Twitter zid

Vrljić, Ivan

Master's thesis / Diplomski rad

2016

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:922842

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-19**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

TWITTER ZID

Diplomski rad

Ivan Vrljić

Osijek, 2016.

SADRŽAJ

1. UVOD ... 1

1.1 Zadatak diplomskog rada ... 1

2. KORIŠTENE TEHNOLOGIJE ... 2

2.1 Tehnologije na poslužiteljskoj strani ... 2

2.1.1 RESTful ... 2

2.1.2 Prednosti ASP.NET Web API-a ... 4

2.2 Tehnologije na klijentskoj strani ... 4

2.2.1 Ukratko o SCSS-u ... 4

2.2.2 TypeScript ... 5

2.2.3 Knockout.js ... 5

3. TWITTER ZID ... 6

3.1 Struktura ... 7

3.2 Stvaranje i funkcionalnost SPA ... 7

3.2.1 Model mainVM ... 8

3.2.2 Model loginVM ... 9

3.2.3 Model twitterVM ... 9

3.2.4 Modeli profileSearchVM i tagSearchVM ... 13

3.2.5 Modeli myProfileVM i othersVM ... 14

3.3 Implementacija poslužiteljske strane ... 15

4. TESTIRANJE MOGUĆNOSTI APLIKACIJE ... 22

4.1 Prijava korisnika u aplikaciju ... 22

4.2 Početna stranica ... 24

4.3 Korisnikov profil ... 27

4.4 Profili ostalih osoba ... 28

4.5 Odjava iz aplikacije ... 29

5. ZAKLJUČAK ... 31

LITERATURA ... 32

SAŽETAK ... 34

ABSTRACT ... 35

ŽIVOTOPIS ... 36

PRILOZI ... 37

1. UVOD

Kroz ovaj rad opisat će se postupak izrade aplikacije za Twitter društvenu mrežu. Objasnit će se način implementacije u obliku jednostranične aplikacije (engl. *Single Page Application - SPA*) za koje vrijedi da se njen sadržaj učitava jednom i dinamički obrađuje na klijentskoj strani, a poslužiteljska strana služi za dohvat podataka. Izrađena aplikacija je u potpunosti responzivna i prilagođava sve svim veličinama ekrana.

Pristup stvarnim korisničkim podatcima obavlja se s Twitterovog API sučelja koje pruža mogućnost pristupa velikoj količini informacija koje se nalaze na toj društvenoj mreži. Kroz rad opisan je način pristupa podatcima koji zahtijeva autorizaciju putem tokena. U poglavlju o obradi podataka na poslužitelju detaljno je definiran način pribavljanja i enkripcije istih.

Za izradu aplikacije korištene su različite tehnologije, od onih vezanih za poslužiteljsku stranu (C#, ASP.NET Web API...), do onih vezanih za klijentsku stranu (JavaScript, CSS, HTML...). Neke od njih, one važnije u implementaciji, detaljnije su opisane kroz ovaj rad.

1.1 Zadatak diplomskog rada

Izraditi web aplikaciju koja će podatke prikupljene putem Twitter API sučelja korisniku prikazivati na vizualno privlačan i zanimljiv način.

Sama aplikacija može se podijeliti u tri dijela: naslovni dio, dio s profilom i dio s pregledom profila ostalih osoba. Naslovni dio sadrži tri paralelne liste trenutno najpopularnijih tema na Twitteru. Korisnik ima mogućnost potražiti oznake koje ga zanimaju i pratiti tweetove odabrane teme. Dio s profilom omogućava korisniku uvid u prijatelje, pratitelje, prijedloge osoba za praćenje te blokirane osobe. Ovaj dio korisniku također pruža prikaz vlastitih i najdražih tweetova. Profili ostalih osoba omogućavaju uvid u njihove tweetove, prijatelje, promjenu statusa praćenja i slično. Za svaki tweet dane su mogućnosti za odgovor na tweet, ponovno dijeljenje, označavanje omiljenim tweetom itd. Aplikacija korisniku omogućava objavu tweetova sa i bez medijskog sadržaja (slika).

2. KORIŠTENE TEHNOLOGIJE

Kroz posljednje desetljeće došlo je do naglog razvoja mrežnih tehnologija, ali i do promjene načina pristupa i primjene internetskih aplikacija. U početku su se web aplikacije svodile na to da se na poslužiteljskoj strani stvori čitavi HTML (engl. *HyperText Markup Language*) koji je popunjen s podatcima prikupljenim iz baze podataka ili nekog drugog izvora te se kao takav slao klijentu. U posljednje vrijeme u značajnom su porastu aplikacije koje se izvode na klijentskoj strani unutar korisnikovog preglednika. Najčešća izvedba ovakvih aplikacija su jednostranične aplikacije (SPA). SPA su web aplikacije koje učitaju jedan HTML i dalje ga osvježavaju i nadopunjuju kako korisnik vrši interakciju s aplikacijom [1]. Kod nekih implementacija sav potrebni HTML, CSS (engl. *Cascading Style Sheets*) i JavaScript učitava se odmah, a kod drugih se preuzima dio sadržaja i naknadno odlazi po dodatni sadržaj. Ovako izvedene aplikacije podatke koji se ubacuju u HTML zaprimaju u obliku tekstualnih poruka definiranog formata. Uobičajeni formati prijenosa su XML (engl. *EXtensible Markup Language*) i JSON (engl. *JavaScript Object Notation*). Navedeni formati imaju veliku zastupljenost budući da su pogodni za mobilne uređaje i tablete koji su po pitanju weba postale dominantnije pristupne točke u odnosu na osobna računala. Ovi uređaji obično komuniciraju putem protokola HTTP (engl. *HyperText Transfer Protocol*) te izvršavaju prijenos manje količine podataka baziranih na tekstu odnosno podatke oblikovane u JSON i XML formate.

2.1 Tehnologije na poslužiteljskoj strani

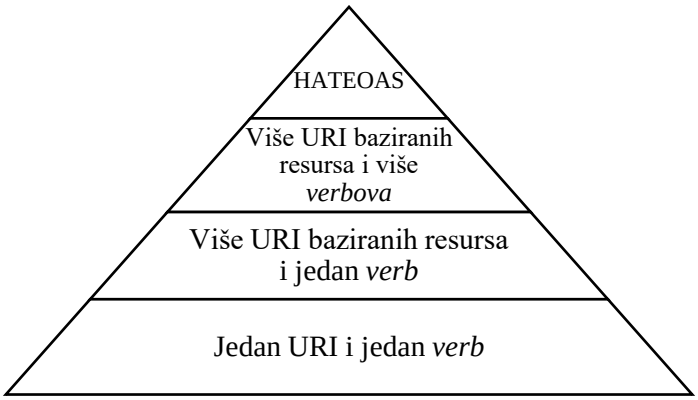
Pozadina projekta je u potpunosti razvijena u Microsoftovom C# (engl. *C Sharp*) programskom jeziku. C# je objektno orijentiran programski jezik opće namjene, a programi izrađeni u njemu izvode se unutar .NET okruženja. Za razvoj aplikacije je korišteno ASP.NET Web API okruženje kojemu je glavni fokus HTTP. Ovo okruženje zasniva se na REST (engl. *REpresentational State Transfer*) arhitekturi koja je ostvarena preko posebnog URL (eng. *Uniform Resource Locator*) usmjeravanja. Rute su izvedene na način da zapravo ne predstavljaju fizičku lokaciju dokumenta već su to RESTful rute koje navode na metode koje se nalaze u upravljačima (engl. *Controllers*) Web API projekta.

2.1.1 RESTful

REST je arhitekturalni stil za dizajniranje mrežnih aplikacija. Čitava ideja ove arhitekture je da se kompleksni mehanizmi za povezivanje između uređaja (npr. RPC, SOAP) zamijene sa

jednostavnim HTTP-om. RESTful aplikacije koriste HTTP zahtjeve za sve četiri CRUD (stvaranje – engl. *Create*, čitanje – engl. *Read*, ažuriranje – engl. *Update*, brisanje – engl. *Delete*) operacije. Služi kao „lagana“ zamjena za složene mehanizme sa svim bitnim odlikama [2].

Krajem 2008. godine Leonard Richardson kreirao je model za REST koji je stvoren kako bi povećao učinkovitost i performanse, a prema njemu to predstavlja osnovu RESTful usluge.



Sl. 2.1. Richardsonov REST model

Na najnižoj razini modela nalazi se sučelje koje je karakterizirano time da jedan URI podržava samo jednu metodu i jedan *verb* [3, str. 11]. Za svaku CRUD metodu koja je implementirana URI izgleda jednako pa korisnik nema točan uvid u resurs. Druga razina ostvaruje REST ideju o centraliziranosti na resurse. URI više nije jedinstven svim metodama jer se dodaju parametri kao što je npr. ID. Na trećoj razini uvodi se mogućnost više HTTP *verbova*, a to su: GET, PUT, POST i DELETE. Korištenjem ovih *verbova* na istome URI-u možemo imati više različitih metoda. Ukoliko je primjerice URI oblika api/Tweets, automatski se zna da se metode odnose na grupu podataka, a ukoliko je poziv oblika api/Tweets/1, jasno je da se odnosi na točno određeni podatak. Sve CRUD metode pozivaju se preko istog URI-a (za grupu ili za pojedinačni element), a koju metodu treba izvršiti odlučuje se s predanim HTTP *verbom*. GET se koristi za čitanje, PUT za ažuriranje, POST za kreiranje novog elementa te DELETE za brisanje. Za GET, PUT i DELETE vrijedi da koliko god puta bili pozivani, uvijek moraju vratiti jednak rezultat bez pogreške [3, str. 12].

POST, za razliku od prošla tri *verba*, svakim pozivom kreira novu instancu koju vraća natrag po uspješnom kreiranju. Iz tog razloga nakon svakog poziva ima drukčiji odgovor. Najviša razina, HATEOAS, definira da se unutar svakog odgovora (XML, JSON objekta) ugrade dodatne veze koje označavaju otkud je koja informacija došla i na koji način korisnik može dalje vršiti interakciju s aplikacijom.

2.1.2 Prednosti ASP.NET Web API-a

Kao glavna prednost okruženja mogu se istaknuti CRUD akcije koje se automatski vežu na API upravljače (engl. *Contoller*) prema imenu. Primjerice, URI oblik *api/Data/tweets* je automatski vezan za metodu *tweets* na upravljaču imena *Data*. Temeljem predanog *verba* (GET, POST…) i broja predanih parametara, upravljač zna točnu metodu koju je korisnik zatražio.

Druga prednost je u tome što se svaki objekt ili pak primitivni podatkovni oblik (string, int…) automatski pretvara u oblik koji je zatražen u zaglavlju zahtjeva (XML ili JSON). Ovime se smanjuje kod jer se ne mora uvoditi dodatna obrada podataka prilikom slanja odgovora.

Među prednosti može se uvrstiti i mogućnost ograničavanja predanih parametara s definiranjem najvećeg broja znamenki, najveće vrijednosti, regularnog izraza…

Web API posjeduje mogućnost omogućavanja CORS-a (engl. *Cross Origin Resource Sharing*) koji je po zadanim postavkama zabranjen [4]. Najčešće ova opcija nije potrebna, ali za API-je koje koristi veliki broj uređaja može biti vrlo korisna.

Posljednja bitna stavka su IHttpActionResult tipovi koji služe za enkapsulaciju podataka koji se šalju klijentu. Moguće je stvoriti odgovore s HTTP statusima kao što su 200 – OK, 404 – Not Found, 500 – Server Issue itd.

2.2 Tehnologije na klijentskoj strani

Na klijentskoj strani nalazi se raznovrsna tehnologija koja se na kraju svodi na tehnologiju koju internet preglednici podržavaju: HTML, CSS i JavaScript. Za pisanje CSS koda korištena je SCSS sintaksa koja donosi brojne prednosti u odnosu na standardni CSS. JavaScript kod pisan je TypeScriptom. HTML je standardan, ali sadržava dodatne atribute koji će služiti za vezanje s vrijednostima podataka u JavaScriptu (Knockout.js veze).

