Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/03BE15Bilbija

IMPLEMENTACIJA *SPOTIFY* VEB KLIJENTA U *REACT* I *DJANGO* RADNOM OKRUŽENJU

IMPLEMENTATION OF A SPOTIFY WEB CLIENT IN REACT AND DJANGO FRAMEWORK

Žana Bilbija, *Fakultet tehničkih nauka, Novi Sad*

Oblast – RAČUNARSTVO I AUTOMATIKA

Kratak sadržaj – *Prezentovan je kratak pregled alata sa Spotify Developer portala sa ciljem boljeg upoznavanja sa istim. Osnove integracije alata sa softverskim rešenjima su predstavljene kroz primer implementacije veb klijenta.*

Ključne reči: *Spotify, React, Django Rest, OAuth*

Abstract – *Presents a short overview of tools located on the Spotify Developer portal, with the goal of an introduction to the aforementioned tools. The basics of integrating these tools into future software solutions are presented through an implementation of a web client.*

Keywords: *Spotify, React, Django Rest, OAuth*

1. UVOD

Kako se razvija digitalni svet i kako se sve više širi domen interneta, povećava se broj servisa pristupačnih na internetu. Prateći povećanje broja servisa, takođe se povećava i obogaćuju tipovi usluga koji dotični servisi nude korisnicima. Jedna od popularnih usluga jeste mogućnost reprodukcije muzike.

Postoji značajan broj sajtova, veb servisa i veb aplikacija koje pružaju upravo tu uslugu, ali ovaj rad se koncentriše na jednu od njih – **Spotify**. Osim usluge reprodukcije muzičkih fajlova, ovaj veb servis nudi i mogućnosti rukovanje putem izdvajanja muzičkih fajlova u svoje arhive, ili pak grupisanja muzičkih fajlova u playliste ili pravljenja imitacije radio stanice, gde sam veb servis dopunjuje playlistu sa sličnim pesmama [1].

Količina usluga za rukovanje muzičkim fajlovima, velika pristupačnost veb servisa kao i postojanje alata za integraciju Spotify-a u druga softverska rešenja su razlozi radi čega je ovaj veb servis izabran kao tema ovog rada

2. REACTJS BIBLIOTEKA

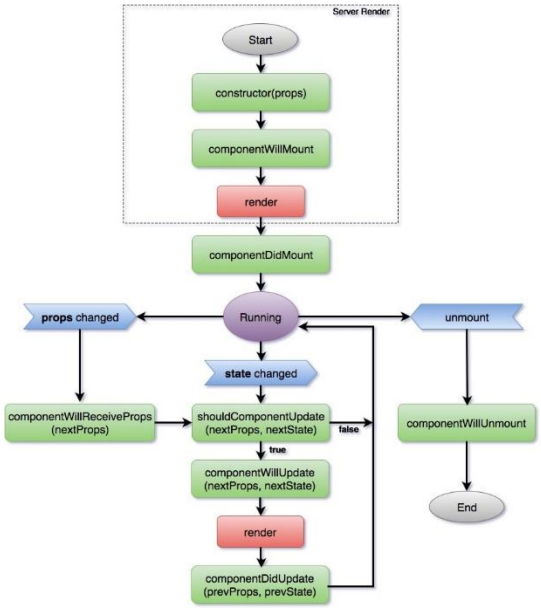
React jeste javno dostupna Javascript biblioteka namenjena implementaciji korisničkih interfejsa [2]. Razvijena je od strane kompanije *Facebook* [3]. Inicijalno je React razvijen kao alternativa XHP-u, radno okruženje za PHP. zasnovano na komponentama [4], prvenstveno namenjeno smanjivanju broja *XSS* napada [3, 5].

Osnovna strukturna jedinica u React-u jeste komponenta [6]. Komponente omogućuju da se korisnički interfejs segmentiše u nezavisne, relativno nezavisne delove. Kao takve, otvara se prilika da se već implementirane komponente ponovo iskoriste, te se izbegava dupliranje koda.

Komponente je moguće definisati u vidu funkcija ili kao klase koje nasleđuju *React.Component* klasu, i svakoj komponenti je moguće proslediti podatke u vidu *props* objekta. Bitna napomena kod props jeste da su oni objekti iz kojih komponenta samo može da čita vrednosti. Znači komponenta ne sme da menja sopstvene props-ove. Dakle može se posmatrati da su React komponente jedan vid tzv. čistih funkcija u odnosu na svoje props.

