesije sinkronizirani i da se ne može dogoditi da sesija ostane bez dediciranog ključa i tako bude izgubljena za dohvaćanje dok se cijela instanca servera ponovno ne pokrene. Obrnuti slučaj je moguć gdje postoji ključ koji nema sesiju. Iz metode za registraciju ključeva vidimo da se unutar te metode ne kreira sesija i instanca preglednika. Instanca preglednika se kreira tek kod prve uporabe ključa. Razlog ovakve implementacije je što bi na prvi način bilo moguće intenzivnim pozivanjem metode za registraciju kreirati veliki broj instanca preglednika u malom vremenu te premašiti memorijske resurse dostupne serveru. Na drugi način sesija se generira tek kod prvog poziva metode koja zahtjeva autorizaciju, a čim imamo autorizirani poziv možemo u slučaju problema blokirati pojedini ključ i izbjeći ugrožavanje rada servera. Ovo znači da će prvi poziv autorizirane metode biti malo sporiji zbog vremena potrebnog za otvaranje instance preglednika i kreiranja sesije. Metoda za dohvaćanje sesije je definirana unutar router-a aplikacije i izgleda ovako:

```
const loadSession = async (authorization, browser) => {
    const key = authorization.replace('Bearer ', '')
```

```
    if (ConditionsUtil.isNullOrEmpty(sessions[key])) {
        const session = {}
        session.context = await browser.createIncognitoBrowserContext()
        session.page = await session.context.newPage()
        session.page.emulate(DevicesProfiles.desktop)
        session.twitter = {
            core: await new Twitter(session.page),
        }
        session.twitter.user = session.twitter.core.user()
        session.twitter.directMessaging = session.twitter.core.directMessaging()

        sessions[key] = session
    }

    return sessions[key]
}
```

Metoda prvo pokuša dohvatiti sesiju prema ključu koji joj se proslijedi kao parametar. Ako sesija za taj ključ ne postoji metoda kreira novi kontekst, stranicu i sprema sve podatke u sesiju koja se pridružuje ključu. Nakon jedne ili druge situacije metoda vrača objekt sesije te se zahtjev dalje normalno izvršava ovisno o logici unutar metode router-a koja ju je pozvala. Sada kada je objašnjena logika stvaranja ključeva i sesija možemo pogledati metodu zaduženo za brisanje istih.

```
app.delete('/keys/remove', async function (req, res) {
    if(!req.body.key) {
        return res.wrap({'isError': true, 'status': 'error', 'message': 'missing a
parameter: key'})
    } else if(!apiService.isKeyValid(req.body.key)) {
        return res.wrap({'isError': true, 'status': 'error', 'message': 'Provided
key is not valid'})
    } else {
        apiService.deleteKey(req.body.key)

        const session = sessions[req.body.key]

        if (!ConditionsUtil.isNullOrEmpty(session)) {
            await session.context.close()
        }

        return res.wrap({'isError': true, 'status': 'ok', 'message': 'Provided key
is deleted and session is closed'})
    }
})
```

Metoda kroz parametar imena "key" u tijelu zahtjeva prima ključ koji korisnik želi obrisati. Ključ je obavezan parametar i ako nije sadržan u zahtjevu metoda vraća pogrešku. Nakon što je ključ učitan provjerava se da li je ključ važeći. Ako nije važeći metoda vrača pogrešku. Ako je ključ važeći metoda prvo briše ključ. Nakon toga dohvaća sesiju prema ključu slično kako to

radi "loadSession()" metoda. Kako je moguće da ključ nema sesiju prvo se provjerava taj slučaj i ako ona ne postoji metoda završava rad i vrača odgovor. Ako sesija postoji poziva se metoda "close()" nad "context" atributom objekta sesije. Ova metoda je asinkrona i nakon što vrati odgovor proces i memorija zauzeta instancom preglednika je oslobođena. Zatvaranje preglednika je puno brže od otvaranja tako da metoda unatoč tome što je asinkrona ne povećava vrijeme odgovora servisa na zahtjev. Ostale metode REST servisa direktno pozivaju i prosljeđuju odgovore iz Twitter API-a koji dolaze iz preglednika nakon prikupljanja. Metode imaju istu strukturu pa će za primjer biti prikazana jedna, a ostale se mogu pogledati na programskom repozitoriju projekta (Barić, 2018a) .

```
app.get('/messages/:threadid/list', async function (request, response) {
    restricted(request, response)
    const session = await loadSession(
        request.header("authorization"),
        browser
    )

    if(!request.params.threadid) {
        return response.wrap({'isError': true, 'status': 'error', 'message':
'missing a parameter: thread id'})
    } else {
        return response.wrap(await
session.twitter.directMessaging.messages(request.params.threadid))
    }
})
```

Možemo vidjeti da se prije izvršavanja svake Twitter metode poziva "restricted()" direktiva koja provjerava identitet autorizaciju korisnika. Nakon što je korisnik potvrđen učitava se njegova sesija koja sadrži instancu Twitter objekta koji se dalje koristi za automatizirano prikupljanje podataka. Ne primaju sve metode parametre, ali one koje primaju uvijek imaju provjeru da li su parametri prisutni i ako nisu odmah vračaju pogrešku. Ovo je iz razloga kako bi se osiguralo da samo ispravni podatci dođu do Twitter API metoda te da se većina grešaka i filtriranje korisničkog unosa popravi i odradi na razini web aplikacije servisa. Ako su svi podatci uredu poziva se odgovarajuća metoda API-a. Sve metode su asinkrone zbog poziva Twitter API metoda.

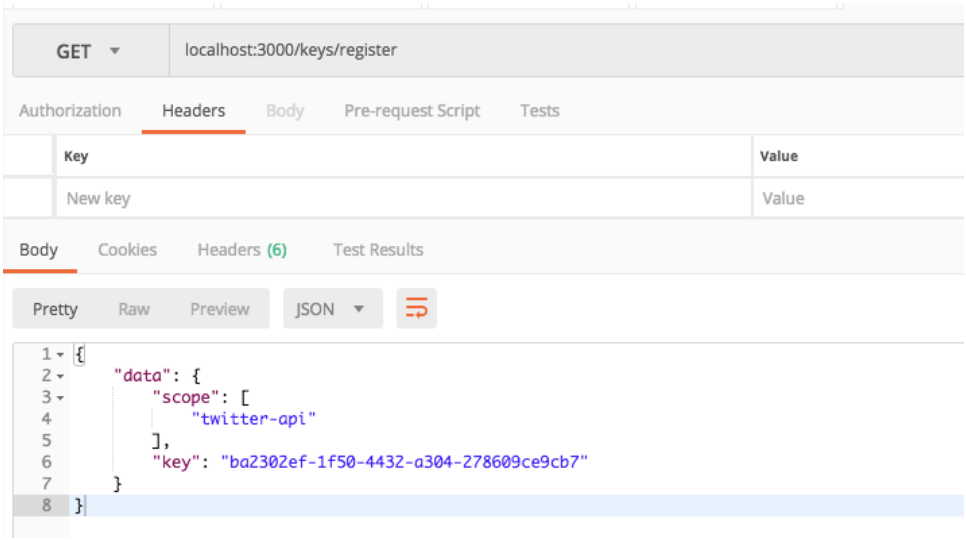
7.4. Demo implementacija

U ovom poglavlju će biti prikazan rad i konzumacija sadržaja API-a. Za konzumaciju API-a i prikaze JSON odgovora će se koristiti aplikacija Postman, a neke slike će prikazivati ispis sadržaja tablično kroz jednostavnu Angular aplikaciju koja konzumira servise. Detaljne

upute za instalaciju i pokretanje REST servisa lokalno se nalaze u README.md datoteci u repozitoriju projekta.