2.2.1 Ukratko o SCSS-u

SCSS, odnosno Sassy CSS, nadskup je sintakse nad standardnim CSS-om. Prilikom prevođenja, SCSS kod pretvara se u običan CSS koji preglednici mogu razumjeti. Najveća prednost ove sintakse nad standardnom jest mogućnost deklariranja varijabli koje se mogu ponavljati tokom koda. Nad varijablama se mogu vršiti osnovne matematičke funkcije za računanje vrijednosti. SCSS uvodi pojam *mixina* koji predstavlja dijelove koda nalik funkcijama

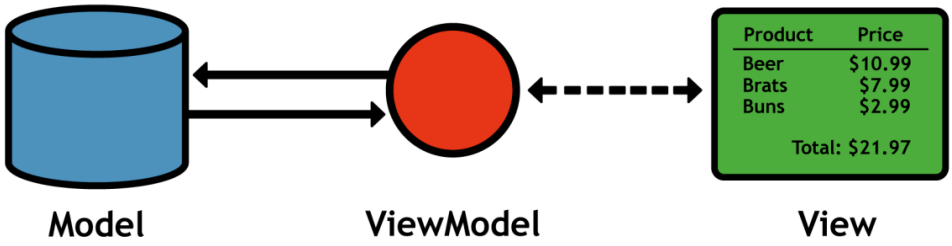
koje primaju parametre i zaprimljene parametre razvrstavaju po CSS svojstvima [5]. *Mixini* su korisni u slučajevima kada se dio koda često ponavlja, primjerice za definiranje koda za tranzicije i animacije. Još jedna prednost je gniježđenje selektora kojim se postiže slojeviti kod kojeg je lakše pratiti i održavati nego li dugačke selektore standardne sintakse CSS-a.

2.2.2 TypeScript

TypeScript je nadskup JavaScripta koji kao primarne prednosti donosi klase, sučelja (engl. *interface*) i statičke tipove podataka. Ideja je bila napraviti JavaScript kod što više nalik onome koji se koristi u C#. Razvija ga Microsoft kao projekt otvorenog koda s GPL licencom. Prije prevođenja .ts dokumenata u .js potrebno je definirati koji će ECMAScript standard biti referenca za izlazni dokument. JavaScript ima dinamičke tipove podataka, što znači da se tip podatka određuje tek u trenutku pridjeljivanja vrijednosti. Dinamički tipovi čest su uzrok pogrešaka u kodu zbog pretpostavki programera da će varijabla poprimiti željeni tip. TypeScript omogućava programeru da definira statički tip varijable. Ovime se osigurava da se, primjerice, unutar varijable tipa *string* ne može upisati decimalni broj bez prethodnog prebacivanja u *string* (izbjegava se potencijalno krivi izračun zbog pretvaranja tipa podatka). Pomoću statičkih tipova razvojno okruženje (IDE) može ukazati na pogreške prije pokretanja koda jer poznavanjem tipa varijable može znati koje se metode mogu izvršiti nad njom. Ukoliko se ne zna točan tip podatka ili ga se ne želi striktno deklarirati, TypeScript daje mogućnost definiranja varijable tipa *any* koja će se ponašati kao standardna JavaScript varijabla [6, str. 97]. Prednost ovakvog pisanja koda je velika, pogotovo kod glomaznih projekata jer se osigurava veća robusnost. Glavni nedostatak je to što se moraju definirati statički oblici za metode biblioteka kao što su jQuery kako bi se mogle koristiti u TypeScript kodu. Za popularnije biblioteke postoje gotove statičke definicije te se mogu uvesti kao gotov kod u projekt.

2.2.3 Knockout.js

Knockout.js je biblioteka za razvoj klijentske strane kod *data-driven* web stranica. Čitav njen kod u potpunosti je izrađen u standardnom JavaScriptu. Ova biblioteka koristi MVVM uzorak (engl. *pattern*) gdje M označava model gdje su podatci spremljeni, V označava pregled (engl. *view*) gdje se nalazi prikaz – HTML te VM (engl. *ViewModel*) koji je posrednik između modela i pregleda. Unutar VM-a nalaze se JavaScript metode kojima se manipulira sadržajem za prikaz i sadržajem unutar modela. Ova struktura prikazana je slikom 2.2.



Sl. 2.2.MVVM uzorak [7, str. 14]

Za spajanje modela i pregleda koriste se *observable* objekti koji prate svaku promjenu svojih polja, a po pojavi promjene automatski se mijenja vezani sadržaj na pregledu. Kako bi se stvorila veza između pregleda i VM-a, u HTML se dodaju posebni atributi (veze) na temelju kojih VM točno zna što treba učiniti u trenutku promjene. Najčešće korištena veza je ona gdje se prikazuje trenutna vrijednost objekta, ali mogućnosti su vrlo raznolike. Osim osnovnih veza, moguće je stvoriti i vlastite veze koje moraju implementirati sve metode koje je Knockout definirao u svojoj dokumentaciji. Osim prikaza vrijednosti, standardne veze omogućavaju pozivanje metoda ili dodavanje klasa nekom elementu u slučajevima korisnikove interakcije s aplikacijom. Pod interakcije smatraju se događaji kao što su klik na element, upis vrijednosti u polje itd.

Observable objekti osim u osnovnoj, prethodno spomenutoj formi, dolaze i u obliku *computed observable*te u obliku *observableArray*. Računski (*computed*) objekti su dinamički kreirane vrijednosti. Njima se omogućava spajanje više *observable* objekata u jedan, a taj novonastali objekt ažurira vrijednost promjenom bilo koje vrijednosti od koje je nastao [7, str. 22]. *ObservableArray* je standardna JS lista koja u sebi sadrži *observable* objekte. Dodavanjem ili uklanjanjem elementa na bilo kojem mjestu u nizu obavještavaju se veze i sukladno tome osvježava se prikaz.

3. TWITTER ZID

U ovom poglavlju opisat će se stvarna izvedba aplikacije koristeći tehnologije opisane u prethodnome poglavlju. Prije početka izrade aplikacije bilo je potrebno registrirati istu na Twitterovim stranicama za razvojne inženjere. Po uspješnoj registraciji generiraju se tajni podatci koje aplikacija kasnije koristi za autoriziranje svojih zahtjeva.

Gledano sa strane korisnika, aplikaciju je moguće rastaviti na tri komponente. Na prvoj komponenti nalaze se tri liste tweetova s trenutno najpopularnijim temama na Twitteru. One se automatski listaju i u trenutku dolaska na dno ponovno se odlazi na API kako bi se preuzeo novi

sadržaj s istom temom. Ukoliko korisnik drži pokazivač miša iznad liste, lista se zaustavlja te se omogućava ručna kontrola nad njom. Micanjem pokazivača s liste ponovno se aktivira automatsko listanje. Ukoliko korisnika ne zanimaju automatski dohvaćene teme, ima mogućnost potražiti teme koje ga zanimaju i pratiti njihov sadržaj. Na drugoj komponenti nalazi se korisnikov profil gdje može pratiti svoje tweetove, prijatelje i sl. Treća komponenta su profili ostalih osoba na kojima se mogu pratiti objavljeni sadržaji osobe, njihovi prijatelji, kontrolirati status praćenja i drugo. Na svakome tweetu korisniku su omogućene akcije odgovora, dijeljena i označavanje omiljenim.

3.1 Struktura

Aplikacija je razdvojena na dva projekta. Prvi projekt je *ObjectData* unutar kojega se nalaze C# DTO klase koje se koriste unutar poslužiteljskog koda. DTO (engl. *Data Transfer Object*) su klase koje sadržavaju povezane podatke i nikakvu operacijsku logiku (ne sadrže metode) [8]. One se mogu koristiti na svim razinama aplikacije i služe kao komunikacijski objekti. Različiti objekti mogu se mapirati u DTO objekte na svim razinama aplikacije s ciljem postizanja jedinstvenog oblika podatka kojim će sve razine znati upravljati.

Drugi projekt sadrži API upravljače i ostali sadržaj aplikacije potreban za klijentsku stranu. Unutar mape *APP* nalaze se modeli SPA aplikacije i njima pripadajući HTML nacrti. *App_Start* mapa sadrži kod kojim se definira način generiranja URL ruta za .NET Web API. Unutar *Content* mape sadržane su slike, fontovi i CSS korišten u aplikaciji. Mapa *Scripts* sadrži JavaScript kod biblioteka koje se uvoze uz to i .d.ts datoteke koje su definicije statičkih tipova za uvezene biblioteke. Unutar datoteke *Web.config* nalaze se postavke aplikacije u koje su spremljeni tajni podatci za autorizaciju poziva. *Index.html* je početna stranica unutar koje se ubacuju HTML nacrti.

3.2 Stvaranje i funkcionalnost SPA

Prilikom pokretanja aplikacije otvara se *Index.html* dokument. Prva JS skripta koja se pokreće jest ona koja definira bazni URL aplikacije i sprema ga u globalnu varijablu. Bazni URL mora se definirati kako bi se moglo definirati URL-ove za dohvat podataka sa poslužitelja. Ova vrijednost se kreira uzimanjem parametara window.location.protocol i window.location.host iz window objekta kojeg kreira preglednik. Potom se učitava require.js, gotova JS biblioteka koja služi za asinkrono učitavanje ostalih korištenih skripti. Po završetku učitavanja require.js-a započinje njegova konfiguracija koja se nalazi unutar *appStart.ts* datoteke. U konfiguraciji

navode se putanje do svih skripti koje se koriste u aplikaciji. Za skripte koje ovise o drugima moguće je definirati međusobne ovisnosti na temelju kojih require.js određuje redoslijed učitavanja. Unutar konfiguracije potrebno je navesti i lokaciju nacrta (engl. *template*) HTML koda koji će se kasnije dinamički učitavati također kroz require.js.

3.2.1 Model mainVM

Posljednja skripta koju require.js učita kod pokretanja jest *mainVM* koja predstavlja korijen aplikacije. Klasa Main je nacrt aplikacije i odmah nakon njenog definiranja kreira se njena instanca. Budući da su svi ostali modeli (VM) sadržani unutar ove instance, ne postoji niti jedno drugo mjesto u aplikaciji iz kojeg se može stvoriti nova instanca Main klase. Ovakav način oblikovanja strukture naziva se *singletone*. *Signletone* je, dakle, uzorak kod kojeg smije postojati samo jedna instanca definirane klase. Ukoliko instanca postoji, pozivanjem konstruktora vraća se referenca na postojeću instancu [9]. U ovome slučaju on predstavlja dijeljeni imenski prostor (engl. *namespace*) koji ima ulogu jedinstvene točke pristupa funkcijama. Parametri koji se nalaze unutar Main klase su instance VM potkomponenti aplikacije. Unutar svakog modela potkomponente nalaze se pripadajući *observable* objekti i metode koje služe za obrađivanje korisnikovih interakcija. U konstruktoru Main klase pozivaju se metode za inicijalizaciju postavki aplikacije. Prva metoda postavlja globalne varijable te provjerava postoje li korisnikovi autorizacijski podatci pohranjeni u HTTP kolačiću (engl. *cookie*). Ukoliko postoje, znači da je korisnik već prijavljen u aplikaciju te se zaobilazi prijava korisnika. Prije pozivanja druge metode potrebno je definirati listu svih ruta koje mogu biti aktivne u aplikaciji. Pri pozivanju iste definiraju se akcije koje će se pozvati promjenom URL-a na neku od vrijednosti upisanih u listi definiranih ruta. Za upravljanje rutama u aplikaciji koristi se sammy.js. On je konfiguriran na način da ukoliko nađe URL u popisu ruta, mijenja vrijednost imena trenutno aktivnog nacrta HTML-a. Ime aktivnog HTML-a je *observable*, a veza na njega unutar *Index.html* dokumenta je *template*. Promjenom njegove vrijednosti knockout.js će započeti preuzimanje nacrta s novopostavljenim imenom. Nacrt se preuzima s rute koja je definirana u require.js postavkama kao što je to prethodno opisano. Po uspješnom učitavanju, sadržaj HTML-a se popunjava onim iz dohvaćenog nacrta. Na ovaj način ostvarena je SPA funkcionalnost jer korisnik zapravo nikada ne napušta početnu stranicu. Unutar Main klase nalazi se metoda za prikaz skočnog prozora s obavijestima koje se koriste u svim komponentama aplikacije.

3.2.2 Model loginVM

U ovome modelu nalaze se metode i parametri vezani za prijavljivanje korisnika u aplikaciju. Osnovni parametar je *observable* objekt koji sadrži korisnikove podatke (korisničko ime, URL slike profila…) na temelju kojeg aplikacija zna koji je korisnik trenutno prijavljen. Ukoliko je korisnik već prijavljen u aplikaciju, njegovi podatci čuvaju se u HTTP kolačiću te se svakim pokretanjem aplikacije iz njega iščitavaju i pohranjuju u navedeni korisnički objekt.

Kod web aplikacija izbor načina trajne pohrane podataka nije velik. Svodi se na pohranjivanje u kolačić ili spremanje na lokalni spremnik preglednika (engl. *localstorage*). Prednost lokalne pohrane je u većoj količini dostupne memorije (5MB za jednu vrijednost - varijablu), a nedostatci su nemogućnost davanja roka isteka vrijednosti nakon kojeg bi se sadržaj trebao obrisati te što se vrijednostima ne može pristupiti s poslužiteljske strane. S druge strane, kolačići su ograničeni na maksimalnu veličinu od 4kB, a prednost im je mogućnost definiranja isteka i mogućnost čitanja s poslužitelja[10]. Budući da se podatak o korisniku za neke zahtjeve prema API-u mora znati, odabran je pristup pohrane s kolačićem.