Da bi se znalo kada treba koju komponentu ponovo prikazati, mora se znati kada je komponenta bila izmenjena u odnosu na njeno ranije stanje. Stoga se uvodi stanje u komponentu. Stanje predstavlja objekat koji sadrži sve podatke o kojima komponenta mora da vodi računa.

Tek kada se stanje u komponenti izmeni, onda React ponovo prikazuje tu komponentu tako što samo njen deo unutar DOM stabla osvežava sa novim podacima koji su zabeleženi u njenom objektu stanja. Takođe metode životnog ciklusa komponente u kojima se može ažurirati stanje komponente.



Slika 1. *Životni ciklus React komponente i metode koje se pozivaju [7]*

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Aleksandar Kupusinac, vanr. prof.

JSX poznat i kao *JavaScript eXtension* jeste dodatak na sintaksu Javascripta. Sam oblik JSX-a podseca na XML sintaksu, i nudi alternativni, koncizniji način pisanja prikaza komponenti u kontrastu čistog Javascript pristupa. Osim kao alternativa strukturiranju komponenti unutar *render* funkcije, JSX omogućava i umetanje promenljivih u sam HTML kod. Pored promenljivih, JSX prihvata i pozive funkcija unutar vitičaste zagrade.

3. DJANGO REST RADNO OKRUŽENJE

Django REST je implementiran da podrži MVC dizajn šablon. Model može biti jednostavan kao već iskonfigurisana *SQLite* fajl baza podataka, ili konkretna relaciona baza podataka ili pak *NoSQL* baza podataka. Kontroler prati koncepte REST komunikacije, dok *View* varira od osnovne HTML stranice do eksternog veb klijenta poput React i sl. Da bi se ostvarila komunikacija sa Django REST-om, neophodno je definisati URL putanje koje će osluškivati unutar *urls.py* fajlu, kao i definisati koji kontroleri (preciznije, same metode u kontrolerima) će odreagovati kada se pošalje zahtev na zadatu adresu.

U definiciji samih funkcija se ne navode putanje koje osluškuju, ali se mogu navesti tipovi zahteva koji funkcije očekuju, putem anotacija. Serijalizacija i deserijalizacija podataka se vrši pravljenjem posebnih klasa za modele koje je potrebno pripremiti za slanje, te i za svaki atribut unutar klase je potrebno naznačiti kako se pretvaraju podaci. Moguće je precizirati da li je u pitanju serijalizacija jednog objekta ili niza objekata putem atributa pri pravljenju objekta klase serijalizatora.

4. SPOTIFY WEB API

Određeni podaci koje WEB API vraća se mogu tretirati kao podaci lične prirode, te je neophodno registrovati se na Spotify Developer sajtu da bi se održala bezbednost dotičnih podataka. Prvi korak u procesu registracije jeste da korisnik ili napravi Spotify nalog ili se uloguje u isti.[8] Dalje nakon prijave preko Spotify naloga, neophodno je registrovati konkretno softversko rešenje. Rezultat registracije jeste da Spotify zna koje konkretno softversko rešenje pristupa WEB API-ju, putem jedinstvenog identifikatora. Ovim se ograničava pristup i predstavlja dodatnu meru bezbednosti za korisničke podatke.

Osim registracije softverskog rešenja na sajtu, neophodno je i dobijene kredencijale podesiti u sklopu softverskog rešenja. Potrebno je navesti dobijeni identifikator, tajni ključ koji se generiše po registraciji kao i URL na koji će se slati token za autorizaciju zahteva [8].

4.1. Tipovi autorizacije

Kad je u pitanju autorizacija softverskog rešenja, postoje dva načina. Prvi podrazumeva da se rešenje autorizuje pristup Spotify platformu što podrazumeva pristup API-ju i drugim resursima za razvoj i integraciju. Drugi je pak autorizacija putem korisničkih kredencijala, gde se korisnički kredencijali šalju pri slanju zahteva ka API-ju, i zauzvrat se dobija token na ograničen vremenski period [9]. Svaki vraćeni token ima određen domen pristupa, a postoje i krajnje putanje API-ja koje ne zahtevaju token za pristup.