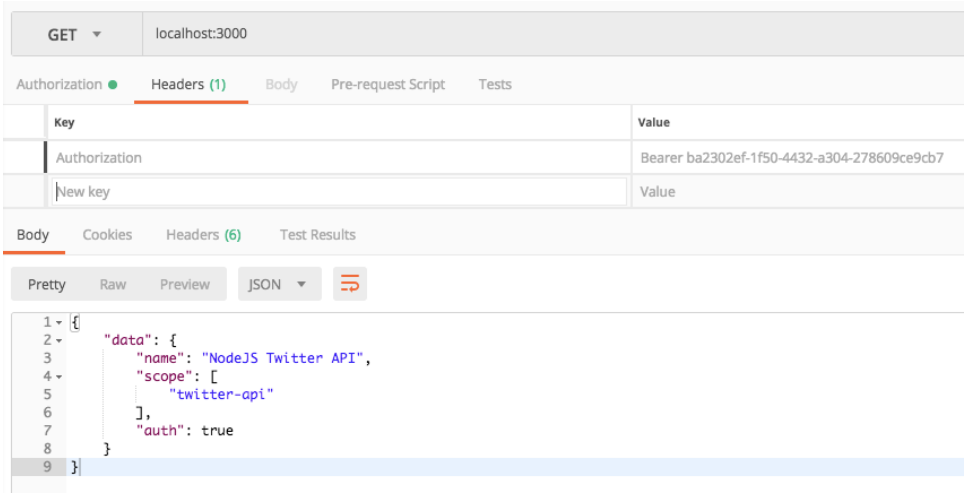
7.4.1. Konzumacija REST servisa

Nakon pokretanja servisa u konzoli će se ispisati port na kojemu sluša server. Adresa će ako je pokrenu lokalno biti na adresi 127.0.0.1 (localhost) . Serveru se može pristupiti iz preglednika, ali će s Postman aplikacijom koja ima sučelje da kreiranje zahtjeva biti puno lakše poslati i prikazati podatke. Prvo što treba napraviti je registrirati se za novi ključ.



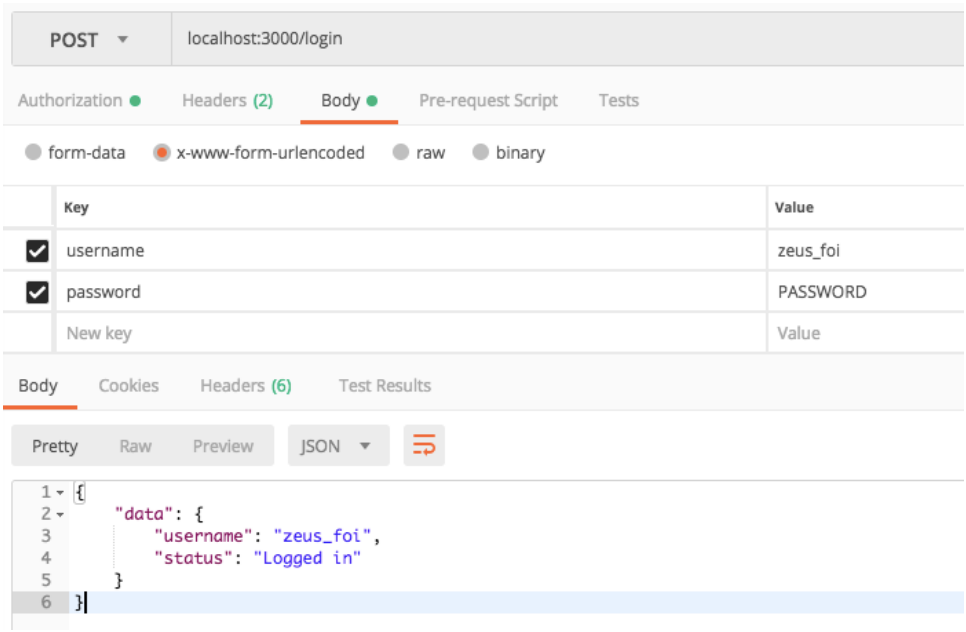
Slika 6 - Odgovor metode za registraciju ključa

Osim ključa atribut "scope" govori za koje metode ključ vrijedi. U ovom slučaju za Twitter API kojega jedino REST servis i podržava, ali u slučaju drugih postoji opcija za jednostavnu nadogradnju. Možemo vidjeti da je odgovor zapakiran u objekt, a podatci odgovora se nalaze unutar "data" atributa. Nakon što dodamo ključ u zahtjeve možemo provjeriti njegov status.



Slika 7 - Odgovor metode za provjeru statusa ključa

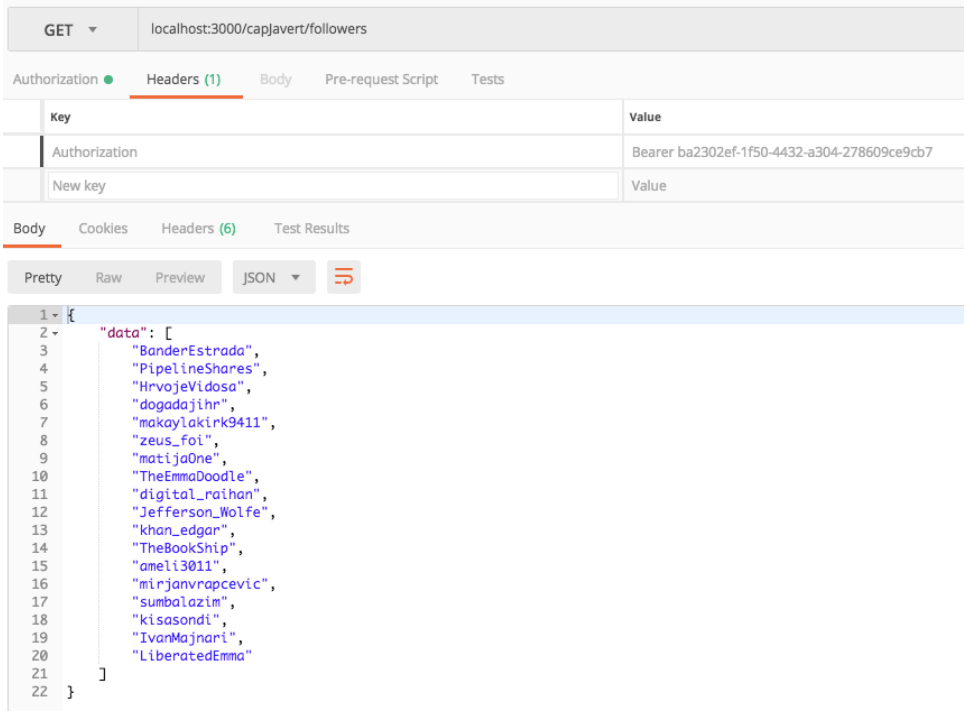
Možemo vidjeti da smo pravilno autorizirani i opet preko atributa "scope" prava pristupa. Sljedeća slika prikazuje poziv metode za prijavu na Twitter. Metoda prima parametre korisničko ime i lozinka koje šaljemo kroz tijelo zahtjeva u formatu "x-www-form-urlencoded" .



