SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Multi agentni sustav u okruženju društvenih mreža

Seminarski rad

Ante Barić
br. indeksa: 45312/16-R

Mentor:
Doc. dr. sc. Markus Schatten

Varaždin, 13. siječnja 2018.

Sadržaj

Poglavlje 1

Uvod

Društvene mreže su danas jedan od najviše korištenih načina komunikacije u današnjem modernom društvu. Ljudi koriste društvene mreže i alate koje isti pružaju uz mnoge svakodnevne radnje. Na primjer student šalje poruku curi na Facebook da će kasniti na spoj jer je dugačak red u menzi, domaćica je našla odlične svježe banane na akciji u konzumu te piše status o tome, čovjek dok putuje kući s posla pregledava naslovnicu i vidi da je prijatelj popratio stranicu za novi film Christiana Bale-a itd. Sve ove interakcije se baziraju na tome da nam društvene mreže pružaju alate da podijelimo svoje životne trenutke, razmišljanja i interese sa drugim ljudima na mreži. Ti isti ljudi onda reagiraju ili dijele svoja mišljenja i na taj način se informacije šire društvenim mrežama. Širenje informacija se u biti događa akcijama korisnika (objava statusa, praćenjem stranica) i reakcijama drugih korisnika na te akcije. Način reagiranja korisnika i akcije koje on poduzima ovisno o nekom događaju možemo karakterizirati kao njegovo ponašanje. Unatoć tome što društvene mreže imaju svoja pravila, funkcionalnosti i definirane načine interakcija svaki korisnik naravno može odabrati kako će te funkcionalnsti koristiti tj. kako će se ponašati na društvenoj mreži. Isto tako danas postoje razni tipovi društvenih mreža. Na primjer neke su orijentirane na slike npr. Instagram, dok su druge kao na primjer Reddit orijenirane na vijesti i raspravu, a neke su više funkcionalne kao na primjer Facebook. Kako društvene mreže postoje na internetu u pristupa im se sa raznih uređaja (mobiteli, laptopi, računala, pametni satovi, tableti itd.) logično je bilo da se neki aspekti istih integriraju u aplikacije na tim uređajima. Vrlo popularna funkcionalnost je tako "Social login" gdje preko računa na društvenoj mreži možemo pristupiti i kreirati račun na raznim stranicama, mobilnim aplikacijama i slično. Naravno postoje i drugi oblici integracija društvenih mreža u aplikacije, npr. web aplikacije ili web stranice mogu imati implementirano komentiranje na pojedine stranice preko društvene mreže te se onda isti komentari osim na stranici mogu vidjeti i komentirati i sa društvene mreže i obratno. Svi ovi načini koegzistiranja aplikacija i funkcionalnosti društvenih mreža zahtjeva nekakvo programsko sučelje preko kojega bi aplikacija pristupala društvenoj mreži i obrnuto. U tu svrhu danas mnoge društvene mreže imaju programska sučelja ili skraćeno API-e koje developeri koriste kako bi ostvarili gore navedene funkcionalnosti. Jedna od ograničenja tih API-a je naravno da nisu sve funkcionalnosti društvenih mreža izložene preko njih jer naravno to bi značilo da svako može napraviti ili prekopirati funkcionalnosti društvenih mreža i koristiti ih unutar vlastitoga sučelja. Često funkcionalnosti koje nisu izležene preko API-a su uređivanje profila, objava statusa, slanje internih poruka unutar društvene mreže i slično. U biti sve funkcionalnosti koje zapravo tvore i definirani prije spomenuta ponašanja korisnika na društvenoj mreži nisu izložena. U svrhu toga ideja ovoga rada je izraditi programsko sučelje za pristup društvenim mrežama te implementacijom agenata simulirati ponašanja tj. akcije i reakcije korisnika na događaje. Kako sve društvene mreže imaju web sučelja agenti

će preko instance internet preglednika navigirati društvenim mrežama te obavljati definirana ponašanja. U sklopu rada kao primjer će biti uzeta Twitter društvena mreža zbog jednostavnosti i usmjerenosti na tako zvane "tweetove" koji su glavni sadržaj na istoj. Isto tako biti će implementirano nekoliko agenata sa različitim ponašanjima. Cilj je kreirati modularnu i proširivu platformu za kreiranje agenata za društvene mreže te istražiti mogućnosti primjene istih u sklopu automatizacije akcija, prikupljanja podataka te promatranja i interakciji sa pravim korisnicima.

Poglavlje 2

Društvene mreže i agenti

2.1 Društvena mreža Twitter

Twitter je društvena mreža namjenjena za čitanje i slanje kratkih poruka koje su prema imenu mreže nazvane "tweet". Svaki tweet je ograničen na 280 znakova (do nedavno je to bilo 140 znakova, ali zbog stalnih zahtjeva korisnika i proširenja mreže broj znakova je povećan) . Twitter je osnovan od strane Jack Dorsey, Noah Glass, Biz Stone i Evan Williams i servis je ubrzo postao sve više popularan, a u 2013. tvrtka je izašla na burzu. Twitter je najviše poznat u Americi, ali je zadnjih godina sve više i više popularan i u Europskim zemljama i Hrvatskoj. (Wikipedia, 2017)

2.2 Simulacija društvene mreže pomoću agenata

Knjiga (Adelinde M. Uhrmacher, 2009) govori kako je česta primjena više agentnih sustava je za simuliranje te se simulacije koriste za agente u različitim aplikacijskim domenama. Simulacije u aplikaciji se koriste za prikaz ponašanja agenata, a agenti se koriste za simulacije okruženja u kojem aplikacija treba raditi. Društvene mreže su kako sam prije rekao odlična domena primjene agenata jer agenti mogu simulirati ponašanje korisnika na mreži npr. u svrhu testiranja novih funkcionalnosti ili testiranja korisničkog doživljaja i odluka. Isto tako društvena mreža može biti korištena za definiranje ponašanja agenta prema specifičnom korisničkog profilu. Prema knjizi (Adelinde M. Uhrmacher, 2009) simulacija je alat koji služi za postizanje dvije stvari od kojih ne moraju obje biti cilj:

1. Razumijevanje stvarnog sustava koji se simulira;
2. Razvoj funkcionalnog stvarnog sustava.

U sklopu ovoga rada funkcionalni stvarni sustav bi bila društvena mreža Twitter, a cilj bi bio razumijevanje stvarnog toga sustava to jest ponašanja korisnika unutar društvene mreže. Korištenjem agenata moguće je simulirati ponašanje korisnika i onda promatrati ista u zatvorenim uvjetima koji ne bi bili mogući u stvarnom okruženju koje je više promjenljivo.