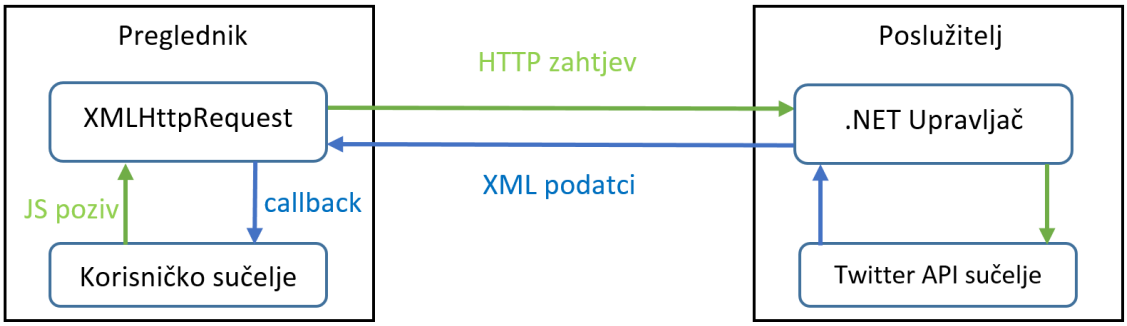
Ukoliko korisnik dođe na stranicu, a u kolačiću nema sadržaja, postavlja se zastavica da korisnik nije prijavljen te se aplikacija odvodi na rutu za prijavu korisnika. Klikom na gumb za prijavu, korisnika se odvodi na Twitterovu stranicu za prijavu korisnika. Nakon uspješne prijave korisnik se ponovno vraća u aplikaciju, a po povratku aplikacija zaprima OAuth tokene koje se kroz metode loginVM pohranjuju u još jedan kolačić. Ovi tokeni koriste se za autorizaciju korisnikovih zahtjeva prema Twitterovom API-u. Kada se korisnik uspješno prijavi, njegovi podatci unose se u *observable* objekt i pripadajući kolačić te se aplikacija obavještava kako je prijava uspješno završena nakon čega se učita početna stranica.

3.2.3 Model twitterVM

Ovaj model vezan je za početnu stranicu aplikacije, odnosno komponentu s listama popularnih tweetova. Sadrži veliku količinu *observable* objekata koji čuvaju dohvaćene informacije i prate korisnikove akcije na stranici. Ukoliko je korisnik prijavljen u aplikaciji i nalazi se na početnoj stranici, aplikacija odlazi na poslužitelj dohvatiti podatke za prikaz. Svaki zahtjev koji aplikacija uputi prema poslužitelju odvija se putem AJAX poziva.

AJAX je oznaka za asinkroni JavaScript i XML (engl.*Asynchronous JavaScript And XML*). On služi za komunikaciju klijenta i poslužitelja preko XMLHttpRequest objekata. Ima mogućnost slanja i zaprimanja raznolikih tipova podataka počevši od JSON-a, XML-a, HTML-a pa do

binarnih datoteka i objekata [11 str. 21 - 31]. Glavna mu je prednost asinkronost koja omogućava izvođenje funkcija bez osvježavanja stranice. Na ovaj način omogućava se dohvat manjeg podskupa podataka ovisno o korisnikovoj interakciji bez kojega bi sav sadržaj morao prethodno biti učitan na klijentovu pregledniku. Ovime se poboljšavaju performanse, ali i korisničko iskustvo. Funkcioniranje AJAX-a unutar aplikacije prikazano je slikom 3.1.



Sl. 3.1. AJAX zahtjev

Unutar TypeScript koda, AJAX poziv postiže se pomoću sljedeće naredbe:

```
$.ajax({
      url: window["baseUrl"] + "/api/Data/getTweets",
      type: "GET",
      contentType: false,
      success: function (data) { self.initFeed(data); },
      error: function (data) { self.failedDataRetrieve(); }
  });
```

Ovaj kod predstavlja JS poziv na slici 3.1. Parametri koji se nalaze unutar vitičastih zagrada su parametri XMLHttpRequesta. Parametar *url* predstavlja putanju do metode na poslužitelju, *type* je HTTP *verb*, *contentType* je tip podatka koji se šalje (u ovom pozivu radi se GET i nema slanja podataka, stoga je ovaj parametar postavljen u *false*), zatim slijedi definiranje callback metoda. Metoda navedena u *success* parametru poziva se u slučaju uspješnog odgovora s upravljača, a *error* u slučaju neuspjeha. Odluku o uspjehu ili neuspjehu poziva AJAX donosi na temelju HTTP statusnog koda koji dobije s poslužitelja. Pod uspjeh uzimaju se odgovori sa statusnim kodom od 200 – 299 te statusni kod 304 (*NotModified*) uz posebnu obradu. Većina callbacka u slučaju uspjeha obavlja obradu zaprimljenog objekta i njegovo pohranjivanje u *observable* objekt. U slučaju neuspjeha, korisniku se prikazuje obavijest o neuspješnom zahtjevu.

Prije opisivanja načina obrade Knockout.js veza potrebno je poznavati tri ključne oznake: $root, $parent i $data. Ključnom riječju $root ukazuje se na korijen aplikacije, što je u ovom slučaju instanca klase Main. Pomoću $parent Knockout.js označava roditeljski objekt onoga u kojem se sada nalazi. Primjerice, ako se *foreach* vezom prolazi kroz listu objekata koji se nalaze u objektu

u instanci tweetsVM klase, tada će $parent označavati instancu tweetsVM. Ova ključna riječ ima mogućnost prelaženja nekoliko generacija unatrag. U tom slučaju ona je oblika $parents[n] gdje n predstavlja broj razina za koje se treba vratiti unatrag. S $data označavamo da se radi o sadržaju trenutnog podatkovnog konteksta. Njega nije potrebno implicitno naglašavati.

Nakon uspješnog dohvata tweetova kroz prethodno opisani kod započinje njihovo učitavanje unutar HTML nacrta *tweetList.html*. Podatci koje ovaj nacrt prima su lista tweet objekata, a on *foreach* vezom prolazi kroz svaki element liste i za svaki popunjava podnacrt. Svaka veza unutar HTML-a definira se unutar atributa *data-bind,* primjerice:

```
<div data-bind="html: $root.tweetsVM.parseUTCDate(dateTimeCreated)"></div>
```

Veza *html* označava da će HTML sadržaj elementa s vezom biti ono što vrati pozvana funkcija. Mjesto gdje se funkcija nalazi vidljivo je iz putanje do nje. U korijenu aplikacije traži se objekt s imenom tweetsVM (u ovom slučaju instanca istoimene klase) i unutar njega postoji metoda *parseUTCDate* koja prima jedan parametar. Ovdje predani parametar je dateTimeCreated koji se, budući da ispred njega nema nikakve putanje, uzima iz trenutnog podatkovnog konteksta. Kao što je prethodno navedeno, ovaj HTML se ispunjava za svaki tweet u listi tweetova. Iz ovoga se može zaključiti da je trenutni kontekst zapravo tweet za koji se stvara HTML.

Osim *html* i *foreach* veze, unutar ovog pregleda korištene su još: *text, visible, event* te *attr*. Veza *text* će u dani element postaviti tekstualni sadržaj dane vrijednosti. Ona ne mora nužno biti funkcija, može biti i parametar *observable* objekta. Takva veza izvedena je na sljedeći način:

```
<div data-bind="text: authorDisplayName"></div>
```

Kod ovakvog slučaja u trenutnom kontekstu traži se parametar *authorDisplayName* te se vrijednost pohranjena unutar njega upisuje unutar div elementa. Veza *if* i *visible* služe za uvjetno skrivanje i prikazivanje elementa. Ukoliko je predana vrijednost laž (logička nula), element se ne prikazuje. Razlika između ove dvije veze je u tome što *if* uklanja element iz DOM-a, a *visible* pomoću parametara CSS-a skriva element. Veza *attr* omogućava uređivanje i dodavanje HTML atributa kao što su *id, class, src, title*… Najčešće korištena veza u aplikaciji je *event*. Ona služi za obradu korisnikovih interakcija na vezanom elementu. Pod interakcije podrazumijevaju se: klik miša, prelazak pokazivačem preko elementa, označavanje elementa, listanje preko elementa (engl. *scroll*), upisivanje znaka i ostali HTML DOM događaji (engl. *event*). Prednost vezanja događaja na ovaj način je u tome što se u funkcije koje se pozivaju mogu predati parametri iz trenutnog konteksta. Kada bi se ista funkcionalnost željela postići sa standardnim JS-om, u pozvanu funkciju morao bi se poslati događaj iz kojeg bi se izvadio element na kojem se događaj

dogodio. S njega bi tada trebao čitati atribut kao što je primjerice *id* iz kojega bi bilo moguće saznati više podataka. Za primjer je dan kod za otvaranje modalnog prozora za odgovor na tweet:

```
data-bind="event:{click: function(){$root.tweetsVM.openNewTweet(authorDisplayName)}}"
```

Kao što je vidljivo u gornjem kodu, svi događaji vezani za jedan element pišu se unutar vitičastih zagrada nakon ključne riječi *event*. U ovom slučaju radi se o samo jednom događaju, ali moguće je definirati i više njih na način da se njihove definicije razdvajaju zarezima. Knockout.js veze koje na kraju imaju zagrade tretira kao funkcije, stoga se one odmah kod stvaranja HTML-a izvode. Budući da je ovo slučaj i kod deklariranja događaja na klik miša, kad metoda ne bi bila stavljena unutar *function(){}* zapisa, vezana funkcija bi bila izvedena odmah. Kad se metoda veze stavi unutar bezimene funkcije, tada neće biti izvedena prilikom učitavanja [12].

TweetsVM sadrži funkcije koje služe za kreiranje tweeta, njegovo slanje na poslužitelj, označavanje omiljenim tweetom i slično. Metode koje služe za komunikaciju od korisnikove interakcije u sučelju do poslužitelja većim dijelom objašnjene su kroz prethodni opis načina funkcioniranja tehnologije stoga se njih neće detaljno objašnjavati.

Metode koje nisu namijenjene standardnoj komunikacijskoj i obrađivačkoj funkcionalnosti, a zanimljive su zbog opisa tehnologije su *charCounter* i *setupAutoScroll*. Metoda *charCounter* je funkcionalna primjena *computed* metode spomenute u poglavlju 2.2.3. Budući da su tweetovi ograničeni na 160 znakova, prilikom korisnikova unosa novog tweeta bilo je potrebno provjeravati broj trenutno unesenih znakova. Polje za unos u modalnom prozor za stvaranje tweeta ima vezu:

```
data-bind="value: tweetsVM.replyComment, valueUpdate: 'afterkeydown'"
```

Veza *value* namijenjena je za HTML elemente koji služe za unos vrijednosti. Funkcija joj je da prikazuje vrijednost elementa koji joj je dodijeljen i u isto vrijeme, ukoliko se dogodi promjena vrijednosti, istu mijenja i u *observable* objektu unutar VM-a. Zadane postavke Knockouta definiraju da se vrijednost u VM-u ažurira tek nakon što se makne fokus s polja za unos. Budući da je ovdje potrebno da se vrijednost promjeni nakon svakog unosa znaka, važno je naglasiti da Knockout ažurira VM nakon svakog pritiska tipke. Ovo se postiže pomoću parametra *valueUpdate* kao što je prikazano u danom kodu.

Unutar *charCounter* metode radi se provjera nad vrijednošću upisanom u *replyComment* objekt. Budući da je ona *observable* i da se zbog gore opisane veze ažurira pritiskom tipke, *charCounter* će također biti pozivan. Ovime se osigurava da se validacija odradi prilikom svakog pritiska

tipke. Validacija podrazumijeva provjeru duljine *replyComment* vrijednosti. Ukoliko je ona manja od 160 znakova, ne radi se nikakva promjena, a ukoliko je veća od 160 znakova, sav sadržaj koji se nalazi iza zadnjeg dozvoljenog mjesta se briše. Brisanje viška sadržaja odmah je prikazano korisniku u polju za unos zbog postojeće *value* veze. Iz tog razloga korisnik zapravo ima dojam kao da sadržaj koji prelazi granicu nije niti upisao. Broj znakova koji je unio vraća se kao rezultat metode na sučelje čime korisnik zna koji mu je broj znakova preostao za unos.