Slika 8 - Odgovor metode za prijavu na Twitter

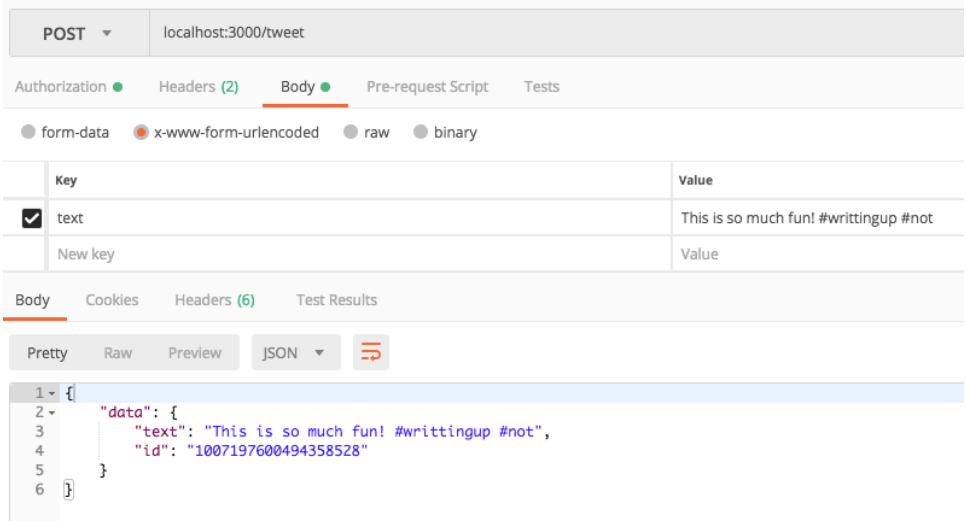
Kod prvog zahtjeva možemo vidjeti da je za odgovor trajao 5400 milisekundi što je dosta duže od prosječnog odgovora. To se odnosi na vrijeme podizanja instance preglednika za automatizaciju i kreiranja sesije. Zahtjev na sljedećoj slici koji dohvaća pratitelje korisnika je trajao 1063 milisekundi pa možemo vidjeti razliku. Brzina odgovora bi se još mogla smanjiti

"Cache" metodama ili postavljanjem REST servisa na računalu s bržom vezom na Internet ili boljim performansama.



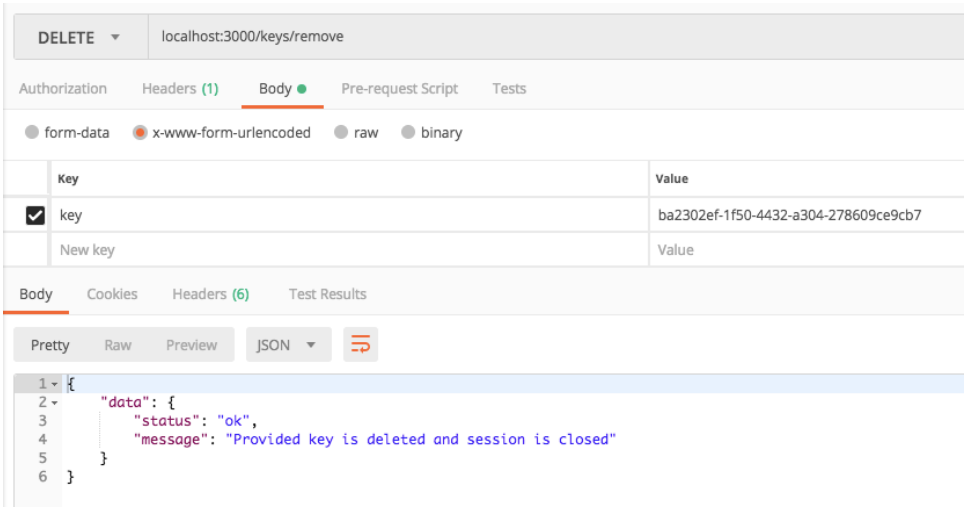
Slika 9 - Odgovor metode za ispis pratitelja korisnika

Metoda za objavu novog Tweet-a je najkompliciranija jer osim što mora upisati podatke, treba čekati na promjene na stranici (novi Tweet se treba prikazati) i onda nakon toga pokupiti podatke tj. Identifikator novog Tweet-a i vratiti ga kao odgovor.



Slika 10 - Odgovor metode za objavu novog Tweet-a

Metoda isto obavlja u 3066 milisekundi što nije loše s obzirom na sve navedene radnje. Ova metoda bi se teže optimizirala nekom "Cache" metodom, ali moguće bi ju bilo možda optimizirati automatizaciju detaljnijom analizom Twitter web aplikacije. Metoda za odjavu briše sesiju unutar konteksta preglednika i omogućava da se korisnik ponovno prijavi kao drugi korisnik ili kasnije kao isti. Nova prijava sada traje nekoliko sekundi kraće jer je preglednik već otvoren i sve što preglednik mora odraditi je automatizirani proces prijave u Twitter web aplikaciju. Naravno zatvaranje sesije unutar preglednika ne znači zatvaranje sesije ključa. Ključ i dalje vrijedi i moguće ga je koristiti sve dok ga ne obrišemo preko metode za brisanje ključeva.