Postoje tri toka zahtevanja autorizacije [9]:

- Korisnička autorizacija koja se osvežava - podrazumeva da krajnji korisnik dopusti softverskom rešenju da pristupa podacima, a kao rezultat se dobija token za pristup koji može da se osveži. Razmena obuhvata i slanje tajnog ključa, te se mora voditi računa o tipu komunikacije.
- Privremena autorizacija korisnika - namenjen klijentskim aplikacijama implementiranim u JavaScript-u i pokrenutim u korisnikovom veb pretraživaču. Ovaj pristup ne zahteva slanje tajnoj ključa, nego se korisnik usmerava ka posebnom servisu za korisničke naloge- Kao rezultat prijave na taj servis dobija se token za pristup. On je kratkog životnog veka, i nemoguć osvežiti.
- Aplikaciona autorizacija koja se osvežava - ovaj pristup se koristi kada se vrši autentifikacija između dva servera, i omogućava pristup krajnjim putanjama koji je rukovode korisničkim podacima. Token se dobija prijavom slanjem identifikator softverskog rešenja i tajni ključ.

```
//Authorization Code Flow
{
    "access_token": "NgCXRK...MzYjw",
    "token_type": "Bearer",
    "scope": "user-read-private user-read-email",
    "expires_in": 3600,
    "refresh_token": "NgAagA...Um_SHo"
}

//Client Credentials Flow
{
    "access_token": "NgCXRKc...MzYjw",
    "token_type": "bearer",
    "expires_in": 3600,
}
```

Slika 2 – *Primer tokena za pristup za Korisničku autorizaciju i Aplikacionu autorizaciju*

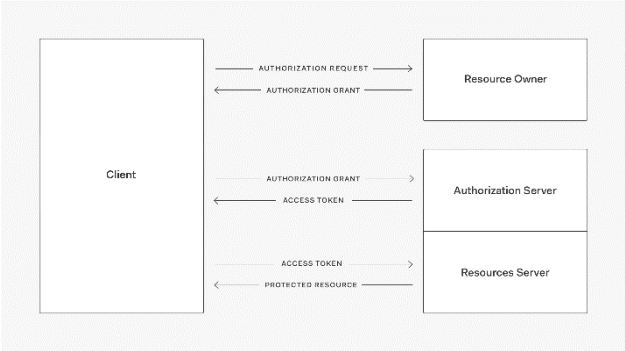
Postoji nekoliko domena koju su vezani za konkretan token pristupa, formirani na osnovu tipa podataka kojima rukovode. Svaka krajnja putanja API-ja spada u neki domen pristupa, a pojedini mogu pripadati nekoliko domena. Neki od domena su:

- Čitanje pesme koja se reprodukuje
- Ažuriranje liste sačuvanih pesama
- Ažuriranje stanja drugih Spotify klijenata
- Rukovanje drugim Spotify klijentima [10]

4.2. OAuth

OAuth je specifikacija koja definiše protokol za delegaciju, i korisno za prikazivanje odluke o autorizaciji u mreži veb-orijentisanih aplikacija, servisa i API-ja. Najčešće je korišćen pri autentifikaciji samih korisnika, ali sam OAuth nije protokol za autentifikaciju. Zapravo OAuth se koristi unutar protokola za autentifikaciju, i njene komponente se pozivaju iz konkretnog protokola za autentifikaciju Konkretno OAuth se razvio kao način da korisnici interneta dozvole sajtovima da pristupe njihovim

informacijama na drugim sajtovima bez da im korisnici daju svoje kredencijale Aktuelna verzija OAuth protokola je OAuth 2.0 koji definiše tok autorizacije za veb aplikacije, desktop aplikacije, mobilne telefone i pametne uređaje.



Slika 3 – *Tok OAuth autorizacije*

4.3. Analiza podataka

Podaci koje vraćaju krajnje putanje Spotify WEB API-ja su organizovani u strukturu koja prati definisani objektni model [11]. Dotični objekti se vraćaju u obliku JSON objekata. Putanje su grupisani po modelu objekta koji vraćaju, a pojedine vraćaju pojednostavljene verzije objekata. Pojednostavljene verzije objekata ne poseduju sve atribute, nego samo one neophodne za identifikaciju, one koje nose osnovne informacije i link ka putanji gde se nalaze potpuni podaci o objektu. Ovim putem se smanjuje veličina odgovora koji se vraća tako što se izbegava ugnježdavanje velikih objekata unutar podjednako ili pak većih objekata.