2.3 Prikupljanje podataka

Osim simuliranja ponašanja korisnika preko agenata, korištenje agenata u okruženju društvene mreže mogu imati i ulogu Crawlera (Wikipedia, 2018) u svrhu prikupljanja podataka. Kako agenti izvršavaju

svoja definirana ponašanja mogu voditi zapise o tome ili na primjer na primjeru Twittera prikupljati podatke o tome koga korisnik prati, što objavljuje, što mu se sviđa i slično. Ono što daje prednost ovakvom načinu prikupljanja podataka je činjenica da je moguće modul za prikuplanje podataka ugraditi unutar agenta i nije potrebno drugim kanalima pratiti korisnika i njegovo ponašanje nego zato što agent simulira ponašanje korisnika može stvarati i konzumirati podatke na sličan ili isti način kao što bi to korisnik radio te ujedno prikupljati i zapisivati iste. Kako bi agent bio kvalitetan crawler članak (Chi-In Wong, 2014) potrebno je analizirati i razumjeti strukturu i interakcije sudionika (korisnika) unutar zadanog sustava (društvene mreže) Za razumijevanje strukture društvene mreže poslužiti ćemo se jednostavnom teorijom grafova u sljedećeoj sekciji.

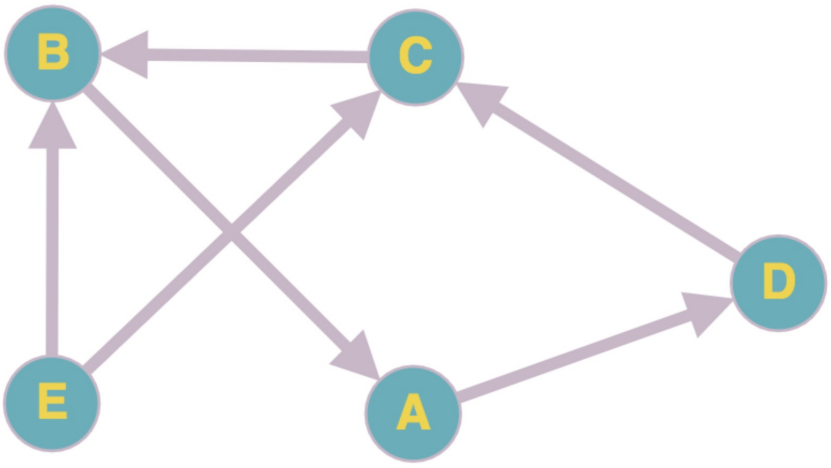
Poglavlje 3

Interakcije i ponašanje korisnika

Prikazati strukturu i odnose između svih korisnika na društvenoj mreži Twitter bilo bi vrlo vremenski zahtjevno i vjerojatno ne moguće odraditi precizno te prikazati na pregledan način, barem ne u okvirima ovoga rada. Ono što se može prikazati je pojednostavljeni prikaz koji prazi osnovne funkcionalnosti društvene mreže. Na Twitteru su to:

1. Objava tweet-a
2. "Like" pojedinog tweet-a
3. Ponovno objavljivanje tweet-a drugog korisnika
4. Pračenje drugih korisnika
5. Pretraživanje trendova, društvene mreže

Kombinacijom ovih radnji možemo profilirati osnovne tipove korisnika na društvenoj mreži Twitter. Jedan takav prikaz se nalazi na sljedećoj slici.



Slika 3.1: Primjer interakcija i ponašanja korisnika

Slika prikazuje odnose 5 korisnika ili u ovom slučaju agenata. Gledajući njihove interakcije možemo definirati ponašanja. Za pojednostavljenje ovdje ćemo pojednostaviti interakcije na pružanje/stvaranje podataka i primanje/konzumiranje podataka. Tako na primjer agent može objaviti novi tweet i to je

stvaranje podataka za druge agente dok drugi agent može pročitati i "like-ati" taj tweet što je primanje tj. konzumiranje podataka od drugog agenta. Kako bi se ovi odnosi lakše pretočili u logiku programskog agenta ovaj graf se još može prikazati preko sljedeće matrice susjedstva:

	A	**B**	**C**	**D**	**E**
A	0	0	0	1	0
B	1	0	0	0	0
C	0	1	0	0	0
D	0	0	1	0	0
E	0	1	1	0	0

Tablica 3.1: Matrica susjedstva za Sliku 3.1

Važno je još napomenuti da svaka od ovih veza teoretski može predstavljati više različitih događaja tj. interakcija. Na primjer agent može kreirati podatke tweetanjem, ali i retweetanjem objava drugih korisnika. Isto tako agent može konzumirati podatke na više načina. Autori u knjizi (Wikipedia, 2018) govore o vezama između korisnika na Facebook-u i kako su prijateljstva ne usmjerene veze jer na Facebook-u korisnici mogu biti prijatelji samo ako obojica na to pristanu to jest jedan pošalje zahtjev za prijateljstvo, a drugi ga prihvati. Na Twitteru je odnos "prijateljstva" malo drugačiji jer je osnovna veza između korisnika tako nazvano praćenje. Korisnik X može pratiti korisnika Y, ali korisnik Y ne mora pratiti korisnika X. Zbog toga su prijateljstva na Twitteru usmjerene veze.

Poglavlje 4

Programsko sučelje

Kako bi agenti mogli pristupati Twitter web aplikaciji bilo je potrebo razviti programsko sučelje. Twitter inaće ima vlastito programsko sučelje za pristup (Twitter, 2018), ali to sučelje kako sam prije rekao ne pruža sve funkcionalnosti potrebne za rad agenata. U tu svrhu razvio sam programsko sučelje u jeziku JavaScript. (Barić, 2018). Osim programskog sučelja postoji REST api implementacija koju je moguće postaviti kroz jednostavni HTTP server i koristiti sa bilo kojom drugom aplikacijom ili programskim jezikom.