Metoda *setupAutoScroll* poziva se tek nakon što se liste tweetova ispune. Ona omogućava funkcionalnost automatskog listanja sadržaja. Ovakvo vladanje postignuto je pomoću JS funkcije *setInterval* koja poziva predani kod nakon isteka zadanoga vremena. Pozivi se neprestano ponavljaju sve dok se ne pozove *clearInterval*. U ovome slučaju vrijeme između poziva postavljeno je na 40ms. U svakom pozivu sve tri liste pomiču se za jedan pixel prema dolje. Prije pomaka svake od listi provjerava se aktivnost pauze na pojedinoj listi. Prelaskom, odnosno izlaskom miša s područja liste pomoću Knockout.js veza, aktivira se ili miče pauza na pojedinoj listi. Lista na kojoj je aktivna pauza se ne lista automatski te se na njoj omogućava ručno upravljanje (listanje pomoću miša). U slučaju da je neka lista dosegla dno, ova metoda stvara AJAX poziv kako bi se dohvatio novi sadržaj.

3.2.4 Modeli profileSearchVM i tagSearchVM

Kao što je moguće zaključiti iz njihovih naziva, oba modela služe za pretraživanje sadržaja na Twitter društvenoj mreži. Model *profileSearchVM* koristi se za pretraživanje profila osoba, a *tagSearchVM* za pretraživanje oznaka. Oba modela izvedena su na jednak način, razlika je jedino u metodi na API-u koje pozivaju i u načinu obrade zaprimljenih podataka.

Polja u kojima korisnik radi unos imaju vezu *value* koja je opisana u prethodnom poglavlju. Unosom svakoga znaka vezani *observable* objekt se mijenja što izaziva poziv *computed* metode. Svaki od modela ima globalnu varijablu koja predstavlja JS setTimeout objekt. On daje mogućnost odgađanja poziva predanoga koda za period koji mu je zadan.

```
public searchProfile = ko.computed(() => {
clearTimeout(this.timer);
if (this.searchInput().length > 0) {
this.timer = setTimeout(this.querySearch, 1000);}})
```

U gornjem kodu prikazana je spomenuta *computed* metoda. Vrijednost *searchInput* jest ona koja se upisuje u polje za unos. Svakom promjenom te vrijednosti izaziva se poziv prikazanog koda. Prva metoda koja se poziva je JS metoda *clearTimeout* koja otkazuje poziv funkcije koji je bio napravljen pomoću *setTimeout* poziva. Ovo rezultira time da se tek 1000ms nakon što korisnik

prestane unositi vrijednosti odlazi na API po podatke. Bez ovakvog vladanja za unos vrijednosti „test“ četiri puta bila bi pozvana metoda *querySearch*, a s ovakvim vladanjem vrlo vjerojatno samo jednom (ovisno o brzini unosa korisnika). Ostale metode koriste se za AJAX pozive prema upravljaču i za obradu zaprimljenih podataka.

3.2.5 Modeli myProfileVM i othersVM

Ovi modeli vezani su za komponente aplikacije za prikaz vlastitog profila i profila drugih osoba. Metode koje se nalaze unutar ovih modela izvedene su na sličan način kao i neke od metoda opisane kroz ovo poglavlje stoga će se opisati samo funkcionalnosti koje do sada nisu korištene.

Zbog ograničenja u broju poziva na Twitterov API, za dohvat podataka uvedene su vremenske oznake. Nakon uspješnog dohvata podatka, istome se stavlja vremenska oznaka dohvata koja sadržava vrijednost trenutnog vremena. Prilikom svakog sljedećeg dohvata provjerava se njegova vremenska oznaka. Ukoliko je prošao dovoljan period od zadnjeg odlaska po podatke, omogućava se ponovni AJAX poziv, ako pak nije, i dalje se koriste zadnje dohvaćeni podatci pohranjeni u *observable* objektu. Vrijeme ograničenja nije veliko kako bi se osigurala ažurnost korisniku prikazanih podataka.

Kod prikaza osoba na profilu koristi se straničenje (engl. *pagination*) kako bi se ograničila količina podataka koji se korisniku u tom trenutku prikazuju. Svaka stranica prikazuje po 30 osoba. Posljednja stranica može sadržavati manje od 30 osoba. Straničenje je ostvareno pomoću dva *observable* objekta. U jednom objektu drže se sve osobe koje su dohvaćene API-em, a u drugom se drže osobe koje se trenutno prikazuju u korisničkom sučelju. Prebacivanjem stranice dohvaća se njen broj na temelju kojeg se sazna indeks prvog elementa u nizu kojeg je potrebno pokazati. Poznato je kako elementi u nizu počinju od indeksa nula pa sve do indeksa n. Iz ovoga se zaključuje da ako num označava broj stranice vrijedi:

```
var startIndex = (num - 1) * 30,
    endIndex = startIndex + 30;
```

Ako je označena stranica num=1, početni indeks bit će nula, a krajnji pak 30 odnosno 29 jer se uzimaju svi elementi do posljednjeg indeksa, ne i onaj na posljednjem indeksu. Po određivanju indeksa započinje upisivanje. Upisivanje se vrši na način da se u objekt za prikaz upišu elementi od određenog početnog do krajnjeg indeksa u objektu gdje se drže svi podatci. Promjenom sadržaja objekta za prikaz automatski se mijenja sadržaj na korisničkom sučelju. Na ovaj način ostvaruje se dojam listanja podataka po stranicama.

3.3 Implementacija poslužiteljske strane

Na poslužiteljskoj strani nalaze se dva ASP.NET Web API upravljača: *BaseController* i *DataController*. Svaka metoda kojoj se pristupa s korisničke strane ima svoju definiciju unutar *DataControllera*. On nasljeđuje *BaseController,* stoga ima implementacije svih metoda koje se u njemu nalaze. U stvarnosti nikada se ne stvara instanca *BaseControllera,* on je stvoren jedino iz razloga preglednosti koda. U njemu se nalaze metode za prebacivanje dobivenih Twitterovih JSON odgovora u odgovarajuće DTO objekte koje korisnička strana također ima definirane.

3.3.1 Definiranje ruta

Osnovni oblik ruta na poslužitelju definira se u trenutku pokretanjem aplikacije na poslužitelju, a definiran je unutar *WebApiConfig* klase:

```
config.Routes.MapHttpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/{id}",
    defaults: new { id = RouteParameter.Optional });
```

Parametar *routeTemplate* prikazuje način na koji će Web API interpretirati rute. Podrazumijeva se da prije gore navedene rute stoji putanja do poslužitelja. Vidljivo je kako nakon „/api/“ slijedi ime upravljača, a zatim proizvoljni parametar (najčešće ime metode).
Pomoću anotacija metode u upravljaču moguće je definirati kako će izgledati ruta do nje.

```
[Route("api/Data/Authorize")]
[HttpGet]
public string Authorize() { … }
```

Anotacije se stavljaju iznad definicije metode, a mogu sadržavati različite informacije. Ovdje se primjenjuju za definiranje rute do metode i HTTP *verba* koji je važeći za tu metodu. Važno je naglasiti da je moguće imati više metoda na istoj ruti koje će obavljati različitu operaciju obzirom na predani HTTP *verb*[13]. U gore navedenom primjeru nije bilo nužno navoditi rutu jer je ta ruta već podrazumijevana, ali radi konzistentnosti koda ipak je navedena. Ukoliko se ne navede ruta, podrazumijeva se da iza imena upravljača slijedi ime metode. Primjer stvarnog poziva gore navedene metode pomoću AJAX-a izgledao bi tako da se u XMLHttpRequest za parametar *url* postavi vrijednost: http://localhost/TwitterWall.SPA/api/Data/Authorize, a parametar *type* postavlja se u vrijednost GET.

3.3.2 Autorizacija zahtjeva prema Twitter API sučelju

Twitterovo API sučelje autorizaciju vrši putem OAuth tokena. U standardnom klijent-poslužitelj okruženju kada klijent od poslužitelja zatraži zaštićeni podatak prvo se mora obaviti

prijava korisnika pomoću lozinke ili nekog drugog sredstva. Ovakav pristup nije prihvatljiv u slučajevima kada aplikacija koja nije nastala od davatelja usluge (engl. *third-party*) pokušava izvršiti dohvaćanje podataka u ime korisnika s njegovim sigurnosnim podatcima. Glavni nedostatci ovog pristupa prema [14] su:

- Aplikacija koja se autorizira u ime korisnika mora negdje pohraniti lozinku (obično u obliku čistoga teksta – iznimno nesigurno).
- Aplikacija korisnika dobiva jednaka prava kao i korisnik. Nema mogućnosti da korisnik odredi koja će se prava aplikaciji dodijeliti.
- Korisnici ne mogu zabraniti aplikaciji da ne spremi njihove podatke za autorizaciju u svoje baze podataka.

Zbog navedenih razloga stvoren je novi način autorizacije takvih aplikacija koji će imati ograničen pristup resursima. Korisniku se prilikom davanja dozvola aplikaciji daje na izbor razina prava koju će joj dati na korištenje. Ista ta dozvoljena prava korisnik može u bilo kojem trenutku ukloniti. Nakon što korisnik dopusti aplikaciji korištenje njegovih resursa, aplikacija dobiva tokene kojima se na poslužitelju autorizira u ime korisnika. Ovaj protokol naziva se OAuth i zadnja službena inačica ovog protokola je OAuth2.0. Twitterovo API sučelje koristi OAuth 1.0.

OAuth na standardnu klijent-poslužitelj autorizaciju uvodi treći element: vlasnika resursa. Klijent u ovom slučaju nije vlasnik resursa, nego aplikacija koja se tako ponaša. Osim navedenih pojmova, OAuth definira još tri: klijentski podatci za prijavu, token i zaštićeni resurs. Klijentski podatci za prijavu (engl. *credentials*) predstavljaju par jedinstvenog identifikatora klijenta i dijeljene tajne (javni ključ) kojima se klijent autorizira na poslužitelju prilikom slanja zahtjeva. Zaštićeni resurs predstavlja sadržaj koji klijent može preuzeti s poslužitelja nakon uspješne autorizacije. Token je jedinstveni identifikator kojeg klijent koristi za autorizaciju u ime vlasnika resursa. Svaki token ima dijeljenu tajnu kojom se utvrđuje vlasništvo tokena.

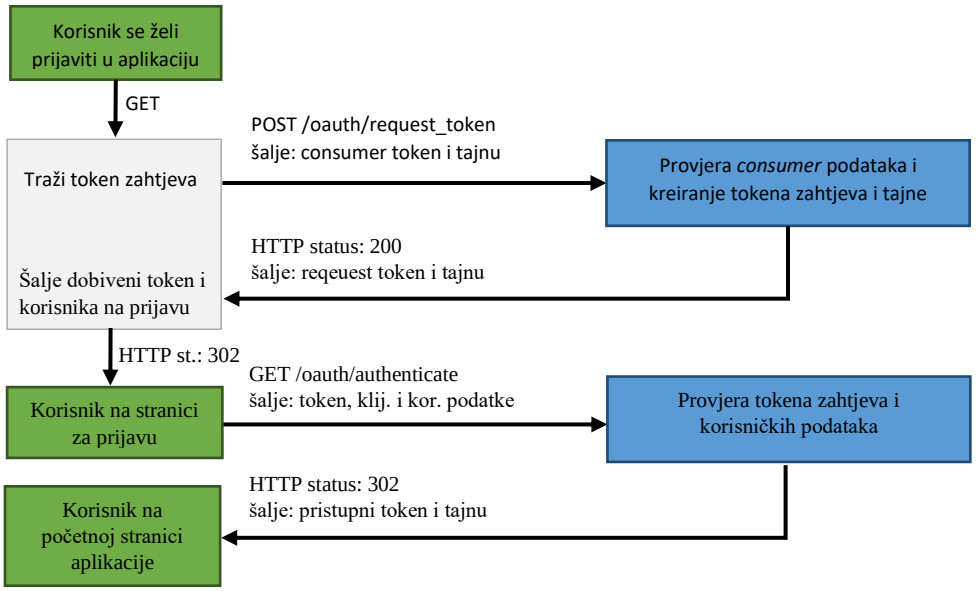
Token zahtjeva (engl. *request token*) jest token kojim klijent dobiva pravo od poslužitelja uputiti vlasnika resursa do stranice za prijavu. Poslužitelj ga udjeljuje klijentu samo ako su njegovi podatci za prijavu ispravni. Token pristupa (engl. *access token*) klijent zaprima tek nakon što korisnik napravi prijavu i dozvoli klijentu pristup zaštićenim resursima.