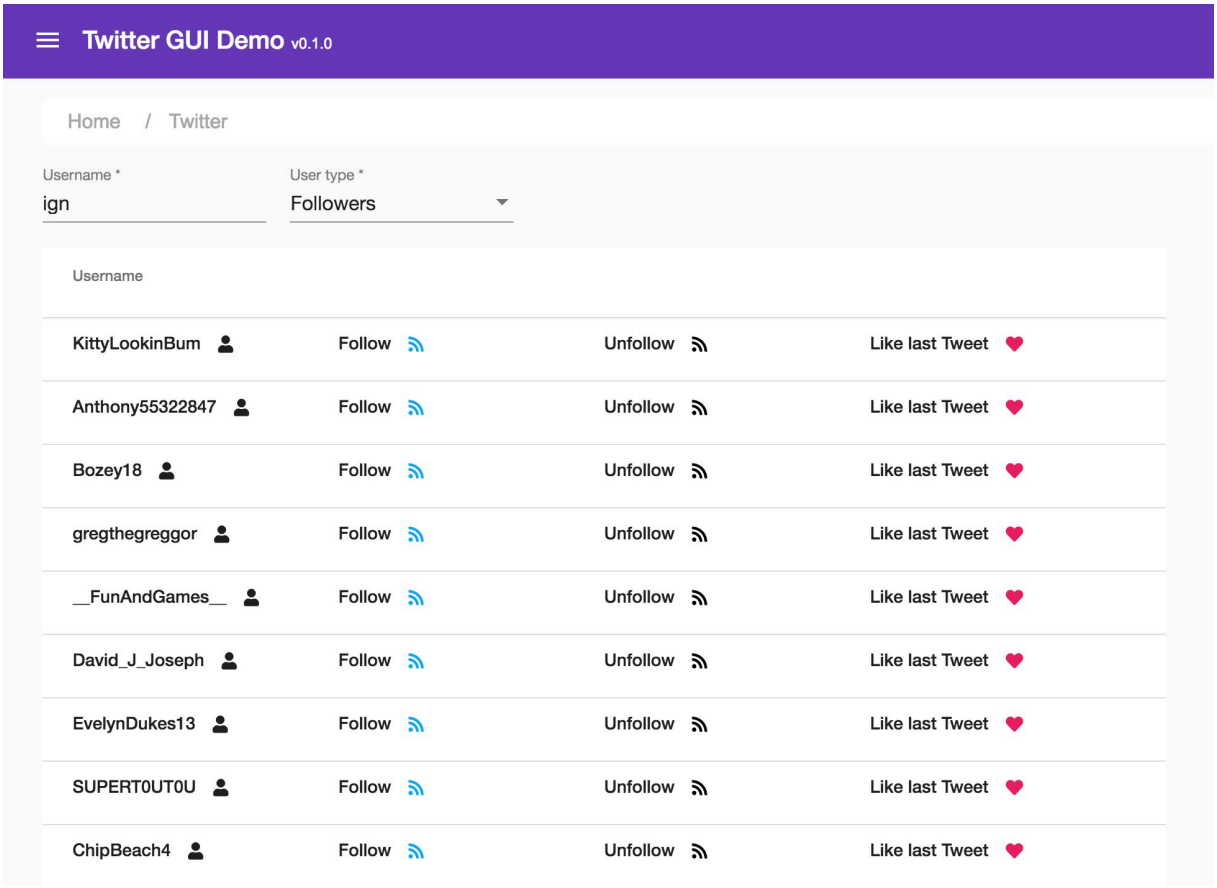
Slika 11 - Odgovor metode za brisanje ključa

Brisanjem ključa se zatvara instanca preglednika te za ponovni pristup moramo zatražiti novi ključ.

7.4.2. Uporaba programskog okvira

U ovom poglavlju će biti opisana jednostavna Angular aplikacija koja će koristiti REST servis za dohvaćanje podataka. Aplikacija se sastoji od dvije komponente. Prva komponenta je zadužena za prikaz podataka o korisnicima i za odrađivanje akcija vezanih uz te korisnike. Druga je zadužena za objavu novog Tweet-a. Kako bi aplikacija mogla raditi potrebno je upisati važeće korisničko ime i lozinku u konfiguracijsku datoteku. Upute je moguće pronaći u README datoteci na repozitoriju aplikacije (Barić, 2018b) . Aplikacija se spaja na REST servis preko predefinirano postavljenog ključa. Treba napomenuti da ova aplikacija nije namijenjena za produkcijska okruženja ili javno objavljivanje. Razlog je tome je što podatci uneseni u

aplikaciju nisu sigurni unutar JavaScript datoteke ako se ona objavi javno. Aplikacija je namijenjena za lokalno pokretanje te prikaz demo implementacije klijentske aplikacije koja konzumira razvijeni REST servis. Na slici 12 možemo vidjeti sučelje prve komponente.

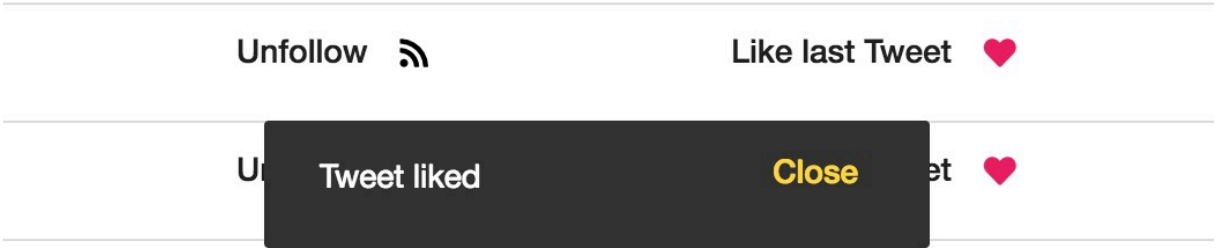


Slika 12 - Sučelje komponente za pregled korisnika

Komponenta na temelju parametra prikazuje korisnike koji prate odabranog korisnika ili korisnike koje prati odabrani korisnik tj. njegove interese. Korisnika možemo odabrati unosom njegovog Twitter korisničkog imena u polje “Username” . Nakon unosa lista se automatski osvježi. Padajuće polje služi za odabir tipa korisnika. Odnosi se na pratitelje ili interese trenutno odabranog korisnika. Treba napomenuti da aplikacija ne kontrolira niti se prijavljuje kao korisnik koji je upisan u polje “Username” nego samo prikazuje podatke o tome korisniku. Slično kao da otvorimo Twitter stranicu i pogledamo profil korisnika. Twitter aplikacije sve radnje izvršava kao korisnik prijavljen korisničkim imenom i lozinkom definiranim u konfiguracijskoj datoteci aplikacije. Za svakog korisnika u listi moguće je odraditi pojedine akcije u sklopu Twitter-a. Akcije su sljedeće:

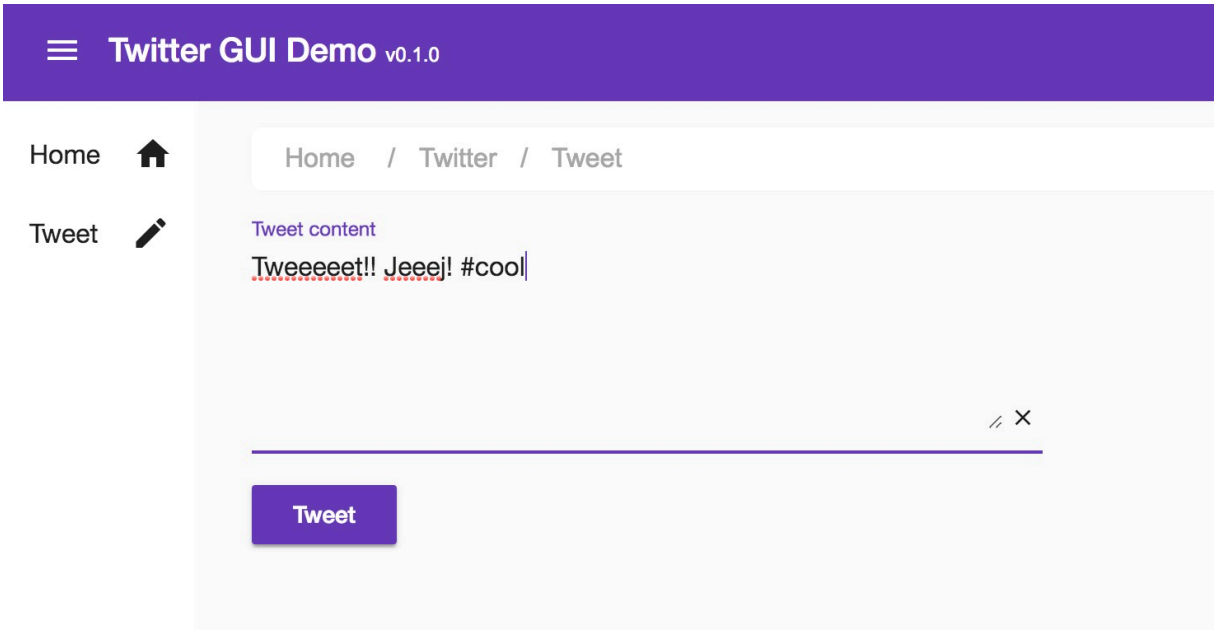
- Pogledati podatke o korisnika klikom na korisničko ime –odabir korisnika je ekvivalent upisu korisničkog imena korisnika u polje “Username”
- Početi pratiti korisnika ako ga već ne pratimo
- Prestati pratiti korisnika ako ga već pratimo
- “Lajkati” zadnji Tweet korisnika