4.1 Twitter API

Programsko sučelje Twitter API razvijeno u sklopu ovoga seminarskog rada sadrži sljedeće metode:

1. **Login** - Prijavi se u Twitter web aplikaciju kao određeni korisnik
2. **Follow** - prati odabranog korisnika
3. **Unfollow** - prestani pratiti odabranog korisnika
4. **Like tweet** - Favoriziraj određeni tweet odabranog korisnika
5. **Like recent tweet** - Favoriziraj sve nedavno objavljene tweet-ove odabranog korisnika
6. **Like last tweet** - Favoriziraj zadnji tweet odabranog korisnika
7. **Retweet** - Objavi određeni tweet odabranog korisnika
8. **Retweet last** - Objavi zadnji tweet odabranog korisnika
9. **Followers** - Dohvati sve korisnike koji prate odabranog korisnika
10. **Interests** - Dohvati sve korisnike koje odabrani korisnik prati
11. **Follow network** - Prati sve korisnike koji prate odabranog korisnika
12. **Follow interests** - Prati sve korisnike koji odabrani korisnik prati
13. **Logout** - Odjavi se iz Twitter web aplikacije kao određeni korisnik

Za realizaciju ovih metoda koristi se programsko sučelja za headless chrome te navigira i klika linkove unutar Twitter web aplikacije kao da to radi stvarni korisnik. Jedna mana ovakvog pristupa je da ukoliko Twitter znatnije promjeni strukturu svoje web aplikacije API više neće raditi i biti će potrebno ponovno prilagoditi sučelje da ispravno navigira po web aplikaciji. Cijelokupno programsko okruženje je dokumentirano i postavljeno na javni repozitorij (Barić, 2018) .

4.2 Headless Chrome

Kako bi pregledavali i koristili Web aplikacije i stranice obično koristimo preglednike koji dobavljaju sadržaj stranice preko HTTP/HTTPS protokola te stvaraju i prikazuju sadržaj stranice čitajući HTML kod stranice i u nekim slučajevima izvršavaju JavaScript kod za manipuliranje i mjenjanje nekih dijelova stranice ovisno o korisnikovima radnjama. Agenti razvijeni u sklopu ovog seminarskog rada će stranicama isto tako pristupati preko preglednika, točnije posebne verije Chrome web preglednika pod nazivom Chromium (Google, 2018b) . Tim koji radi na Chromium pregledniku je prošle godine izdao programsko sučelje Puppeteer (Google, 2018c) za developer tools alat unutar Chrome/Chromium preglednika pod nazivom DevToolProtocol (Google, 2018a) . Ovo programsko sučelje omogućava kontrolu i pristup funkcijama developer tools alata kroz programski jezik JavaScript. Tako onda imamo pristup DOM-u trenutne otvorene stranice web aplikacije, pretraživanju i prikupljanju podataka stranice, izvršavanje programskog koda i funkcija unutar stranice i slično. Svaki agent kod svoga instanciranja kreira instancu Chromium preglednika. Nakon toga putem razvijenog Twitter API-a (Barić, 2018) navigira Twitter web aplikacijom kao da to radi stvarni korisnik unutar svog web preglednika. Primjer koda koji koristi metode iz Puppeteer biblioteke:

```
...

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");

...
```

Sve metode se pozivaju nad instancom klase Page koja predstavlja jedan otvoreni tab u instanci Chromium web preglednika. Metode kao Page.type, Page.click prema selektoru predanom kroz argument odrađuju upisivanje odnosno kliktanje na element stranice.

4.3 Integracija unutar NodeJS okruženja

Svi navedeni dijelovi programskog paketa razvijeni su unutar NodeJS (Joyent and other Node contributors, 2018) programskog okruženja. NodeJS pruža lakši pristup i integraciju vanjskih paketa, upravljanje dependency-ima i pakiranje te izvršavanje unutar komandne linije i izvan konteksta preglednika. Isto tako podržava sve najnovije ES2017 standarde programskog jezika JavaScript kao što su await i async (ECMA, 2017) . Vanjski paketi korišteni u sklopu implementacije su:

1. **Body parser** - koristi se za prikaz odgovora API web servera
2. **EveJS** - NodeJS platforma za više agentne sustave (B.V., 2018)
3. **Express** - Web framework, korišten za API web server (StrongLoop and other expressjs.com contributors, 2018)
4. **Puppeteer** - DevTools protocol API (Google, 2018c)
5. **Request** - korišteno za posluživanje i dohvaćanje sadržaja preko HTTP protokola

4.4 Implementacija

Za bazne funkcionalnosti TwitterAgent-a koriste se metode iz roditeljske klase eve.Agent koja je dio EveJS paketa. Roditeljska klasa pruža bazne metode za komunikaciju agenata Agent.send i Agent.receive. Prva metoda šalje poruku agentu dok je druga zapravo callback metoda koja se izvršava kada agent primi poruku od drugog agenta. Ponašanje samog agenta nije definirano unutar TwitterAgent klase nego isti sadrži referencu na Twitter API i sadrži metode koje koriste taj API za odrađivanje zadataka u sklopu Twitter web aplikacije. Važno je da ovdje ne postoji poveznica između agenta instance preglednika koju koristi nego oni komuniciraju preko Twitter API sučelja. Sučelje može egzistirati i bez agenta te je primjer takve implementacije prikazan u server.js (Barić, 2018) datoteci gdje je isto implementirano unutar web servera. Ukoliko TwitterAgent nema definirano ponašanje kako onda može izvršavati zadatke i simulirati ponašanje pravog korisnika? Po uzoru na eve.Agent klasu svaki konkretni agent koji nasljeđuje TwitterAgent klasu sam definira svoje ponašanje implementirajuči metode TwitterAgent.runBehaviour i TwitterAgent.onEvent.

```
TwitterAgent.prototype.onEvent = async function (parsedMessage) {
    // implement onEvent behaviour
};

TwitterAgent.prototype.runBehavior = async function () {
    // implement behaviour
};
```

Prva metoda izvršava ponašanje agenta u kontekstu Twitter web aplikacije dok druga metoda sluša događaje koje šalju drugi agenti i reagira na njih. Događaji se prenose preko poruka TwitterAgent.notify metodom. Ova metoda zapravo koristi Agent.send za slanje poruka, ali umjesto obične tekstualne poruke šalje JSON poruku koju onda agent metodom Agent.receive proba pročitati i ukoliko je JSON ispravan reagira sa odgovarajućim ponašanjem kroz metodu TwitterAgent.onEvent. JSON definicija Event poruke je sljedeća:

```
{
 "event": EventEnum,
 "username": String,
 "tweetId": Number,
}
```

Unutar poruke atribut event je obavezan parametar. EventEnum može imati jednu od sljedećih vrijednosti:

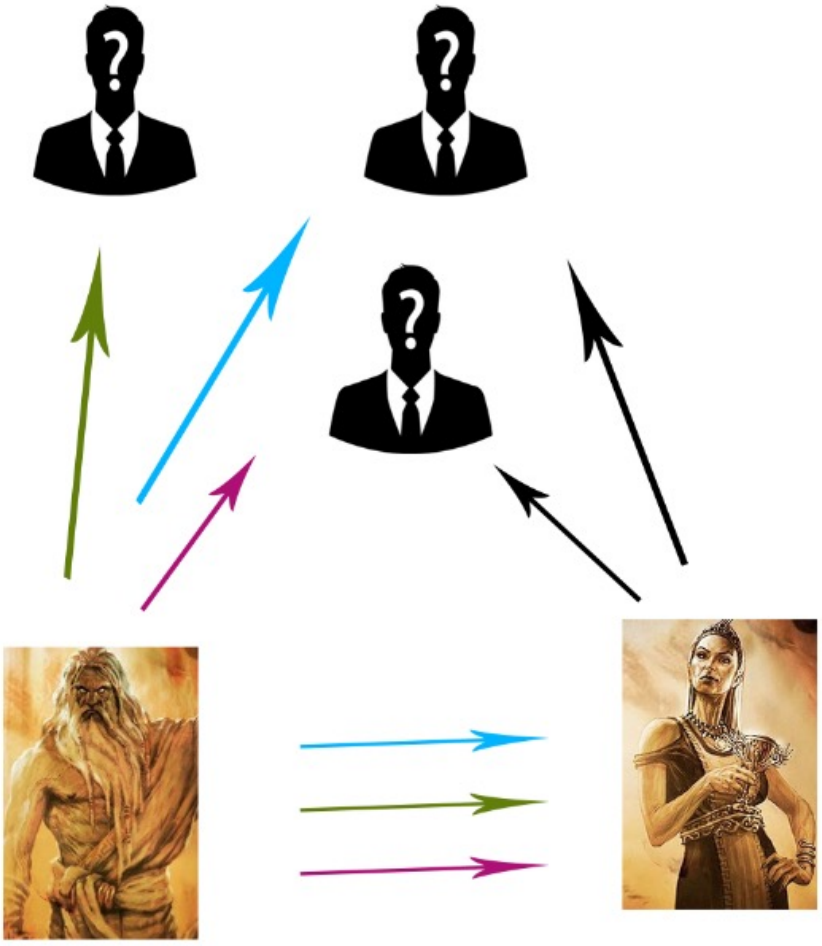
1. **hello** - pozdravna poruka agenta
2. **followed** - agent X je počeo pratiti agenta Y
3. **unfollowed** - agent X je prestao pratiti agenta Y
4. **liked** - agent X je favorizirao objavu agenta Y
5. **tweeted** - agent X objavio tweet
6. **retweeted** - agent X je retweetao objavu agenta Y

Atributi username i tweetId su opcionalni parametri i mogu oba biti pristutni ili samo jedan od njih (ovisno i tipu događaja) Na primjer kod "followed" događaja Event poruka će sadržavati atribut username koji označava korisničko ime korisnika kojega je agent X počeo pratiti. Za razliku od toga kod "retweeted" događaja će sadržavati i username agenta koji je objavio tweet i identifikator tog novog tweeta koji je objavljen. Ukoliko poruka nije valjane definicije TwitterAgent ignorira takvu poruku.

Poglavlje 5

Prikaz i opis simulacije

Za simulaciju i praktični primjer u sklopu ovoga rada izražena su dva agenta od koji svaki ima jednu ulogu. Agent Zeus ima implementirano ponašanje stvaranja/pružanja podataka dok agent Hera aktivno čeka na događaje i reagira ovisno o njihovom tipu.



Slika 5.1: Interakcije agenata sa korisnicima (Studio, 2018)

Slika 4.1 prikazuje način interakcije agenata Zeus i Hera sa korisnicima društvene mreže Twitter.

Različita boja strelice prikazuje drugačiji dip događaja. Nakon svake radnje Zues obaviještava Heru preko poruka o događaja te onda Hera ukoliko prepoznaje primljeni događaja izvršava definiranu radnju. Kao što možemo vidjet na slici 4.1 Hera u tom primjeru reagira na dva od tri događaja jer za jedan nema definirano ponašanje što simulira situaciju gdje pravni korisnici na neke informacije reagiraju pozitivno, na neke negativno, a na neke uopće ne reagiraju.

5.1 Opis simulacije

Cilj simulacije je prikazati mogućnosti koje pruža okruženje razvijeno u sklopu ovog seminarskog rada. U svrhu simulacije su kreirana dva Twitter računa po nazivima **@zeus_foi** i **@hera_foi**. Računi će nakon završetka projekta biti zatvoreni kako ne bi narušavali uvijete korištenja društvene mreže Twitter. Praćenje rada agenata tijekom simulacije može se vršiti preko ispisa log-a u terminalu nakon pokretanja ili ukoliko se isključi "headless" način rada onda će se nakon pokretanja simulacije pokrenuti dvije instance Chromium preglednika i unutar svakoga će biti moguće vidjeti radnje koje agenti automatizirano izvršavaju.

```
...

await puppeteer.launch({headless: false, timeout: 0}).then(
    async browser => {
        let zeus = new Zeus('USERNAME'}, 'PASSWORD');

        let page = await browser.newPage();
        await page.emulate(DevicesProfiles.desktop);

        zeus.twitter = await new Twitter(page);

        await zeus.runBehavior();
    }
);

...
```

Instanca preglednika se mora pokrenuti unutar asinkrone funkcije koja onda nakon otvaranja i učitavanja preglednika vraća njegovu instancu kroz callback. Instanca preglednika ima metode kao što je Browser.newPage koja omogućava otvaranje novih stranica, zatvaranje stranica, sesija i slično. Važno je da se sve metode koje se tiču instance preglednika pozivaju unutar callback metode jer unutar nje je sigurno da je preglednik još uvijek otvoren. Argument headless označava da li želimo izvršavanje preglednika bez grafičkog sučelja ili normalno sa sučeljem.

5.2 Agenti učesnici

Kako sam već prije rekao u simulaciji učestvuju dva agenta pod nazivima Zeus i Hera. Prije programiranja njihovih ponašanja za simulaciju odradio sam nekoliko testnih ponašanja sa svrhom prikupljanja podataka i upoznavanja oba agenta sa nekim stvarnim korisnicima na društvenoj mreži Twitter. Na

primjer krenuo sam sa time da Agenti prate poznate profile npr. @google, @apple, @bethesha, @santamonica i slično, a onda u nekoliko navrata pustio da agenti prolaze kroz njihove objave, followere i interese. Nakon što su Agenti popunili svoju mrežu ovisno o njihovom trenutnom stanju definirao sam svakome ponašanja koja ću opisati u daljnjem tekstu.