Za Twitterov API, proces obavljanja dohvata pristupnog tokena provodi se kroz sljedeće korake:

- Korisnik (vlasnik resursa) otvara aplikaciju (klijent) i odabire gumb za prijavu

- Aplikacija odlazi na Twitter API (poslužitelj) i traži token zahtjeva
- API provjerava ispravnost klijentskih podataka, ako je sve uredno vraća token zahtjeva
- Aplikacija zaprima token zahtjeva i šalje korisnika zajedno sa tokenom na Twitter radi autorizacije korisnika.
- Twitter otvara stranicu za prijavu na kojoj jasno definira korisniku koja će prava dozvoliti aplikaciji ukoliko se prijavi
- Uspješnom prijavom stvara se pristupni token i njegova tajna koji se zajedno sa korisnikom vraća aplikaciji.

Ovaj postupak s navedenim HTTP statusima grafički je prikazan na slici 3.2. Uspješnim zaprimanjem pristupnog tokena aplikacija ima sve potrebne komponente za dohvat zaštićenih resursa s Twitter API sučelja.



Sl. 3.2 Prijava korisnika i dohvat pristupnog tokena

Zelenom bojom na slici 3.2 označena su stanja koja se izvode u korisnikovu pregledniku (klijentu), sivo su stanja koja se izvode na Web API upravljačima, a plavo su označena stanja koja se izvršavaju na Twitterovom API sučelju.

OAuth je vrlo koristan za ovakvu primjenu, općenito je dobro prihvaćen i često korišten. Glavni mu je nedostatak osjetljivost na napade mrežne krađe identiteta (engl. *phishing*), jer je korisnika moguće usmjeriti na stranice koje su vizualno potpuno jednake službenim stranicama, a zapravo služe za prikupljanje korisnikovih autorizacijskih podataka u maliciozne svrhe.

Oblik autorizacijskog zaglavlja u zahtjevima za dohvat korisnikovih podataka s Twitter API-a strogo je definiran i izgleda kao što je to prikazano u sljedećem kodu:

```
Authorization:
   OAuth oauth_consumer_key="xvz1evFS4wEEPTGEFPHBog",
   oauth_nonce="kYjzVBB8Y0ZFabxSWbWovY3uYSQ2pTgmZeNu2VS4cg",
   oauth_signature="tnnArxj06cWHq44gCs1OSKk%2FjLY%3D",
   oauth_signature_method="HMAC-SHA1",
   oauth_timestamp="1318622958",
   oauth_token="370773112-GmHxMAgYyLbNEtIKZeRNFsMKPR9EyMZeS9weJAEb",
   oauth_version="1.0"
```

Parametar *oauth_consumer_key* dobiven je prilikom registracije aplikacije na Twitterovim stranicama za razvojne inženjere. Ovaj podatak spremljen je unutar konfiguracijskog dokumenta aplikacije i u trenutku instanciranja upravljača čita se pomoću sljedećeg koda:

```
consumerKey = System.Configuration.ConfigurationManager.AppSettings["consumerKey"];
```

Drugi parametar je *oauth_nonce* koji predstavlja jedinstvenu oznaku za svaki zahtjev koji se upućuje prema API-u. U dokumentaciji se navodi da je dovoljno generirati bilo kakvu nasumičnu alfanumeričku vrijednost uz uvjet da je kodirana u base64 vrijednost. U kodu je to postignuto na način da se broj otkucaja trenutnog vremena prebacio u base64 vrijednost:

```
Convert.ToBase64String(new ASCIIEncoding().GetBytes(DateTime.Now.Ticks.ToString()))
```

Parametar *oauth_signature* sadrži vrijednost koja se stvara kroz algoritam za stvaranje potpisa kojem se moraju predati svi parametri zahtjeva, tajna tokena za pristup i tajna tokena dobivenog kod kreiranja aplikacije. Način generiranja ovog potpisa bit će opisan dalje u tekstu. Svrha mu je autorizacija aplikacije koja šalje zahtjev na API.

Konstantne vrijednosti oauth_signature_method i oauth_version čuvaju se unutar klase Constansts, a pristupa im se pomoću sljedećeg koda:

```
Constants.OAUTH_VERSION
```

Vremenska oznaka (*oauth_timestamp*) označava vrijeme kreiranja zahtjeva. Prema dokumentaciji, vrijeme mora biti zapisano kao broj sekundi od Unix epohe. Pomoću C# je to postignuto sa:

```
var timeSpan = DateTime.UtcNow - new DateTime(1970, 1, 1, 0, 0, 0, 0, DateTimeKind.Utc);
return Convert.ToInt64(timeSpan.TotalSeconds).ToString();
```

Parametar *oauth_token* jest vrijednost tokena za pristup dobivenog nakon korisnikove prijave. Kao što je već spomenuto, ova vrijednost čuva se unutar HTTP kolačića, a prilikom stvaranja svakog zahtjeva čita se iz njega pomoću sljedećeg koda:

```
var cookie = Request.Headers.GetCookies("twitterWall_Token").FirstOrDefault();
var cookieData = cookie["twitterWall_Token"].Value.Split('*');
return cookieData[0];
```

Vrijednosti tokena i njegove tajne nalaze se u kolačiću razdvojeni znakom '*'. Ako upisanu vrijednost razdvojimo na niz elemenata tako da se ona lomi na znaku '*', tada se za vrijednost tokena uzima vrijednost na mjestu nula, a za tajnu ona vrijednost na mjestu jedan.

Kreiranje potpisa zahtijeva poznavanje svih parametara koji će se staviti u zahtjev. To podrazumijeva sve prethodno opisane parametre, URL koji se poziva, parametre unutar URL-a i HTTP metodu. Za kodiranje koriste se URL kodiranje i HMAC. URL kodiranje ili *Percent Encoding* standard je za pisanje URI-a. Sve alfanumeričke vrijednosti ostaju jednake, a specijalni znakovi poput razmaka ili uskličnika kodiraju se u oblik %x, gdje x označava kodni broj. HMAC (*Keyed-Hashing for Message Authentication*) je mehanizam za autentikaciju poruka koji koristi kriptografske hash funkcije [15]. U ovom slučaju za hash koristi se SHA-1.

Za primjer stvaranja potpisa poziva prikazat će se metoda kojom se dohvaća trenutni status prijateljstva između korisnika i osobe na čijem profilu se korisnik nalazi.

Prvi korak jest definiranje URL-a za koji se stvara zahtjev:

```
var resource_url = "https://api.twitter.com/1.1/friendships/show.json";
```

Sljedeća stavka jest stvoriti OAuth podatke i upisati u oblik koji je prikazan kod objašnjavanja Twitter OAuth zahtjeva. Na popunjeni OAuth oblik dodaju se parametri koji će se nalaziti u pozivu. U ovom slučaju je to korisničko ime trenutnog korisnika i korisničko ime korisnika čiji je profil otvoren. Svaki od parametara mora se kodirati URL kodiranjem, a u C# se to postiže predavanjem vrijednosti metodi Uri.EscapeDataString():

```
var baseString = string.Format(Constants.OAUTH_BASE_FORM +
$"&source_screen_name={Uri.EscapeDataString(currentUser.displayName)}&target_screen_name=
{Uri.EscapeDataString(targetUser)}",consumerKey, nonce,Constants.OAUTH_SIGNATURE_METHOD,
timeStamp, getOAuthTokenFromCookie(), Constants.OAUTH_VERSION);
```

Ovako kreirana osnovna forma sada se dorađuje na način da se na početak dodaje HTTP metoda, a na kraj URL na koji se odlazi po podatke (bez parametara):

```
baseString = string.Concat("GET&", Uri.EscapeDataString(resource_url), "&",
Uri.EscapeDataString(baseString));
```

U ovom trenutku zahtjev je u potpunosti složen i može se započeti kodiranje korištenjem tajne tokena dobivene kod prijave aplikacije i tajne pristupnog tokena:

```
var compositeKey = string.Concat(Uri.EscapeDataString(consumerSecret), "&",
Uri.EscapeDataString(oauthSecret));
string oauth_signature;

using (HMACSHA1 h = new HMACSHA1(ASCIIEncoding.ASCII.GetBytes(compositeKey)))
{
	oauth_signature =
	Convert.ToBase64String(h.ComputeHash(ASCIIEncoding.ASCII.GetBytes(baseString)));
}
```

Nakon izvođenja gornjeg koda stvoren je oauth_signature parametar koji se ubacuje unutar stvarnog zahtjeva. Budući da su u ovom trenutku skupljeni svi parametri nužni za odlazak na API, moguće je napraviti poziv.

3.3.3 Slanje zahtjeva i obrada podataka

U ovom poglavlju objasnit će se postupak koji slijedi nakon uspješnog kreiranja autoriziranog zahtjeva. Kreirani zahtjev se upućuje prema Twitterovom API-u koji vraća zatražene podatke ukoliko je autorizacija uspješno izvršena.

Kao primjer obradit će se metoda za dohvat objavljenih tweetova osobe čiji profil je trenutno otvoren. Podrazumijeva se da je prethodno kreirano autorizacijsko zaglavlje kao što je objašnjeno u prethodnom poglavlju. Započinje se kreiranjem novog HTTP zahtjeva:

```
HttpWebRequest request = (HttpWebRequest)WebRequest.Create(resource_url);
request.Headers.Add("Authorization", authHeader);
request.Method = "GET";
request.ContentType = "application/x-www-form-urlencoded";
```

Zahtjev se kreira metodom WebRequest.Create() kojoj se za parametar predaje URL koji se treba pozivati na Twitterovom API-u. URL mora biti u punom obliku tj. osim osnovne putanje mora sadržavati i pripadajuće parametre. U zaglavlje se dodaje autorizacijski atribut kojem se za vrijednost postavljaju kodirani podatci kreirani na način kako je to objašnjeno u prethodnom poglavlju. Potrebno je definirati i HTTP metodu (u ovom slučaju GET) i tip sadržaja.

Slanje kreiranog zahtjeva šalje se GetResponse() naredbom, a odgovor se pohranjuje unutar HttpWebResponse objekta. Ovaj objekt omogućava čitanje sadržaja odgovora pomoću GetResponseStream() metode. Budući da je odgovor zapravo niz znakova formatiranih u JSON, odgovor se čita StreamReaderom. Nakon čitanja podataka potrebno je iste prebaciti iz JSON objekta u C# objekt radi obrade. Postupak prebacivanja objekta iz tekstualnog oblika u stvarni

objekt naziva se deserijalizacija. Za ovaj postupak korišteno je C# proširenje Newtonsoft.JSON. Pozivanjem metode JsonConvert.DeserializeObject stvara se dinamički C# object. Dinamički tip podatka (*dynamic*) omogućava rad nad tipom podatka koji nije striktno definiran. Na ovaj način može se vršiti obrada podatka u objektu bez prethodnog određivanja njegovih parametara. U ovom slučaju je to vrlo korisno jer bez navedene mogućnosti prethodno bi morao biti definiran objekt koji će imati sve parametre jednake kao i odgovor koji je došao s API-a. Odgovori s API-a su vrlo složeni, a prilikom obrade podataka velika se količina dobivenih vrijednosti zanemaruje. Spremanje odgovora i njegova deserijaliazija obavlja se sljedećim kodom:

```
HttpWebResponse response = (HttpWebResponse)request.GetResponse();
dynamic result = JsonConvert.DeserializeObject(new
System.IO.StreamReader(response.GetResponseStream()).ReadToEnd());
```

Problem kod dinamičkog objekta je što se ime parametra u objektu mora točno znati kako ne bi došlo do pucanja koda zbog čitanja nepostojeće vrijednosti. Kodom u nastavku prikazan je način prebacivanja podataka iz *dynamic* objekta u DTO objekte.

```
foreach(var item in result)
{
    var tweet = new SingleTweet();
    var dt = DateTime.Now;
    DateTime.TryParseExact(item.created_at.Value, Constants.TWEET_TIME_FORMAT, new
           System.Globalization.CultureInfo("en-US"), DateTimeStyles.None, out dt);
    var tempUrl = "";
    tweet.tweetId = item?.id_str;
    tweet.tweetText = item?.text;
    tweet.dateTimeCreated = dt.ToString("yyyy-MM-dd HH:mm");
    tweet.isFavorited = item?.favorited;
    tweet.authorDisplayName = item?.user?.screen_name;
    tweet.authorImgUrl = item?.user?.profile_image_url;
    if (item.entities.media != null) { tempUrl = item.entities.media[0].media_url; }
    if (!string.IsNullOrEmpty(tempUrl)) tweet.imagesUrls.Add(tempUrl);
    resultList.Add(tweet);
}
```

Budući da je dobiveni rezultat lista tweetova, foreach petljom prolazi se kroz svaki dobiveni tweet i njega se prebacuje u instancu SingleTweet objekta koji je DTO. Zbog sigurnosti, kod čitanja vrijednosti korišten je tzv. elvis operator (?.). Ovaj operator prvo provjerava je li vrijednost iza točke nepostojeća (null), ukoliko je, tada za učitanu vrijednost postavlja null i ne ide dalje kroz hijerarhiju, ako pak vrijednost postoji, on će ju pročitati kao i standardni operator pristupa (.). Ovime se izbjegava pucanje koda zbog čitanja nepostojeće vrijednosti. Datum koji je bio upisan unutar dobivenog tweet objekta vrlo je složenog oblika i potrebno ga je prebaciti u standardni ISO oblik. C# daje mogućnost stvaranja DateTime objekta iz niza znakova uz pomoć metode DateTime.TryParseExact(). Za parametar očekuje definiciju načina zapisa podatka koji se pokušava pretvoriti u datum. U ovom slučaju taj format se čita iz konstanti. Na temelju tog

podatka C# zna koji će broj pridijeliti godini, mjesecu itd. Iz DateTime objekta kasnije je lako stvoriti tekstualni oblik korištenjem toString() metode koja za parametar prima željeni format ispisa. Kada se završi mapiranje tweeta, on se dodaje na listu koja će se vratiti na klijentsku stranu.