Svaka akcija kao i dohvaćanje podataka radi zahtjev na REST servis. Aplikacije čita odgovore servisa i prikazuje ih kao notifikacije unutar aplikacije. Ako se kod nekog zahtjeva dogodi pogreška aplikacije može pročitati pogreške vraćene kao odgovor REST servisa i isto ih prikazati kao notifikaciju. Slika 13 pokazuje izgleda notifikacije.



Slika 13 - Notifikacija aplikacije

Notifikacije su implementirane kao stog te ako korisnik napravi više akcija one će se redom prikazivati kako odgovori na zahtjeve budi stizali do aplikacije. Aplikacija prikazuje samo jednu notifikaciju odjednom. Druga komponenta sadrži polje za unos teksta i linkova za Tweet. Forma ima implementiranu validaciju na dužinu znakova teksta koji je unesen za Tweet (280 znakova ograničenje na Twitter-u) . Forma ne može biti poslana ako je tekst prazan ili ne zadovoljava validaciju kako bi se ograničili loši pozivi na REST servis. Komponenta isto prikazuje notifikaciju kada je Tweet objavljen ili je došlo do pogreške. Slika 14 prikazuje sučelje druge komponente.



Slika 14 - Sučelje za objavu Tweet-a

Aplikacija sadrži i meni preko kojega je moguće prelaziti između komponenti. Aplikacije ne sadrži sve metode REST servisa zbog toga što je cilj bio prikazati osnovnu demo prezentaciju povezivanja na servis. Aplikacija osim Angular okvira koristi Material dizajn temu i stilske datoteke za organiziranje sadržaja. Integrirana je u NodeJS okruženje tako da ju je moguće instalirati preko istog okruženja kao i REST servis. Upute su u README datotekama oba projekata respektabilno.

8. Prevencije i kontrola pristupa podacima

Cilj ovog poglavlja je dublje objasniti ključne koncepte ponašanja automatiziranih Internet preglednika u odnosu na one koje koristi prava osoba. Drugi dio je pregledati dostupne modele zaštite i njihovu uspješnost. Osim toga treba pogledati i pravnu stranu ovakvog načina pristupa informacijama i posljedice koje mogu proizaći iz takvih akcija. Kako je u sklopu ovog seminarskog rada automatizacija rađena na Twitter Web aplikaciji za primjer će se uzeti u obzir pravila i uvijete korištenja te stranice.

8.1. Interakcija s automatiziranim preglednicima

Kako bi opisali razlike u interakciji između automatiziranih preglednika bez sučelja (eng. Headless) i pravih (UI based) preglednika prvo treba definirati njihove glavne razlike. Pravi preglednici ili preglednici kojima upravljamo preko korisničkog sučelja (eng. User Interface, UI) prikazuju sadržaj i daju vizualni prikaz korisničkih akcija u kontekstu internetske stranice. U slučaju automatizacije, preglednici automatiziraju radnje korisnica nad identičnom sučelju kakvom bi pristupali pravi korisnici. Preglednici prikazuju i čitaju HTML, primjenjuju stilove, izvršavaju JavaScript i slično. Većinu tih stvari preglednici bez sučelja ne moraju raditi jer na kraju nikada ne prikazuju nešto vizualno korisniku. Iz tog razloga moguće je uštedjeti na vremenu i memoriji te dobiti na brzini. Na primjer jedan jednostavan filter koji implementirani REST servis ima je da svaka Twitter stranica kojoj se pristupa automatski ne učitava slike, fontove i neke ne potrebne stilske datoteke. Ovo se odrađuje unutar "loadSession()" metode router-a:

```
await session.page.setRequestInterception(true);

session.page.on('request', request => {
    if (request.resourceType() === 'image' || request.resourceType() === 'font') {
        request.abort()
    } else if(request.resourceType() === 'stylesheet' &&
        request.url().indexOf('twitter_core.bundle.css') === -1) {
        request.abort()
    } else {
        request.continue()
    }
});
```

Možemo vidjeti da postavljamo funkciju koja odrađuje filtriranje nad događaj koji šalje stranica tipa "request". Svaki puta kada stranica napravi mrežni zahtjev ova metoda će se izvršiti i odlučiti da li zahtjev može dalje ulančati ili ga treba otkazati. Na ovaj način se uštedi i po sekunda na vremenu odgovora. Zaključak bi bio da su za automatizaciju daleko bolji preglednici, a Headless Chrome je po provedenim testovima daleko brži u učitavanju stranica, a troši i manje memorije (Hatator, 2017) . Jedna velika prednost nad ostalim automatiziranim pristupima (scraperi i crawleri) je da što se tiče serverske strane pristup preko preglednika bez sučelja i sa sučeljem je u potpunosti ista. Navigacija kroz stranicu radi stvarne akcije koji opet čak mogu biti registrirani preko analitičkih programa kao Google Analytics ako ih stranica integrira. Isto tako razne metode prevencije automatizacije često traže znakove automatizacije upravo u načinu na koji se stranici pristupa. Na primjer korisnik će povući miš na gumb, kliknuti gumb i otići na sljedeću stranicu. Dok će automatizirani programi to napraviti bez među koraka. Upravo zato je teže detektirati automatizaciju preko preglednika nego druge načine automatizacije i prikupljanja podataka.

8.2. Metode prevencije

Koliko god automatizacija bila korisna nekada jednostavno ne želimo da netko na ovaj način zlostavlja ili ilegalno koristi i skuplja sadržaj s naših web stranica. U ovom poglavlju će biti navedene metode, smjernice i upute kako onemogućiti ili barem otežati automatiziranim Internet preglednicima da pretražuju i izvlače podatke s web stranica. Kako je automatizirano prikupljanje podatka širi pojam definirat će se samo metode koje su najkorisnije za onemogućavanje pristupa automatiziranim preglednicima. Neke od metoda prevencija su preuzete s GitHub stranice autora (JonasCz, 2017) .