5.2.1 Agent Zeus

Agentovo ponašanje se temelji na iteraciji kroz interese (korisnike koje prati) i proširivanje svoje mreže praćenjem interesa tih korisnika i tako dok god ne dođe do kraja svoje liste interesa. Agent samo na početku ponašanja pregledava svoju listu interesa i dodatne korisnike koje počne pratiti tijekom ponašanja ne dodaje u listu za to ponašanje nego će te korisnike pregledati tek u ponovnom pokretanju svoga ponašanja tj. novoj simulaciji. Tijekom svake iteracije agent uz praćenje novih korisnika metodom slučajnosti određuje da li će odraditi neki dodatnu radnju. Radnje su definirane na sljedeći način:

1. **Follow/Unfollow** - prosječno u 3 od 10 slučajeva agent će prestati pratiti trenutnog korisnika iz svojih interesa
2. **Tweet** - proječno u 1 od 20 slučajeva agent će objaviti novi tweet na svom profilu.
3. **Retweet** - prsoječno u 5 od 10 slučajeva agent će retweetati objave trenutnog korisnika iz svojih interesa
4. **Like** - u 2 od 10 slučajeva agent će favorizirati zadnji tweet trenutnog korisnika iz svojih intersa

Ponašanje agenta Zeus je profilirano za korisnika koji je vrlo aktivan na društvenim mrežama i svakodnevno tweet-a, prati nove korisnike i otkriva nove informacije. Korisnik osim toga isto tako stvara veliki broj sadržaja koji prolazi društvenom mrežom, a upravo drugi agent u simulaciji Hera je tu da prikaže koliko kreiranje novih informacija od strane interesa korisnika utječe na korisnika i njegovo ponašanje na društvenoj mreži.

```
Zeus.prototype.runBehavior = async function () {
    await this.login();

    await this.notify("foi_hera", {"event": "hello", "username":
    this.username});

    for (let interest of await this.interests()) {
        await this.followInterests(interest);

        if (Math.random() > 0.7) {
            await this.unfollow(interest);

            try {
                await this.notify("foi_hera", {"event": "unfollowed",
                "username": interest});
            } catch (e) {
                console.log("Agent not found");
            }
```

```
    } else {
        await this.follow((interest));

        try {
            await this.notify("foi_hera", {"event": "followed",
            "username": interest});
        } catch (e) {
            console.log("Agent not found");
        }
    }

    if (Math.random() > 0.95) {
        await this.tweet(await Helpers.getQuote());

        try {
            await this.notify("foi_hera", {"event": "tweeted",
            "username": this.username});
        } catch (e) {
            console.log("Agent not found");
        }
    }

    if (Math.random() > 0.5) {
        let tweetId = await this.retweetLastTweet(interest);

        try {
            await this.notify("foi_hera", {"event": "retweeted",
            "username": interest, "tweetId": tweetId});
        } catch (e) {
            console.log("Agent not found");
        }
    }

    if (Math.random() > 0.8) {
        let tweetId = await this.likeLastTweet(interest);

        try {
            await this.notify("foi_hera", {"event": "liked",
            "username": interest, "tweetId": tweetId});
        } catch (e) {
            console.log("Agent not found");
        }
    }
```

```
        await Helpers.sleep(5000)
    }

    await this.logout();
};
```

Kod prikazuje definirano ponašanje agenta Zeus. Možemo vidjeti da nakon svake radnje izvršava metodu notify i obavještava u ovom slučaju drugog agenta o svojoj radnji. Naravno u slučaju više agenata i kompleksnijoj simulaciji ovdje bi se obavijest mogla slati svim agentima ili samo nekima ovisno o situaciji. Kako agenti tijekom izvršavanja radnji preko DevTools protokola i Puppeteer biblioteke praktički navigiraju stranicom, a svaka metoda TwitterAPI-a je asinkrona koristiti se await kako bi agent sa čeka izvršenje asinkrone funkcije i tek onda nastavio sa daljnjim radom. Dodatno prije svake nove iteracije agent pričeka 5 sekundi kako bi se nakon izvršavanja radnji. Ovo niti nije nužno pošto Puppeteer biblioteka sama ima integrirano prepoznavanje da li web aplikacija ili stranica u pozadini odrađuju asinkrone pozive ili ne. Odmak je postavljen kako se ne bi poslalo možda previše requestova i twitter ne bi prepoznao ponašanje agenta kao spam. Ovakvo ponašanje ipak više sliči korisniku koji sa razmacima odrađuje svoje akcije.

5.2.2 Agent Hera

Hera je agent čije se ponašanje temelji na odgovaranju na radne drugih korisnika u ovom slučaju agenta Zeus. Hera prepoznaje sljedeće događaje te na njih reagira:

1. **hello**
2. **followed**
3. **tweeted**
4. **liked**
5. **retweeted**

Hera ne mora nužno na događaje reagirati kreiranjem iste radnje kao one koja je uzrokovala događaj. Na primjer kada Zeus počne pratiti nekog novog korisnika, Hera će favorizirati zadnji tweet istog korisnika. Kada Zeus favorizira neki tweet Hera će u prosjeku 4 od 10 puta isto favorizirati taj tweet dok će u ostalim slučajevima retweetati taj isti tweet na svom profilu. Isto tako će svaki novi Zeusov tweet favorizirati ili retweetati na svom profilu. Kada Zeus retweeta objavu nekog korisnika Hera ga tada (ukoliko ga prati) prestaje pratiti. Herino ponašanje se temelji na profilu korisnika koji je vrlo zainteresiran ili je fan profila korisnika kojeg prati. Kao rezultat toga Hera će prikupljati i pratiti sve aktivnosti i sadržaje koje objavluje Zeus. Isto tako kako je specifično zainteresirana za sadržaje koje objavljuje Zeus ignorirati će sadržaje drugih korisnika.

```
Hera.prototype.onEvent = async function (parsedMessage) {
    if (this.twitter.data.session === null) {
        await this.login();
    }

    switch (parsedMessage.event) {
```

```
        case "hello":
            await this.follow(parsedMessage.username);
            break;
        case "followed":
            await this.likeLastTweet(parsedMessage.username);
            break;
        case "unfollowed":
            await this.unfollow(parsedMessage.username);
            break;
        case "liked":
            if (Math.random() > 0.6) {
                await this.like(parsedMessage.username,
                parsedMessage.tweetId);
            } else {
                await this.retweet(parsedMessage.username,
                parsedMessage.tweetId);
            }
            break;
        case "tweeted":
            if (Math.random() > 0.6) {
                await this.likeLastTweet(parsedMessage.username);
            } else {
                await this.retweetLastTweet(parsedMessage.username);
            }
            break;
        case "retweeted":
            await this.unfollow(parsedMessage.username)
    }

    Helpers.sleep(5000);
};
```

Kod pokazuje definirano Herino ponašanje. Treba napomenuti da se ovo ponašanje izvršiva samo kada Hera primi valjanu poruku prema JSON definiciji Event poruke. Spomenuta metoda receive koja čita dolazne poruke izgleda ovako:

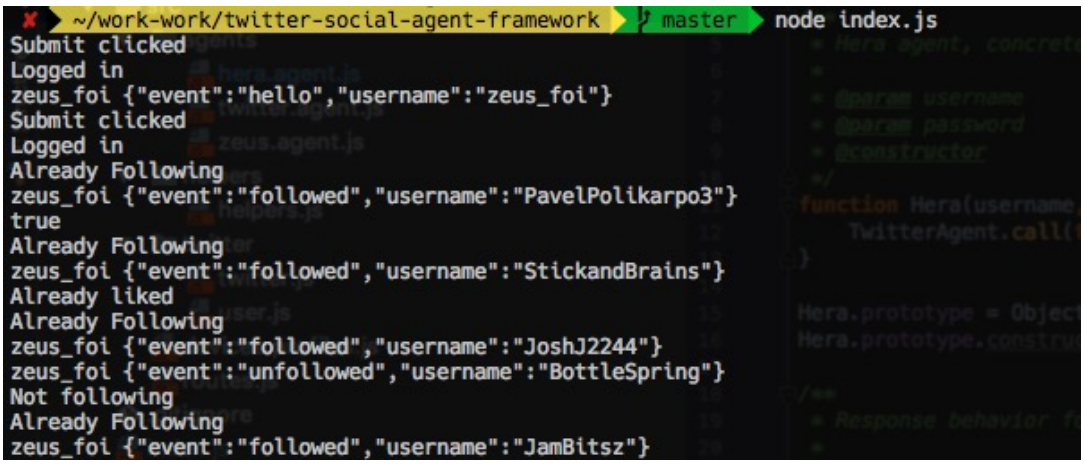
```
TwitterAgent.prototype.receive = function (from, message) {
    console.log(from, message);

    try {
        let parsedMessage = JSON.parse(message);

        this.onEvent(parsedMessage)
    } catch (e) {
        console.log(from, message)
    }
```

```
};
```

Konkretni agent ne mora imati implementirano onEvent ponašanje te ako ga nema, a primi poruku jednostavno se neće ništa dogoditi.

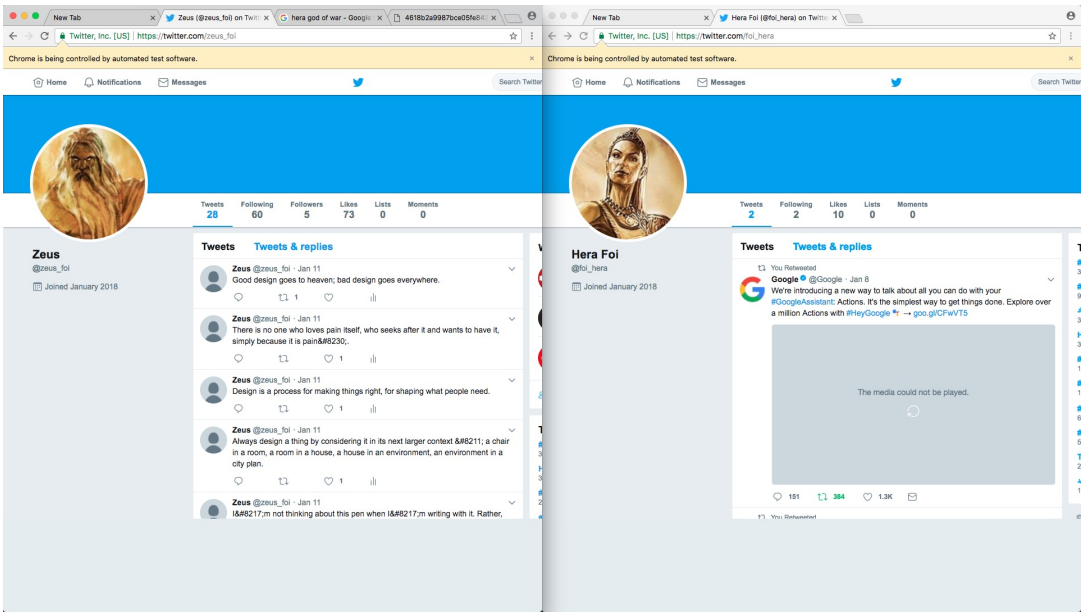
5.3 Prikaz rada simulacije

Upute za pokretanje simulacije nalaze se u README.md (Barić, 2018) datoteci na repozitoriju projekta. Log kod pokretanja u headless načinu rada prikazan je na slici 5.2