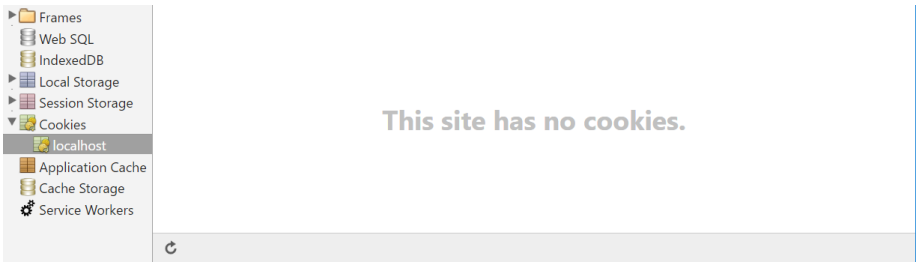
Sve metode unutar upravljača obavljaju sličnu aktivnost, razlika je samo u URL-u na koji se odlazi na API, vrsti podataka koji zaprima i vrsti podatka u koji rezultat prebacuje. U suštini su sve metode vrlo slične i nema potrebe svaku pojedinačno opisivati.

4. TESTIRANJE MOGUĆNOSTI APLIKACIJE

U ovom poglavlju testirat će se i objasniti mogućnosti koje aplikacija pruža. Slijed prolaska započet će korisnikovom prijavom, a završiti korisnikovom odjavom iz aplikacije.

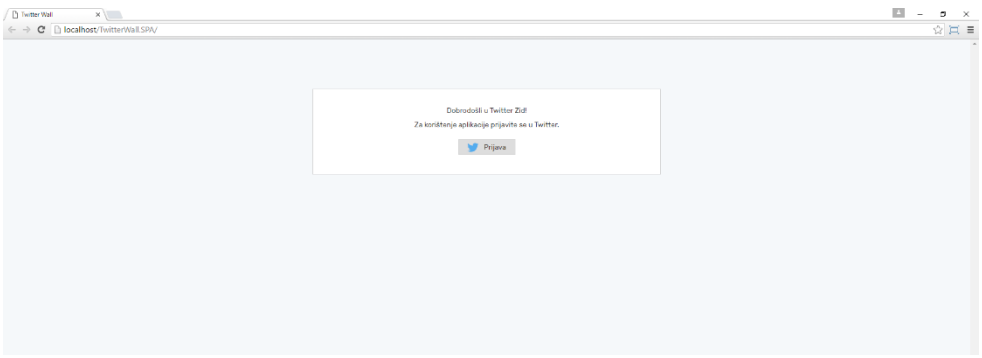
4.1 Prijava korisnika u aplikaciju

Dolaskom korisnika u aplikaciju prvo se provjerava postoji li HTTP kolačić s korisničkim podatcima. Kao što je prikazan slikom 4.1, ne postoji niti jedan spremljeni kolačić. Očekivano je da se u ovoj situaciji otvori stranica koja korisniku daje mogućnost za prijavu.



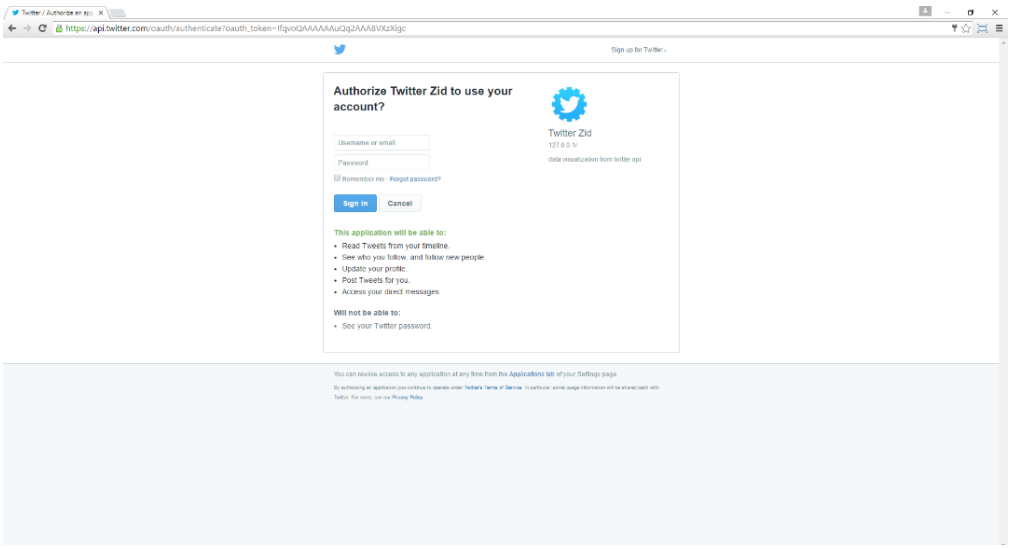
Sl. 4.1 Stanje HTTP kolačića

Ovu situaciju aplikacija je uspješno prepoznala i korisniku je prikazana stranica na slici 4.2.



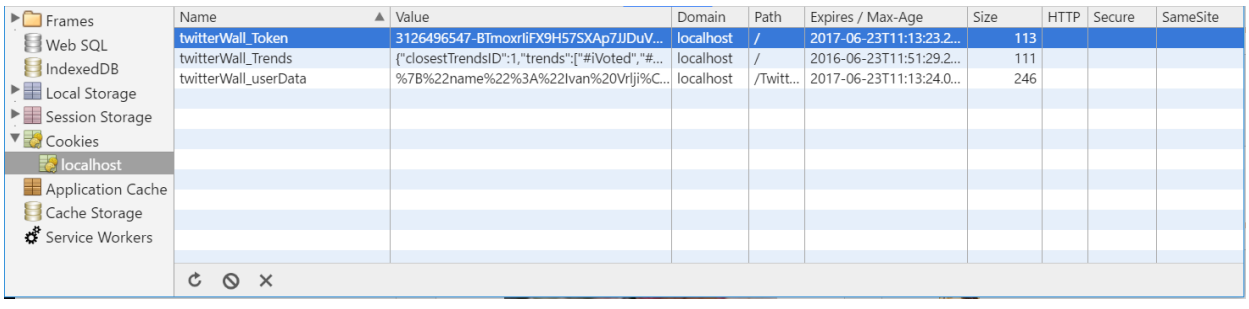
Sl. 4.2 Stranica za prijavu

Jedina mogućnost koju aplikacija pruža u ovoj situaciji jest navesti korisnika na stranicu za autorizaciju kojoj korisnik pristupa pritiskom na gumb *Prijava*. Kada korisnik dođe na stranicu za prijavu, može točno vidjeti kojim resursima daje pravo pristupa ukoliko uspješno izvrši prijavu. Ova stranica prikazana je na slici 4.3.



Sl. 4.3 Twitterova autorizacijska stranica

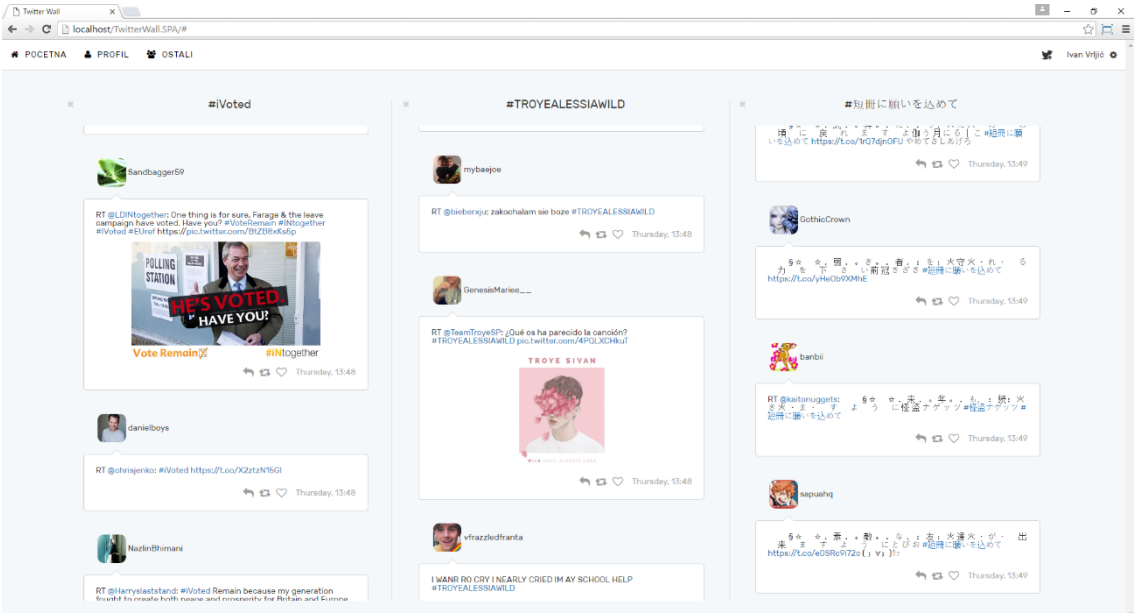
Očekivano ponašanje nakon prijave jest da se stvori kolačić s korisnikovim podatcima uz kojeg se sprema i dobiveni pristupni token. Nakon ovoga korisnika se preusmjerava na početnu stranicu. Na slici 4.4 prikazano je novo stanje kolačića. Vidljivo je kako su korisnički podatci i token uspješno spremljeni. U aplikaciji je predviđeno da se ovi podatci čuvaju jednu godinu u kolačiću što je postignuto postavljanjem datuma isteka za 365 dana, što se može zaključiti iz datuma na slici 4.4. Budući da se korisnika usmjerilo na početnu stranicu, u popisu kolačića pojavio se još jedan kolačić koji sadrži trenutno najpopularnije teme na Twitteru – twitterWall_Trends. Ovaj podatak sprema se na ovaj način zbog umanjenja broja odlazaka po tweetove.



Sl. 4.4 Stanje kolačića nakon prijave

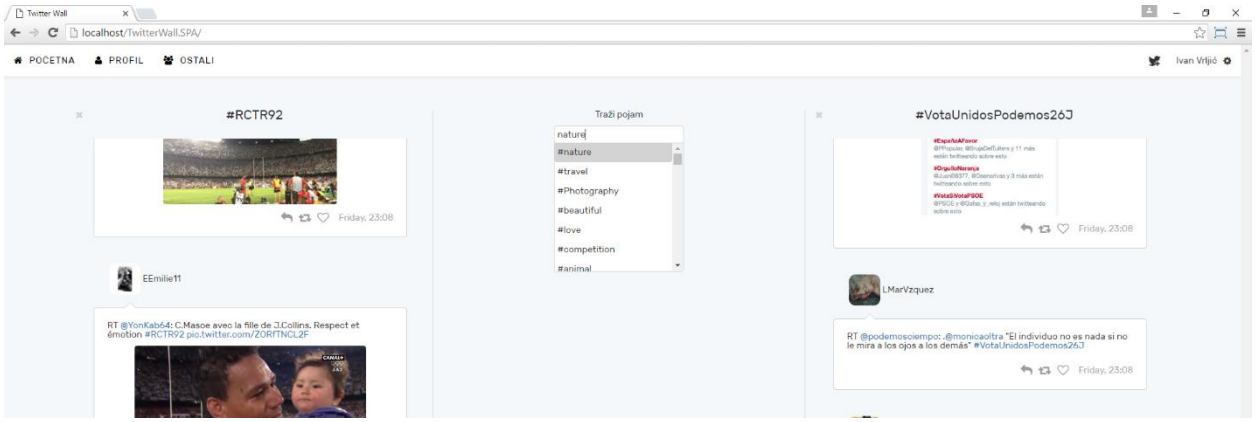
4.2 Početna stranica

Dolaskom na početnu stranicu korisniku se prikazuju tri paralelne liste tweetova s najpopularnijim temama na Twitteru. Prikaz listi nalazi se na slici 4.5. Na ovoj slici vidljiv je manji nedostatak u aplikaciji. Za korisnike koji nemaju definirano jezično područje na Twitteru (slučaj za trenutnog korisnika) podatci se uzimaju globalno. Posljedica toga je mogućnost uzimanja podataka koji su na jezicima koji ne koriste latinski alfabet, u ovom slučaju je to bio japanski. Rezultati su najčešće na engleskom jeziku, ali rezultat ovisi o trenutnim trendovima na Twitteru.