8.2.1. Pregled mrežnog prometa, logova i ograničavanje broja zahtjeva

Automatizirani preglednici često rade veći broj zahtjeva od običnog korisnika. Zbog toga je moguće pratiti i detektirati veći broj zahtjeva s neke IP adrese. Kako bi se implementirao ovakav način zaštite potrebno je uložiti u analitiku i praćenje serverskog prometa. Srećom danas mnogi web serveri pružaju tu i ili slične funkcionalnost bez ikakvih dodataka (Chang, 2017) . Razni pružatelji usluga kao što su DigitalOcean, Amazon AWS, Cloud Flare i drugi pružaju automatsku zaštitu, ograničavanje broja zahtjeva (eng. Rate Limiting) . Često pružaju

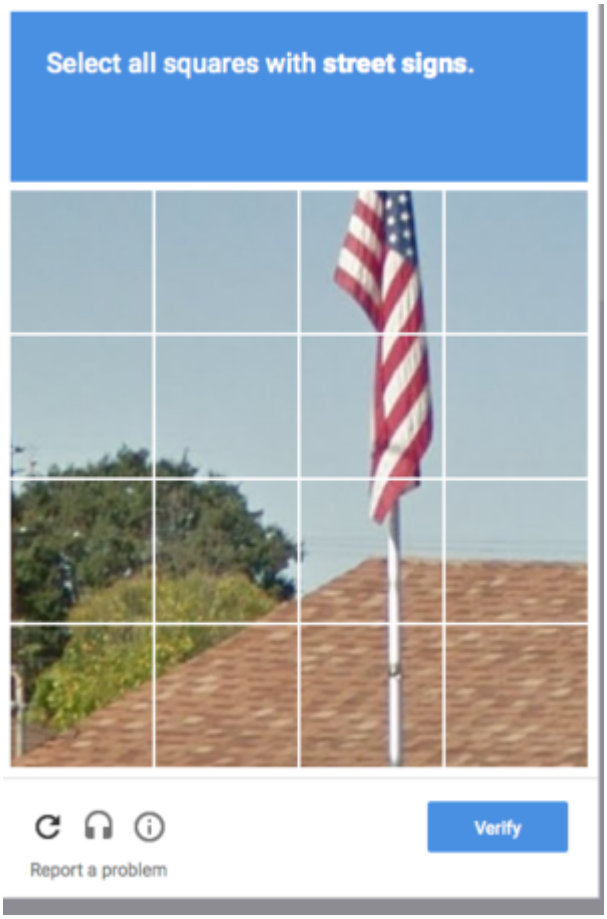
i detaljne logove pristupa tako da je moguće i kasnije analizirati neki specifični slučaju i ručno blokirati IP adresu ili spektar IP adresa.

8.2.2. CAPTCHA upitnici

CAPTCHA (eng. Completely Automated Public Turing test to tell Computers and Humans Apart) je poznata metoda prevencije pristupa sadržaju "botovima". Kao pojam je došla 2003. Godine (Wikipedia, 2018h). Bazira se na testu koji korisnik mora proći kako bi došao do određenog sadržaja na stranici. Najčešće je to prikaz slike tekstualnog sadržaja koji je potrebno pročitati i upisati u polje ispod. Ako je upisani tekst isti kao i onaj zapisan na serverskoj strani korisnika se pušta dalje, a ako nije dobiva se nova slika s tekstom ili se nakon većeg broja pogrešnih odgovora može iskoristiti neka od metoda iz prijašnjeg poglavlja. Problem s rješavanjem je da tekst nije baš čitljiv i često je problematičan čak i za prave ljude. Pogotovo one sa slabijim vidom ili one starije sa slabijim sposobnostima zapažanja. Ovaj način je naravno u današnje vrijeme prepoznavanja slika pomoću strojnog učenja i sličnih metoda postao malo ne efikasan.

Slika 15 - Primjer CAPTCHA upitnika

Danas CAPTCHA-e ne moraju biti samo slike s tekstom. Na primjer vrlo popularna i široko korištena je CAPTCHA razvijena od strane kompanije Google pod nazivom reCAPTCHA. Glavna funkcionalnost je da se korisniku osim pruženog testa uzimaju u obzir i drugi parametri kao što su ponašanje korisnika na stranici (pokreti miša, pritisci gumbova itd.) , brzina pristupanja sadržaju, status prijave na društvene mreže preko oAuth 2.0 servisa kao što su Facebook Login, Google Login i slični itd.



Slika 16 - Prikaz reCAPTCHA upitnika

Prilagođena je i korisnicima s invaliditetom, gluhima, a dostupna je i za mobilne platforme uz naravno web platformu. Upravo praćenje korisničkih radnji na stranici dosta dobro može detektirati automatizirane preglednike jer će oni često skakati s mjesta na mjesto i brže kliktati zbog prirode automatizacije.

8.2.3. Korištenje slika kao tekstova

Ova metoda je malo upitna i zastarjela, ali ideja je vrlo jednostavna. Naime ideja je da se na serverskoj strani bitni tekstovi pretvore u slike JPEG, PNG ili nekog drugog slikovnog formata. Tako je moguće onemogućiti izvlačenje i direktno kopiranje teksta. Isti razlog zašto nije primjenjivo je taj da se pomoću metoda strojnog učenja lako može iz slike detektirati i pročitati napisani tekst. Druge mane su otežavanje korištenja stranice za prave korisnike. Isto tako stranica će biti znatno memorijski "teža" i sporije će se učitavati.

8.2.4. Često mijenjanje strukture stranice

Metoda koja zvuči trivijalno, ali je jako logična svakome tko se ikada bavio ili je izrađivao automatiziranog bota-a za prikupljanje podataka s web stranica. Kako je prije opisano automatizirani preglednik traži određene HTML elemente i CSS selektore kako bi onda izvršio akcije nad njima ili uzeo njihov sadržaj. Ako se struktura stranice promjeni, programer mora ponovno napisati ili doraditi skriptu za automatizaciju kako bi pravilo prolazila stranicom i odradila potrebne akcije. Ako se struktura stranice stalno mijenja onda posao dorade skripti za automatizaciju postaje prevelik da bi opravdao pogodnosti koje pruža automatsko prikupljanje podataka. Implementacije ove prevencije su različite. Moguće je na primjer u određenom periodu ili kod svake nove pod verzije stranice zamijeniti sve CSS klase s nasumičnim alfanumeričkim identifikatorom. Na primjer klasa "main-heading" će postati "06132d1b7a4fcb62f635faf44c324cc6b0117e7b" i slično. Isto tako moguće je mijenjati redoslijed HTML tag-ova i strukture DOM-a. Na primjer s modernim CSS-om i HTML-om je moguće kreirati identičan raspored na više načina. Mana ovoga svega je da i programerima koji žele zaštititi stranicu posao postaje teži radi stalne potrebe za izmjenama koda, testiranjem i slično. Naravno i sve ostale metode obfuskacije koda gdje je teže i programeru automatiziranog preglednika razumjeti strukturu i način funkcioniranja stranice su u nekoj mjeri učinkoviti.