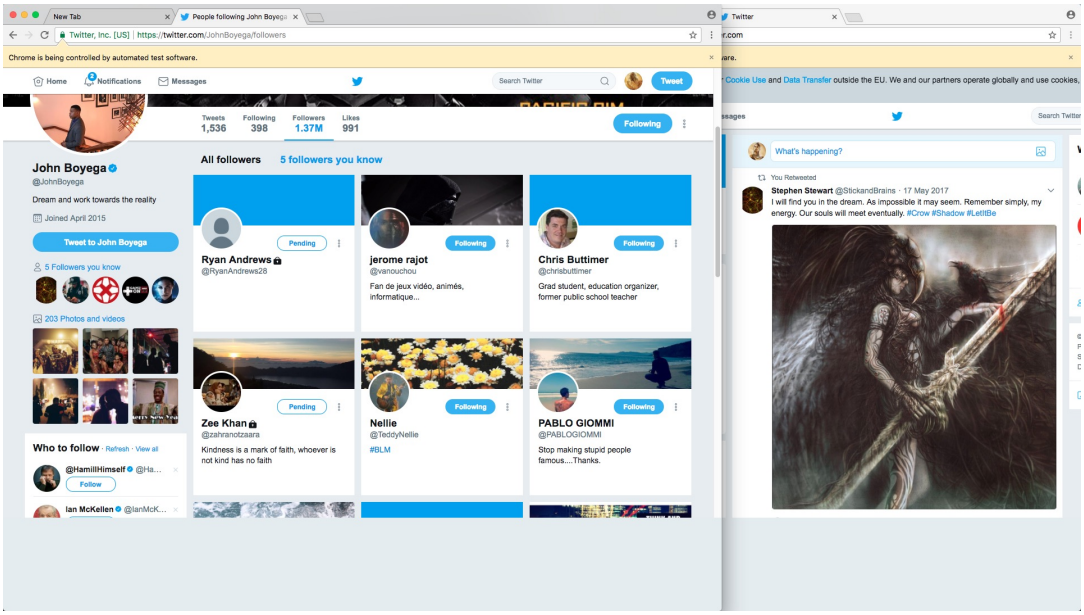


Slika 5.2: Primjer interakcija i ponašanja korisnika

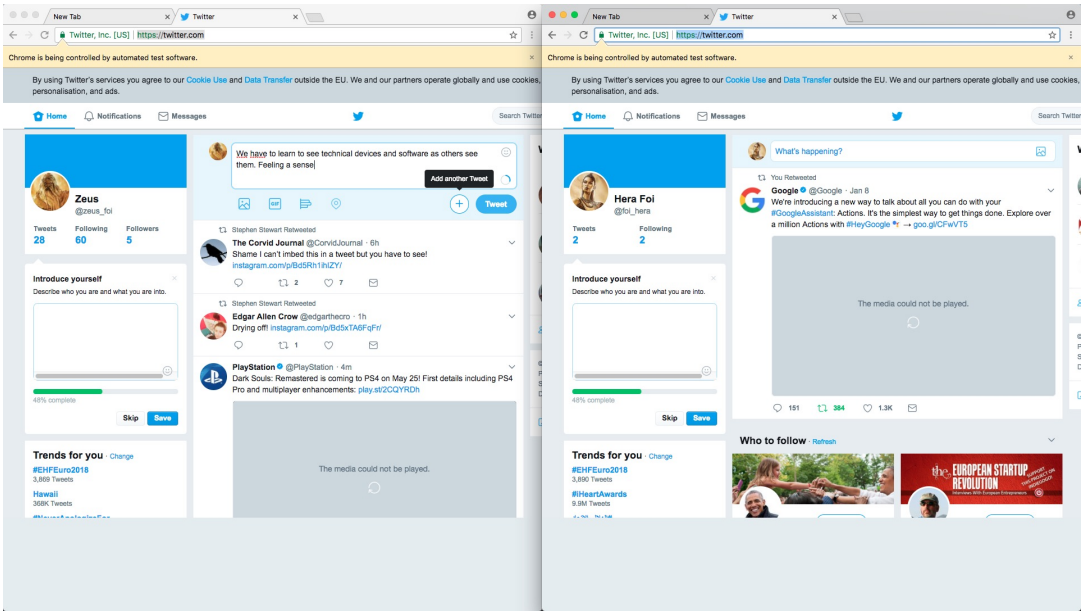
Ako isključimo headless način dobivamo više vizualni prikaz radnji agenata. U sljedećim slikama ekran je prepolovljen na dva dijela gdje je lijevi dio Zeusova instanca prelgednika, a lijevi dio Herina.



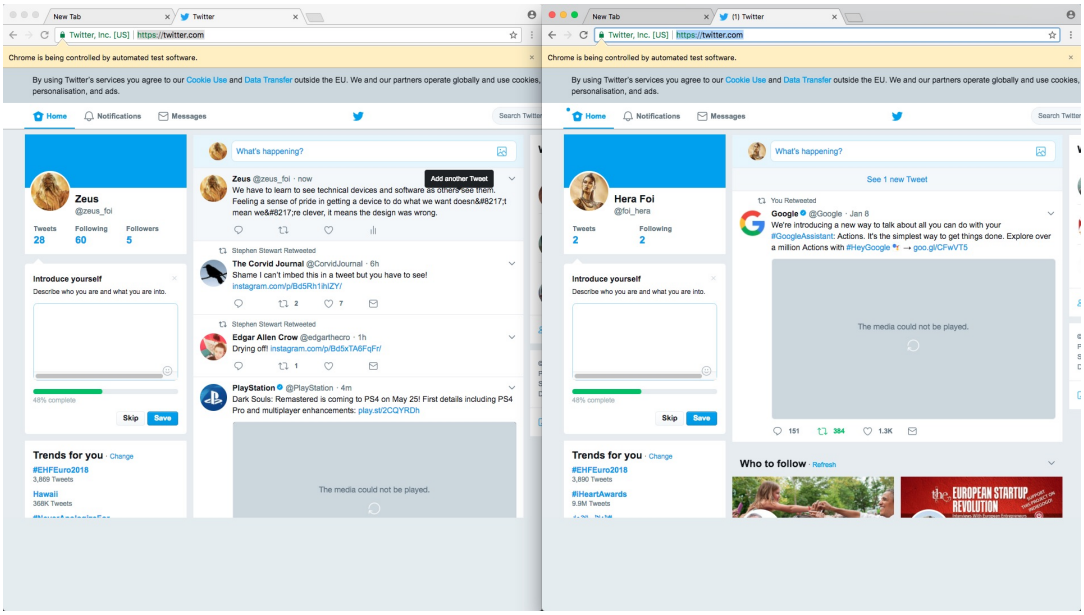
Slika 5.3: Sadržaj generiran od strane agenata na njihovim profilima



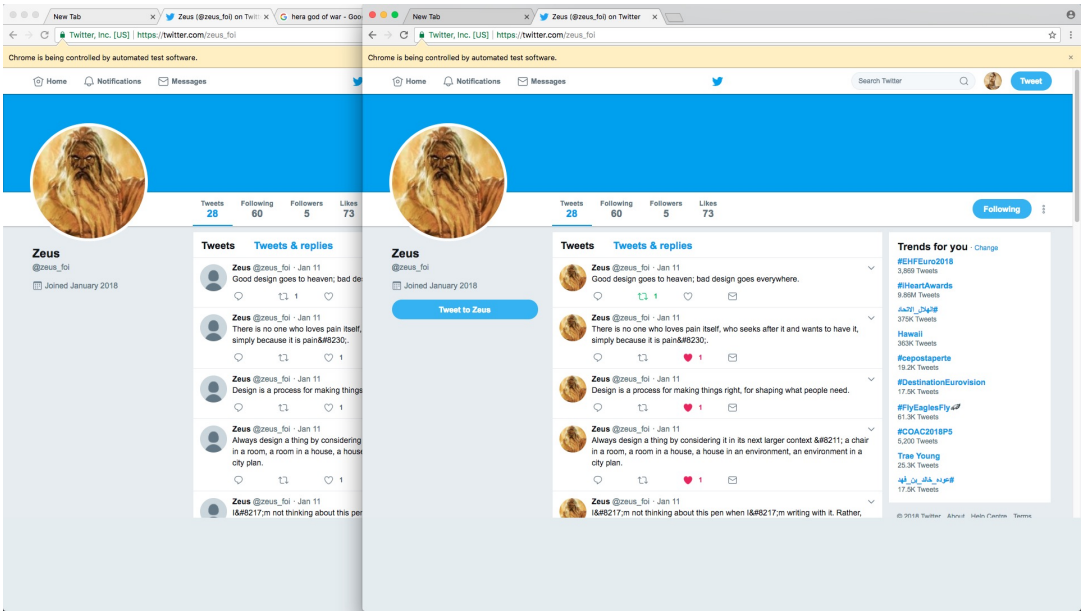
Slika 5.4: Zeus prati interese korisnika