Ukoliko se vraća kolekcija objekata, i gde broj konkretnih objekata varira, kolekcija objekata se dodatno smešta unutar objekta podeljenog na stranice. Time se olakšava dobavljanje ostalog niza podataka u kolekciji, i smanjuje veličina odgovora, te se ne zatrpava komunikacija. Ovo je naročito važno kod softverskih rešenja sa tokenima za pristup kratkog životnog ciklusa. Objekti podeljeni na stranice imaju atribute koji sadrže URL putanje do krajnjih putanja gde se nalaze potpuni oblici traženog tipa objekta.

5. IMPLEMENTACIJA

Softversko rešenje koje je tema ovog rada se sastoji iz dva dela. Prvi deo jeste server koji je napisan u *Django REST* radnom okruženju i komunicira kako sa veb klijentom, tako i sa API-jem posredstvom biblioteke *Spotipy*. Drugi deo je sam veb klijent implementiran pomoću *ReactJS* biblioteke kao i pojedinih dodatnih biblioteka za izgled samog veb klijenta i dodatnu biblioteku za razrešenje putanja u klijentu.

5.1. *Spotipy* biblioteka

Spotipy je biblioteka napisana u *Python* programskom jeziku i koja se može integrisati u softversko rešenje bez dodatne konfiguracije [12]. Koristeći *Spotipy* biblioteku, nudi se mogućnost pristupa svim krajnjim putanjama Spotify WEB API-ja. Takođe nudi funkcionalnosti putem kojih se izvršava autorizacija softverskog rešenja u koje je integrisana. Da bi se uopšte započela komunikacije, neophodno je instancirati *Spotify* objekat, definisan unutar samog *spotipy* modula. Svi tipovi zahteva se prosleđuju posredstvom Spotify objekta, pozivajući adekvatne metode. Metode korespondiraju tipu zahteva. Biblioteka podržava dva vida autorizacije: Korisnička autorizacija koja se osvežava i Aplikaciona autorizacija koja se osvežava. Da bi se koristila Aplikaciona autorizacija, potrebno je koristiti klasu *SpotifyClientCredentials* iz modula *oauth2*, i objekat te klase se navodi kao parametar pri instanciranju objekta *Spotify* klase.

Ukoliko se pak ide putem Korisničke autorizacije (kao što je slučaj u ovom radu) onda je neophodno navesti klijentski identifikator, tajni ključ i URL za preusmeravanje.

Za slanje zahteva za vid Aplikacione autorizacije poziva se posebna metoda *prompt_for_user_token* koja pored nabrojanih parametara očekuje i parametar za domen za koji se zahteva autorizacija. Generisani token se prosleđuje na URL koji je naveden za preusmeravanje i kopira u terminal da bi biblioteka mogla učitati i privremeno arhivirati.

Klasa *SpotifyOAuth* iz modula *oauth2* vodi računa o isteku tokena, i ima sačuvanu putanju do Spotify-jevog servisa za autorizaciju. Potom se dobijeni token navodi pri instanciranju objekta klase Spotify.

```
import spotipy
import spotipy.util as util

scope = 'user-library-read'
util.prompt_for_user_token(username,scope,client_id='clientID',
  client_secret='clientSecret',redirect_uri='app-redirect-url')
if token:
    sp = spotipy.Spotify(auth=token)
    results = sp.current_user_saved_tracks()
    return results
```

Slika 4 – *Poziv funkcije za slanje zahteva za autorizaciju*

5.2. Arhitektura rešenja

Kao što je bilo spomenuto softversko rešenje se sastoji iz servera implementiranog u Django REST radnom okruženju i veb kijenta pisanog u ReactJS biblioteci. Arhitektura servera je grupisana vertikalno po tipu korisničkog zahteva koji ispunjavaju, tj. po domenu funkcionalnosti koje obuhvata. Veb klijent je organizovan na osnovu komponenti od koji se sačinjava, te su i one dalje grupisane na osnovu toga koja komponenta je enkapsulirana drugom. Kako se svi podaci dobavljaju iz Spotify-jevom API-ja, ne postoji konkretna baza podataka za skladištenje podataka, izuzev generisanja Microsoft Word dokumenta sa spiskom korisnikovih sačuvanih pesama.