8.2.5. Dodavanje i posluživanje lažnog sadržaja

Odnosi se na unošenje lažnog sadržaja kako bi se "zaprljali podatci" koje automatizirani preglednik skuplja. Pretpostavka je da automatizirani preglednik ne razumije podatke koje skuplja te samo prati niz naredbi iz skripte u kojoj je napisan. Tako onda možemo uz prave podatke na nasumičnim mjestima dodati HTML elemente s lažnim sadržajem. Takav sadržaj možemo sakriti s CSS stilovima kako ga običan korisnik ne bi vidio. Moguće je lažni sadržaj doraditi da služi kao svojevrsna zamka (eng. Honeypot) . Njezin zadatak je da oponaša stvarne podatke ili stvarnu akciju na stranici. Na primjer ako imamo popis članaka i svaki članak ima link na svoj tekst. Možemo unutar toga ubaciti sakriveni članak koji sadržava link na posebnu stranicu koja ako je posjećena čita IP adresu s koje je došao zahtjev i blokira ju na primjer sljedeća 24 sata. Mana ovog pristupa je da na primjer automatizirani Chrome preglednik kroz Puppeteer sučelje čak ima opciju da ne radi interakciju s elementima koji su ne vidljivi i skriveni preko CSS-a (vizualno) . Naravno programer mora vidjeti ovakve lažne elemente i onda programirati preglednik da tako postupa, ali nekada je dovoljno i samo otežati pristup toliko da izgubi svoju profitabilnost.

8.2.6. Praćenje broja zahtjeva

Praćenje se temelji na uspoređivanju seta resursa tj. zahtjeva za njima koje traži normalni posjet aplikaciji i traženje sumnjivih razlika. Treba naravno paziti da razlike ne budu zbog prirode uređaja s kojega korisnik pristupa stranici. Neke od sumnjivih ponašanja bi bila:

- Klijent ne zahtijeva neke ili sve slike i ili multimedijalne elemente. U samom kodu REST servisa unutar ovoga rada je korištena metoda gdje se slike ne zahtijevaju kako bi se dobilo na brzini izvođenja automatiziranih akcija.
- Klijent ne traži stilske datoteke
- Klijent ne traži datoteke font-ova (fontovi su često znatnije veličine)
- Repetitivno traženje većeg broja sadržaja u kratkom vremenu koje nije tipično za web adresu podrijetla zahtjeva - ovo se može nadovezati na metode iz poglavlja 8.2.1.

8.3. Poštivanje uvjeta korištenja

Internet aplikacije koje pružaju usluge korisnicima, zadržavaju ili procesiraju njihove podatke ili imaju bilo kakav oblik poslovnog odnosa imaju definirane uvijete korištenja. Oni definiraju prava koje korisnici moraju poštovati kako bi koristili usluge na web stranici (Wikipedia, 2018i) . Uz uvijete korištenja postoje pravila i politike korištenja. Dio iz pravila Twitter web aplikacije (Twitter, 2018a) koji će biti opisan u ovom poglavlju tiče se automatiziranog pristupa Twitter servisima i web aplikaciji. Pravila (Twitter, 2018b) definiraju točke u dvije kategorije. To su kategorije dozvoljenih i ne dozvoljenih stvari. Dozvoljeno je:

- Kreiranje rješenja koja automatski objavljuju korisne informacije preko Tweet-ova
- Vođenje kreativnih kampanja koje automatski odgovaraju korisnicima koji se počnu interakciju sa sadržajima
- Kreiranje rješenja koja automatski odgovaraju na poruke
- Kreiranje novih servisa koji pomažu ljudima (mora poštovati pravila kompanije Twitter)
- Treba omogućiti da aplikacija pruža dobro korisničko iskustvo (eng. User Experience)

Sve ove točke se direktno odnose na direktno korištenje Twitter-ovi servisa preko kojih radi i Twitter korisničko sučelje tako zabranjuju sljedeće aktivnosti:

- Zabranjeno je kršenje ovih i drugih pravila
- Zlostavljanje oficijelnih Twitter-ovi API servisa ili pokušaj zaobilaženja restrikcija

- Korištenje drugih rješenja za automatizaciju, kao na primjer skriptiranje Twitter stranice - ovakve aktivnosti mogu rezultirati trajnim suspendiranjem korisničkog računa
- Neželjena pošta ili bilo kakav drugi oblik zlostavljanje drugih Twitter korisnika

Korištenje automatizacije web preglednika je zaobilaženje korištenja oficijelnih kanala i pristupa resursima Twitter aplikacije. Uzevši to u obzir automatizacija preglednika ne koristi aplikaciju na niti drugačiji način od onoga kojeg bi mogao koristiti pravi korisnik. Unatoč tome automatizacija može brže i konstantnije u više iteracija obaviti radnje i treba paziti da se na ovaj način ne prekrši pravilo o zlostavljanju Twitter servisa. Iz tog razloga su u sklopu REST servisa ove aplikacije uvedeni ključevi koji sprječavaju mogućnost zlostavljanja i svaki zahtjev je autoriziran prije izvršavanja automatizacije. Zlostavljanje se može ostvariti odrađivanjem velikog broja simultanih akcija odjednom. Takve akcije često nije moguće napraviti preko normalnog sučelja ili normalnim korištenjem. Ove metode unutar Twitter API-a razvijenog u sklopu ovog projekta potencijalno daju mogućnost zlostavljanja u tom pogledu:

- Like recent tweet - moguće je u kratkom roku "lajkati" zadnje Tweet-ove velikog broja korisnika. Ovo može drugom korisniku proizvesti veći broj notifikacija ili obavijesti na email adresu
- Followers - Moguće je pregledati i dohvatiti sve korisnike koji prate određenog korisnika
- Interests - Moguće je pregledati i dohvatiti sve interese tj. korisnike koje prati određeni korisnika
- Follow network - Moguće je u kratkom vremenu početi pratiti veliki broj korisnika i onda isto napraviti za svakog novo praćenog korisnika i tako u neodređeno dugom vremenu
- Follow interests – Isto kao i prethodna metoda

Ovaj rad je napisan striktno u istraživačke svrhe te niti jedna od metoda nije bila korištena u svrhe zlostavljanje drugih Twitter korisnika. Svi testovi su odrađeni na malom broju korisnika i na manjem broju zahtjeva kako bi se testirala i dokazala pretpostavka na početku ovoga rada.