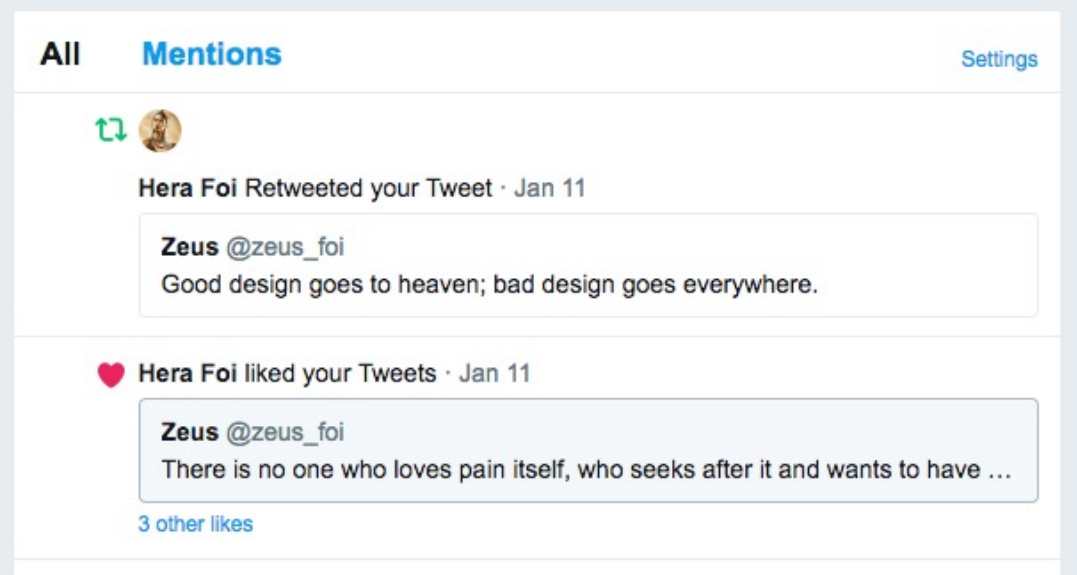
Slika 5.5: Zeus upisuje svoj novi tweet.



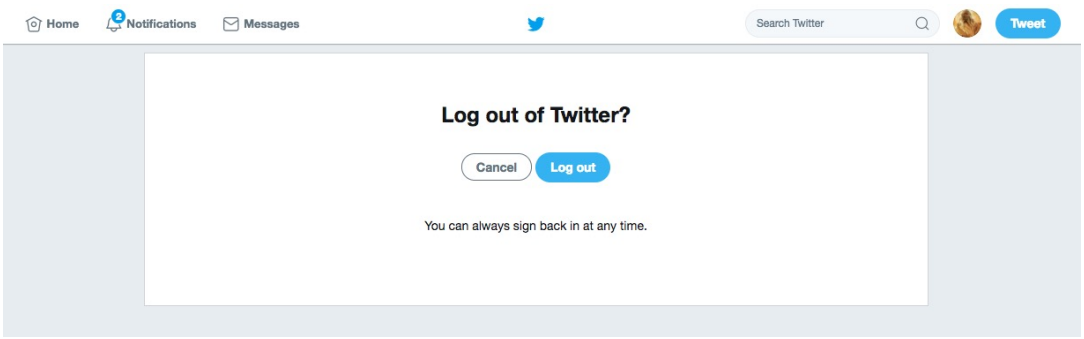
Slika 5.6: Objavljeni tweet i ekran desno prikazuje da hera odma ima u svojoj instanci preglednika notifikaciju za novi tweet



Slika 5.7: Hera favorizira i retweeta Zeusove objave



Slika 5.8: Zeus dobiva notifikacije o akcijama koje je obavila Hera nad njegovim sadržajem



Slika 5.9: Kada završe svoja ponašanja agenti se odjave iz web aplikacije

Poglavlje 6

Zaključak

Agenti u sustavu društvenih mreža mogu imati više uloga, svrha i namjena. 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 ukoliko je API razvijen po istim smjernicama Agenti će isto tako raditi sa drugim društvenim mrežama. Slabost ovakvog tipa agenata 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 agenta temeljem podataka koje prikuplja kako bi samostalno mijenjao svoje ponašanje ovisno o promjenama stanja mreže. Dakako ovo bi zahtjevalo implementaciju neke vrste Machine learning algoritma, 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 agente u dinamičnom okruženju društvenih mreža. Treba napomenuti da je tehnologija DevTools protokola i nekih drugih metoda korištenih unutar ovog rada još uvijek u Beta fazama ili 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 developera koji koriste iste.

Bibliografija

Adelinde M. Uhrmacher, D.W., 2009. Multi-Agent Systems: Simulation and Applications. CRC Press.

Barić, A., 2018. *Twitter social agent framework*. Dostupno na `https://github.com/capJavert/twitter-social-agent-framework`.

B.V., A., 2018. *evejs*. Dostupno na `https://github.com/enmasseio/evejs`.

Chi-In Wong, Kin-Yeung Wong, K.W.N.W.F.K.H.Y., 2014. *Design of a Crawler for Online Social Networks Analysis*. Tech. rep., City University of Hong Kong.

ECMA, 2017. *ECMAScript® 2017 Language Specification*. Dostupno na `https://www.ecma-international.org/publications/standards/Ecma-262.htm`.

Google, 2018a. *Chrome DevTools Protocol API Docs*. Dostupno na `https://chromedevtools.github.io/devtools-protocol/`.

Google, 2018b. *Chromium*. Dostupno na `https://www.chromium.org/Home`.

Google, 2018c. *Puppeteer*. Dostupno na `https://github.com/GoogleChrome/puppeteer`.

Joyent, I. and other Node contributors, 2018. *About Node.js*. Dostupno na `https://nodejs.org/en/about/`.

StrongLoop, I. and other expressjs.com contributors, 2018. *Express4.16.1 Fast, unopinionated, minimalist web framework for Node.js*. Dostupno na `https://expressjs.com/`.

Studio, S.S.M., 2018. *Explore God Of War, Book Series and more!* Dostupno na `https://www.pinterest.com/pin/428123508312043847/`.

Twitter, I., 2018. *API reference index*. Dostupno na `https://developer.twitter.com/en/docs/api-reference-index`.

Wikipedia, 2017. *Twitter*. Dostupno na `https://hr.wikipedia.org/wiki/Twitter`.

Wikipedia, 2018. *Web crawler*. Dostupno na `https://en.wikipedia.org/wiki/Web_crawler`.