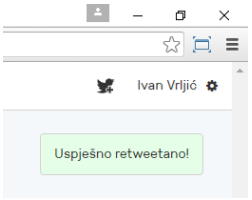
Sl. 4.5 Početna stranica aplikacije

Svaka od ove tri liste automatski se pomiče prema dolje. Kada korisnik dođe pokazivačem miša preko jedne od njih, ta lista se zaustavlja, a druge se nastavljaju pomicati prema dolje. Predviđeno je da se dolaskom na dno liste ode po nove tweetove koji su vezani za temu liste koja je došla do dna. Dohvatom podataka lista se pomiče na vrh i kreće ponovno pomicanje prema dolje. Testiranje je provedeno na način da se kroz vrijeme promatralo ponašanje aplikacije. Dolaskom na dno svaka od listi je uspješno zatražila nove podatke i uvela ih. Za to vrijeme ostale liste su se ponašale kao i kada je lista koja odlazi po podatke imala podatke. Zaključeno je da sve funkcionira kako je i predviđeno. Ukoliko korisnika ne zanima neka od tema može ju zatvoriti klikom na znak „x“. Nakon klika pojavljuje se polje za pretraživanje, a klikom na rezultat pretraživanja otvara se odabrana tema.



Sl. 4.6 Traženje nove teme

Na svakom tweetu postoje kontrole u donjem desnom kutu. Pod kontrolama podrazumijevaju se gumbi za odgovor na tweet, dijeljenje tweeta (retweet) i označavanje omiljenim (favorite). Osim navedenih gumba tu se još nalazi i vrijeme objave. Za ispis datuma vremena korištena je biblioteka moment.js koja na temelju korisnikova preglednika određuje format datuma za ispis. Budući da je korišteni preglednik postavljen na engleski jezik, vidljivo je kako se dani objave zapisuju na engleskom jeziku. Pritiskom na opciju za retweet objavljuje se tweet s istim sadržajem u ime korisnika. Nakon što aplikacija uspješno podijeli tweet, korisnika se obavještava kratkom porukom u gornjem desnom uglu koja nestaje nakon nekoliko sekundi. Izgled poruke je na slici 4.8.

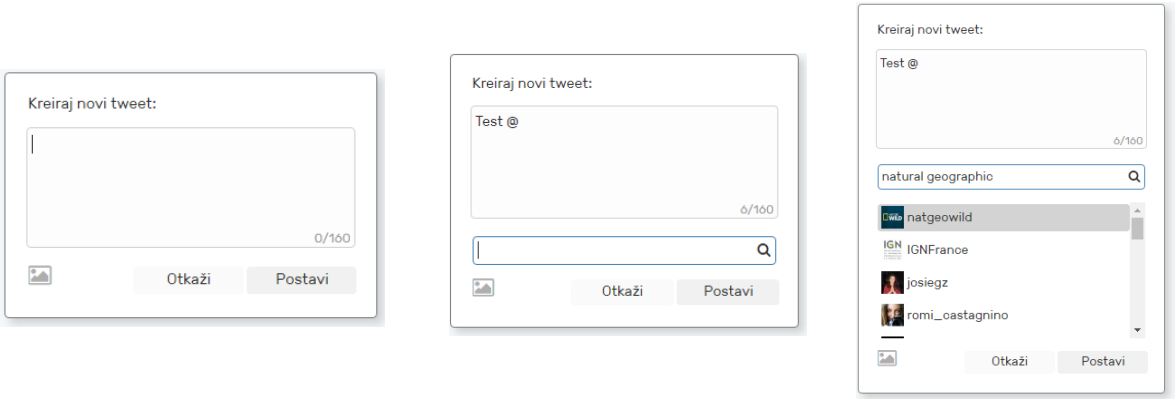


Sl. 4.8 Izgled poruke sa obavijestima

Opcija za oznaku objave omiljenom korisnika obavještava da je uspješno promijenio oznaku tako što mijenja svoj izgled. Ikona je u slučaju kada je objava označena omiljenom ispunjena, a prazna kada nije. Kako to korisnik vidi prikazano je na slici 4.9.

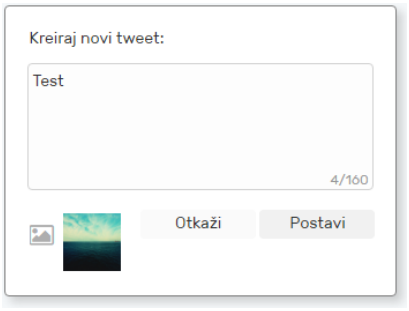
Sl. 4.9. Status oznake omiljene objave

Funkcije za odgovor na tweet i kreiranje novog tweeta imaju jednake funkcionalnosti, jedina razlika je u tome što se kod odgovora automatski u sadržaj tweeta dodaje ime osobe kojoj se šalje odgovor. Obje funkcionalnosti u pozadini izvršavaju isti kod, stoga je dovoljno obaviti provjeru samo jedne.



Sl. 4.10 Prozor za unos novog tweeta

Slikom 4.10 prikazan je modalni prozor za odgovor na tweet ili pak za stvaranje novog tweeta. U donjem desnom uglu prostora za unos prikazan je broj preostalih znakova koje korisnik može unijeti. Unošenjem nasumičnih vrijednosti testirano je brojanje i limitiranje unosa. Ustvrđeno je da broj znakova koji je korisnik unio odgovara onome koje aplikacija pokazuje. Drugi test je potvrdio ispravnost limitiranja broja unosa na 160 znakova. Predviđeno je da kada je zadnji upisani znak vrijednost „@“, tada se korisniku otvara polje za unos i na njega se automatski stavlja fokus. Ovo polje služi za unos imena osobe koju se želi unijeti u objavu. Upisom teksta odlazi se na API po listu osoba (desno na slici 4.10). Klikom na osobu na listi njeno korisničko ime se automatski dodaje u tekst tweeta i fokus se ponovno vraća na polje za unos teksta. Prije unosa imena u tekst valjalo je provjeravati duljinu istoga. Ukoliko je duljina ukupnog teksta na koji se nadoda korisničko ime prelazila vrijednost 160 znakova, korisničko ime se ne bi smjelo dodati u tekst. Osim specijalnog načina unosa, korisnička imena imaju i specijalan način brisanja. Pritiskom tipke za brisanje na bilo koji oblik @tekst ne briše se samo zadnji znak nego čitava vrijednost @tekst. Unosom različitih tekstualnih podataka ustanovljeno je ispravno ponašanje na temelju gore opisanih zahtjeva. Osim ovih mogućnosti, potrebno je bilo testirati i uspješnost objave napisanog tweeta na samoj društvenoj mreži. Kreiranje tweeta pruža mogućnost stvaranja istog sa i bez medijskog sadržaja. Medijski sadržaji su zapravo slike koje je moguće priložiti u tweet. Pritiskom na ikonu slike u prozoru za stvaranje otvara se prozor operativnog sustava za odabir. Po odabiru slike ona se prilaže u AJAX objekt te se korisniku prikazuje kao što je to na slici 4.11.

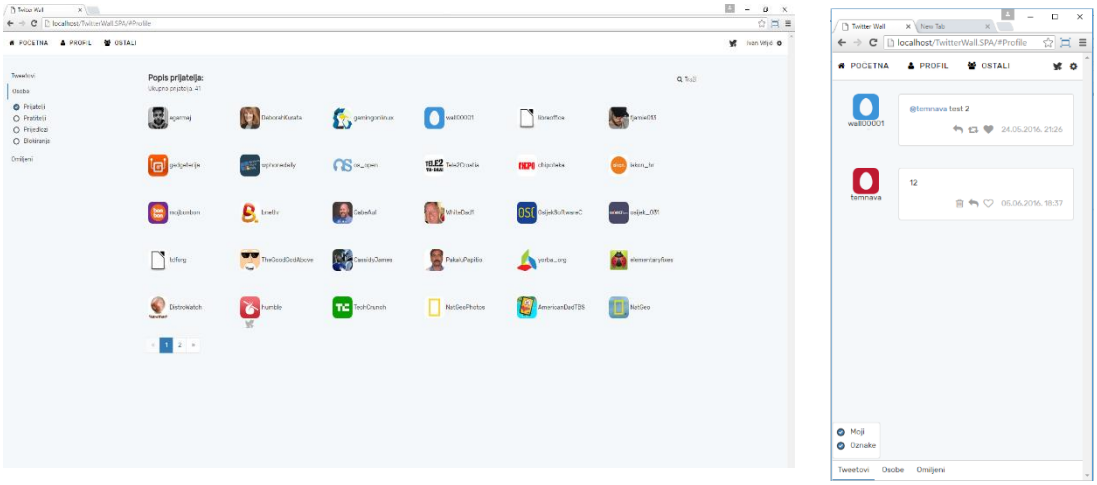


Sl. 4.11 Unos tweeta sa slikom

Nakon što je korisnik završio stvaranje tweeta (sa ili bez slike), pritiskom na gumb *Postavi* odlazi se na API i objavljuje tweet. Nakon nekoliko provedenih testova sa i bez medija ustanovljeno je kako ova funkcionalnost ispravno obavlja predviđeni zadatak.

4.3 Korisnikov profil

U ovom poglavlju opisat će se potkomponenta aplikacije koja prikazuje podatke profila trenutno prijavljenog korisnika. Izgled ove stranice vidljiv je na slici 4.11. Ova stranica podijeljena je u tri dijela: tweetovi, prijatelji i omiljeni sadržaj. Između ovih dijelova korisnik se prebacuje odabirom jednog od njih u izborniku s lijeve strane ekrana. Klikom na jedan od njih prikazuju se njegove potkategorije. Kada je neka potkategorija odabrana, pored njenog imena pojavljuje se bijela kvačica u plavome polju. Korisnik ima mogućnost mijenjati odabir u bilo kojoj kombinaciji. Kada odabere neku od kategorija, ona se pojavljuje u nastavku trenutno prikazanog HTML-a. Kada se ukloni oznaka za prikaz, vezani sadržaj nestaje iz prikaza. U opciji tweetovi korisnik može pregledavati vlastite tweetove i one u kojima je označen. Unutar opcije osobe nalaze se: prijatelji, pratitelji, prijedlozi i blokirane osobe. Prijatelji su osobe koje korisnik prati, a pratitelji osobe koje prate korisnika. Obje ove liste prikazuju se pomoću straničenja u slučaju da se na popisu nalazi više od 30 osoba. Klikom na opciju „Traži“ u gornjem desnom uglu korisniku se otvara polje za unos. Unosom vrijednosti u to polje korisniku se filtriraju prijatelji i prikazuju samo oni koji u imenu za prikaz ili korisničkom imenu sadržavaju unesenu vrijednost. Cijela stranica izrađena je na način da se skalira s veličinom ekrana. Na slici 4.12 vidljiva je razlika između izgleda na mobilnim uređajima i na velikim zaslonima. Bočna traka s izbornicima prebacuje se na dno, a pritiskom na ime kategorije u prozorčiću pokazuje se izbornik potkategorija.



Sl. 4.12 Profil korisnika (desktop i mobilna verzija)

Kod vlastitih tweetova umjesto opcije dijeljenja postavljena je opcija za brisanje. Klikom na tu opciju otvara se prozor koji korisnika pita za potvrdu brisanja. Ukoliko korisnik pristane na brisanje, odlazi se na API i briše tweet te ga se briše i iz popisa za prikaz.

4.4 Profili ostalih osoba

Klikom na sliku profila osobe koja je objavila tweet odlazi se na njihov profil. Osim ovoga načina, osobe je moguće pronaći na način da se ode na posljednju potkomponentu aplikacije i ondje pretraži osoba. Izgled ove potkomponente prikazan je slikom 4.13.