9. Zaključak

REST servisi su danas postali najčešći izvor dohvaćanja podataka koji se prikazuju u web aplikacijama. Korisnici su navikli na stranice koje su brze, učitavaju se jednom i nikada ne prekidaju svoj tok. Isto tako integracija raznih servisa unutar aplikacija koje razvijamo je postala česta praksa. Isto tako često integracija ovisi o REST servisima koje daje na raspolaganje pružatelj tih usluga. Uz napredak tehnologije i novih metoda više ne moramo nužno ovisiti i ograničavati se na trenutne standarde razvoja. Automatizacija je uvijek bila vrlo značajan pojam u razvoju programskih proizvoda i olakšava mnoge procese. Automatizirani preglednici i preglednici koji se mogu koristiti bez korisničkog sučelja pružaju niz mogućnosti za nove načine pristupa podatcima na internetu. Ovakvi preglednici se danas koriste već u mnoge druge svrhe kao što su automatizirano testiranje stranica, a efikasnost u tome baš pruža potencijal za automatizirano prikupljanje podataka sa stranica na isti način. Modularni sustav razvijen u sklopu ovog seminarskog rada moguće je lako proširiti i koristiti za korištenje u druge svrhe te biti prilagođen drugim i kompleksnijim ponašanjima korisnika. Prilagođeni Twitter API prikazuje koncept razvoja takvog sučelja za društvenu mrežu Twitter, ali isto tako ista metodologija i kombinacija tehnologija može biti korištena za razvoj sličnih API-a za druge društvene mreže, a ako je API razvijen po istim smjernicama automatizirani preglednici će isto tako raditi s drugim društvenim mrežama. Slabost ovakvog tipa prikupljanja podataka je svakako ovisnost o razumijevanju strukture pravog sustava (društvene mreže) i sposobnost pojedinca/programera da pravilno prilagodi ponašanje agenta zadanim ponašanjima korisnika. Korak dalje bi bio implementiranje samo zaključivanja i strojnog učenja temeljem podataka koje prikuplja kako bi samostalno mijenjao svoje ponašanje ovisno o promjenama stanja mreže reagirao i mijenjao akcije automatizacije. Dakako ovo bi zahtijevalo implementaciju neke vrste algoritma strojnog učenja, ali takav pothvat je zapravo projekt sam za sebe. Mislim da implementacija ovakvog programskog okruženja opravdava početni cilj rada te odgovara na pitanje da li je moguće koristiti automatizirane preglednike za automatizirano prikupljanje podataka u stvarnom vremenu. Društvena mreža je uzeta za primjer je baš ako ovaj način radi u dinamičnom okruženju društvenih mreža onda su jednostavnije stranice puno lakše za implementaciju. Treba napomenuti da je tehnologija DevTools protokola i nekih drugih metoda korištenih unutar ovog rada još uvijek u fazama ranog razvoja te će svakako još više napredovati pa će i implementacija ovakve vrste agenata isto tako biti sve pristupačnija. To će po mogućnosti pomoći daljnjoj standardizaciji ovakvih metoda i većem broju programera koji će ih koristiti.

Popis literature

1. Barić Ante 2018a, capJavert/twitter-api, Dostupno na https://github.com/capJavert/twitter-api [2.5.2018]
2. Barić Ante 2018b, capJavert/twitter-api-demo-gui, Dostupno na https://github.com/capJavert/twitter-api-demo-gui [2.5.2018]
3. Browser Market Share 2018, Browsers Market Share June 2018, Dostupno na https://netmarketshare.com/browser-market-share.aspx?options=%7B%22filter%22%3A%7B%22%24and%22%3A%5B%7B%22deviceType%22%3A%7B%22%24in%22%3A%5B%22Desktop%2Flaptop%22%5D%7D%7D%5D%7D%2C%22dateLabel%22%3A%22Trend%22%2C%22attributes%22%3A%22share%22%2C%22group%22%3A%22browser%22%2C%22sort%22%3A%7B%22share%22%3A-1%7D%2C%22id%22%3A%22browsersDesktop%22%2C%22dateInterval%22%3A%22Monthly%22%2C%22dateStart%22%3A%222017-05%22%2C%22dateEnd%22%3A%222018-04%22%2C%22segments%22%3A%22-1000%22%7D [2.5.2018]
4. Chang Emily 2017, Monitoring Apache web server performance, Dostupno na https://www.datadoghq.com/blog/monitoring-apache-web-server-performance/ [2.5.2018]
5. Google 2018a, Google Chrome, Dostupno na https://www.google.com/chrome/ [2.5.2018]
6. Google 2018b, Chrome DevTools Protocol Viewer, Dostupno na https://chromedevtools.github.io/devtools-protocol/ [2.5.2018]
7. Google 2018c, Tools - Puppeteer, Dostupno na https://developers.google.com/web/tools/puppeteer/ [2.5.2018]
8. Hacker News 2016, Never trust the client, Dostupno na https://news.ycombinator.com/item?id=11583008 [2.5.2018]
9. Hámori Ferenc 2017, This is what Node.js is used for in 2017 - Survey Results, Dostupno na https://blog.risingstack.com/what-is-node-js-used-for-2017-survey/ [2.5.2018]
10. Hartator 2017, Headless Chrome vs PhantomJS Benchmark, Dostupno na https://hackernoon.com/benchmark-headless-chrome-vs-phantomjs-e7f44c6956c [2.5.2018]

11. JonasCz 2017, How-To-Prevent-Scraping, Dostupno na https://github.com/JonasCz/How-To-Prevent-Scraping [2.5.2018]
12. Mayko Lexy 2015, HTTP/2: the Pros, the Cons, and What You Need to Know, Dostupno na https://www.sitepoint.com/http2-the-pros-the-cons-and-what-you-need-to-know/ [2.5.2018]
13. Mozilla 2018, MDN web docs - JavaScript, Dostupno na https://developer.mozilla.org/bm/docs/Web/JavaScript [2.5.2018]
14. SEO Hacker 2017, HTTPS Now Mandatory for Secure Data in Chrome, Dostupno na https://seo-hacker.com/https-mandatory-secure-data-chrome/ [2.5.2018]
15. Twitter 2018a, Rules and Policies, Dostupno na https://help.twitter.com/en/rules-and-policies [2.5.2018]
16. Twitter 2018b, Rules and Policies – Twitter Automation, Dostupno na https://help.twitter.com/en/rules-and-policies/twitter-automation [2.5.2018]
17. Wikipedia 2018a, HTTP2, Dostupno na https://en.wikipedia.org/wiki/HTTP/2 [2.5.2018]
18. Wikipedia 2018b, Browser Security, Dostupno na https://en.wikipedia.org/wiki/Browser_security [2.5.2018]
19. Wikipedia 2018c, Blink (web engine), Dostupno na https://en.wikipedia.org/wiki/Blink_(web_engine) [2.5.2018]
20. Wikipedia 2018d, Cron, Dostupno na https://en.wikipedia.org/wiki/Cron [2.5.2018]
21. Wikipedia 2018e, Event driver programming, Dostupno na https://en.wikipedia.org/wiki/Event-driven_programming [2.5.2018]
22. Wikipedia 2018f, Express.js, Dostupno na https://en.wikipedia.org/wiki/Express.js [2.5.2018]
23. Wikipedia 2018g, Twitter, Dostupno na https://en.wikipedia.org/wiki/Twitter [2.5.2018]
24. Wikipedia 2018h, CAPTCHA, Dostupno na https://en.wikipedia.org/wiki/CAPTCHA [2.5.2018]
25. Wikipedia 2018i, Terms of service, Dostupno na https://en.wikipedia.org/wiki/Terms_of_service [2.5.2018]

Popis slika

Popis slika treba biti izrađen po uzoru na indeksirani sadržaj, te upućivati na broj stranice na kojoj se slika može pronaći.