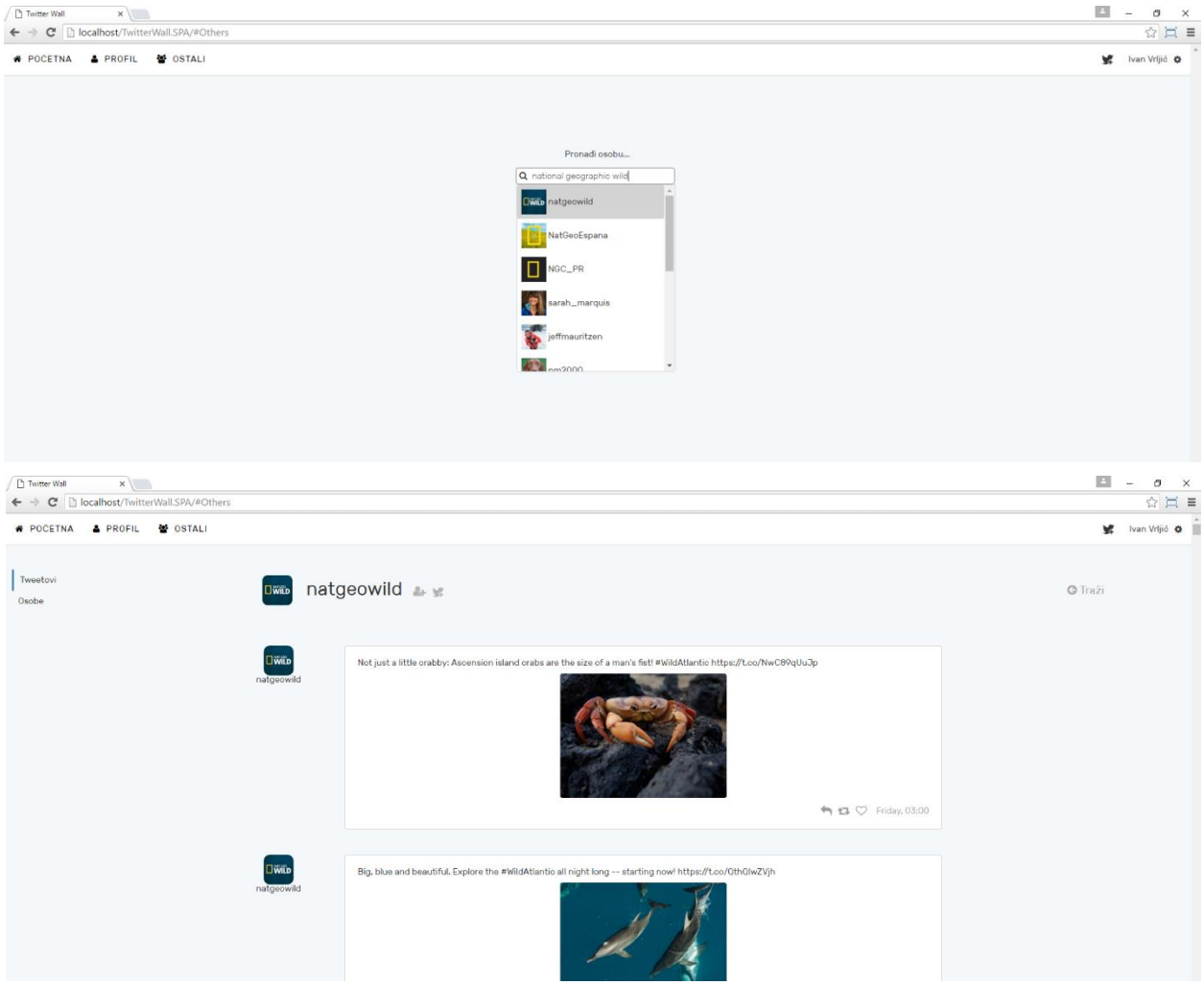
Dolaskom na ovu stranicu bez klika na nečiji profil otvara se gornji prikaz sa slike 4.13 gdje se korisniku omogućava pretraživanje osoba. Nakon što unese željeni upit, korisniku se daje rezultat povezanih osoba. Klikom na neku od osoba iz liste otvara se njen profil.

U ovoj potkomponenti sadržaj je također tematski odvojen. Prva grupa su tweetovi koje je korisnik objavio, a druga grupa su osobe. Unutar osoba korisnik može vidjeti s kime je osoba čiji profil gleda prijatelj i koga ta osoba prati.

Pokraj slike profila stoje kontrole za slanje tweeta korisniku i za promjenu statusa prijateljstva.

Klikom na ikonicu za stvaranje tweeta otvara se prozor sa slike 4.10. Klikom na ikonicu za promjenu statusa prijateljstva korisnik može dodati ili ukloniti osobu iz liste prijatelja. Promjenom stanja prijateljstva mijenja se i prikazana ikona. Ukoliko je osoba prijatelj korisnika tada ikona ima oznaku sa znakom „x“ što označava da će se klikom na ikonu prekinuti prijateljstvo, a ako nije u listi prijatelja tada ikona ima znak „+“ što znači da će se osoba dodati u

listu prijatelja. Testiranjem je ustanovljeno da se ikona ispravno ažurira kod svake promjene stanja prijateljstva.

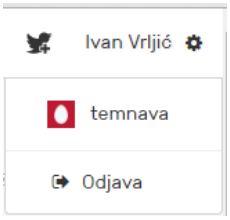


Sl. 4.13 Profili ostalih osoba

Ukoliko korisnik aplikacije želi pretražiti drugu osobu, može kliknuti na ikonu u gornjem desnom uglu (slika 4.13. dolje) gdje piše traži. Nakon odabira ove opcije, korisnika se ponovno vraća na polje za pretraživanje gdje može tražiti novu osobu. Važno je napomenuti kako se profil odabrane osobe čuva sve dok se ne odabere nova osoba ili klikne na opciju za ponovno traženje.

4.5 Odjava iz aplikacije

Opcija za odjavu korisnika iz aplikacije nalazi se u glavnom izborniku na vrhu stranice. Klikom na ime korisnika prikazuje se izbornik sa slike 4.14. Kada korisnik klikne na opciju odjava započinje brisanje svih korisnikovih kolačića. Na ovaj način osigurava se da aplikacija više ne zna tko je prijavljen.



Sl. 4.14 Izbornik za odjavu

Nakon brisanja kolačića, korisnika se ponovno usmjerava na stranicu za prijavu. Kod testiranja ustanovilo se da se aplikacija nakon odjave ponaša kao da je korisnik došao prvi puta. Situacija s kolačićima jednaka je kao i na slici 4.1 čime se zaključuje da odjava funkcionira ispravno.

5. ZAKLJUČAK

Kroz rad objašnjen je postupak izrade aplikacije za Twitter. Objašnjene su osnove funkcioniranja korištenih tehnologija kao što su Knockout.js, TypeScript, SCSS itd. Osim osnova funkcioniranja, objašnjen je i način na koji se pomoću definiranih tehnologija došlo do željenog vladanja aplikacije. Glavna prepreka u izvedbi aplikacije bilo je autorizirati korisnika i dohvatiti podatke s Twitterovog API sučelja. Razlog tome je kompleksan postupak dobivanja OAuth tokena za pristup nakon čega slijedi slanje složenih upita prema API-u koristeći specifičan način stvaranja zahtjeva i njegove enkripcije.

Dalje kroz rad opisan je način testiranja i prikaz mogućnosti aplikacije. Ustanovilo se kako sve implementirane mogućnosti funkcioniraju ispravno. Jedini nedostatak aplikacije jest da kada ju koristi korisnik koji nema definirano jezično područje, podatci se uzimaju gledajući globalno najpopularnije teme. U određenim situacijama ovakav pristup može dovesti do toga da se korisniku prikazuju rezultati koji nisu na latinskom alfabetu. Ovakve situacije nisu česte, ali i ne predstavljaju problem jer korisnik u bilo kojem trenutku može zamijeniti temu koja ga ne zanima.

Slična rješenja poput ove aplikacije već postoje, ali ta rješenja nude istovremeno praćenje isključivo jedne teme. Ova aplikacija pruža mogućnost praćenja tri teme istovremeno. Osim praćenja najzanimljivijih tema, korisniku se daje mogućnost istraživanja profila drugih osoba, praćenja stanja vlastitog profila te osnovne kontrole nad tweetovima (brisanje, dijeljenje…). Aplikacija također pruža i mogućnost objave novih tweetova sa i bez medijskog sadržaja.

LITERATURA

[1] Mike Wasson, **ASP.NET - Single-Page Applications: Build Modern, Responsive Web**, Studeni 2013., (https://msdn.microsoft.com/en-us/magazine/dn463786.aspx), 9.06.2016.

[2] **Learn REST**, What is a REST? (http://rest.elkstein.org/2008/02/what-is-rest.html), 9.06.2016.

[3] J. Kurtz, B.Wortman **ASP.NET Web API 2, Building a REST service from start to finish**, Apress, SAD, 2013.

[4] Mike Wasson, **Enabling Cross-Origin Requests in ASP.NET Web API 2**, 2014. (http://www.asp.net/web-api/overview/security/enabling-cross-origin-requests-in-web-api), 13.6.2016.

[5] **Sass (Syntactically Awesome StyleSheets) Documentation (**http://sass-lang.com/documentation/file.SASS_REFERENCE.html), 13.6.2016.

[6] Basarat Ali Syed, **TypeScript Deep Dive**, GitBook, 2016.

[7] Pr Ryan Hodson: **Knockout.js Succinctly**, Technology Resource Portal, 2015.

[8] Rob Daigneau, **Dana Transfer Objects**, (www.servicedesignpatterns.com/requestandresponsemanagement/datatransferobject), 14.6.2016.

[9] Addy Osmani, **Learning JavaScript Design Patterns**, O'Reilly Media, 2015. v1.6.2

[10] Roy Thomas Fielding, **Architectural Styles and the Design of Network-based Software Architectures,** University Of California, Irvine 2000.

[11] Edmond Woychowsky, **Creating Web Pages with AJAX**, Pearson Education, 2007.

[12] **Knockout.js Documentation**, Click Event (http://knockoutjs.com/documentation/click-binding.html), 17.6.2016.

[13] Mike Wasson, **Create a REST API with Attribute Routing in ASP.NET Web API 2** (www.asp.net/web-api/overview/web-api-routing-and-actions/create-a-rest-api-with-attribute-routing), 18.6.2016.

[14] **The OAuth 1.0 Authorization Framework**, E. Hammer-Lahav, Ed., IETF, 2010.

[15] **HMAC: Keyed-Hashing for Message Authentication**, H. Krawczyk, M. Bellare, R. Canetti,1997.

SAŽETAK

U ovome radu obrađena je jednostranična web aplikacija za Twitter. Glavnina aplikacije pisana je TypeScript i C# programskim jezikom. Tehnologije na kojima se temelji izrađena aplikacija su AJAX i Knockout.js. Kroz rad detaljnije je pojašnjen postupak primjene navedenih tehnologija kako bi se postiglo željeno vladanje aplikacije.

Stranica omogućava korisniku pregled trenutno najpopularnijih tema na Twitter društvenoj mreži. Od prve tri najpopularnije teme korisniku se stvaraju tri liste s tweetovima koje se automatski spuštaju prema dnu. Dolaskom na dno učitavaju se novi tweetovi s istom temom. Na svakom tweetu aplikacija korisniku omogućava dijeljenje tweeta, stvaranje odgovora i označavanje omiljenim. Osim navedenih funkcionalnosti, korisnik ima uvid u svoj profil i profil drugih osoba gdje može vidjeti prijatelje, pratitelje, objavljene tweetove itd.

Ključne riječi: Twitter, API, AJAX, SPA, autorizacija

ABSTRACT

Twitter Wall

This paper presents single page web application for Twitter. Majority of this application is written in TypeScript and C# programming language. Main technologies on which application is built upon are AJAX and Knockout.js. Detailed description of applying mentioned technology so that application behaves as desired is given throughout the paper.

This web page gives user an ability to look through the most popular themes on Twitter social network. Three most popular themes are shown in three parallel feeds that are being automatically scrolled to the bottom. When a feed scrolling reaches the bottom, new tweets with the same theme are being retrieved. On each tweet there are options to share, reply and favorite. There is also functionality for user to overview its own profile and other peoples profiles where he can see his friends, followers, shared tweets etc.

Key words: Twitter, API, AJAX, SPA, authorisation

ŽIVOTOPIS

Ivan Vrljić rođen je 3. srpnja 1992. godine u Starim Mikanovcima. Osnovno obrazovanje započinje upisom u Osnovnu školu Matije Antuna Reljkovića u Cerni 1999. godine. Završetkom osnovne škole 2007. godine nastavlja obrazovanje upisom u Tehničku školu Ruđera Boškovića u Vinkovcima. 2011. godine završava srednju školu s diplomom smjera tehničar za mehatroniku. Iste godine upisuje se na preddiplomski studij računarstva u Osijeku. Ovaj studij završava 2014. godine čime postaje prvostupnik računarstva. Završetkom ovog studija upisuje diplomski studij procesnog računarstva.

Potpis: Ivan Vrljić

PRILOZI

- MS Visual Studio projekt