Naravno ovo ne negira potrebu za sopstvenim modelima, štaviše veličina pojedinih podataka te i međusobna povezanost istih nameće obradu podataka na modele koji sažeto enkapsuliraju potrebne podatke shodno korisničkim zahtevima.

Parsiranje JSON odgovora sa API-ja u odgovarajuće klase modela se vrše u servisima. U servisima su takođe raspoređeni pozivi ka API-ju, te i provera da li je već dobijeni token za pristup istekao ili ne, tako što se porede vreme kada je token dobijen i trenutno vreme. Maksimalni životni ciklus tokena je jedan sat.

Konekcioni podaci potrebni za API su skladišteni u JSON fajlu. Ovim se drže podaci na jednom mestu, te je pogodnije za buduće izmene u konfiguraciji.
Pri instanciranju servisa na serveru, učitavaju i dodeljuju se podaci neophodni za autorizaciju zahteva ka API-ju. funkcionalnost arhiviranja spiska svih korisnikovih sačuvanih pesama u Microsoft Word dokument je implementirana pomoću modula za rukovanje dokumentima.
Obuhvata pravljenje novog dokumenta, podešavanje putanje za isti, dodavanje podataka u sam dokument koristeći metode za generisanje paragrafa i čuvanje istog na zadatoj putanji.
Pri pokretanju veb klijenta, neophodno je da se korisnik uloguje sa svojim Spotify nalogom. Uneseno korisničko ime se dalje prosleđuje server i koristi pri autorizaciji zahteva ka API-ju, sve do momenta kada se korisnik odjavi sa veb klijenta. Posle uspešne prijave korisnik ima opciju da izlista sve svoje sačuvane pesme, da izlista nekoliko poslednjih pesama koje su se puštale, izlista sve izvođače ili pak njihove albume. U donjem delu svakog od ovih prikaza se nalazi plejer, sa opcijama za kontrolu puštanja/pauziranja pesme.
Za slanje zahteva ka serveru se koristi funkcija *fetch()*. Potrebno je samo proslediti URL, tip zahteva, zaglavlja i podatke u JSON formatu URL putanje u veb klijentu su iskonfigurisane pomoću *React Router* dodatka [13]. Ova komponenta kao atribut prihvata URL putanju u obliku teksta, koja može da ima definisane parametre u sebi.
Da se naznači eksplicitno koja komponenta se prikazuje kada se preusmeri na određeni URL, u atribut *component* u *Route* komponenti se navodi naziv željene komponente

6. ZAKLJUČAK

U ovom radu opisana je implementacija softverskog rešenja veb klijenta za *Spotify*. Veb klijent je implementiran u vidu servera koji komunicira sa eksternim servisom, konkretno Spotify-jevim WEB API-jem, i u vidu veb klijenta sa kojim korisnik direktno stupa u interakciju u svom veb pretraživaču. U komunikaciji se razmenjuju JSON fajlovi.

U daljem razvoju bi se dobavljanje tokena za pristup prebacilo da sam server dobavi token umesto trenutno da se ručno mora kopirati u terminal sa URL-a na koji je preusmereno. Takođe bi se razmatrala mogućnost prelaska isključivo na veb klijenta u *ReactJS* biblioteci kako se već razvija dodatna biblioteka koja komunicira sa WEB API-je.

Ovo bi ubrzalo performanse jer svakako je brže komunicirati sa API-jem direktno nego preko servera kao posrednika. I na kraju ostaje mogućnost nabavke plaćenog naloga, čime se omogućuje pristup krajnjim putanjama za kontrolu drugih *Spotify* uređaja putem API-ja.

7. LITERATURA

[1] Spotify, https://www.spotify.com/int/about-us/contact/
[2] React – A Javascript library for building user interfaces, https://reactjs.org/
[3] JavaScript's History and How it Led To ReactJS, https://thenewstack.io/javascripts-history-and-how-it-led-to-reactjs/
[4] XHP, https://github.com/hhvm/xhp-lib
[5] XSS, OWASP, https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
[6] React Components and Props, https://reactjs.org/docs/components-and-props.html
[7] React Component Lifecycle Diagram, https://levelup.gitconnected.com/componentdidmakesense-react-lifecycle-explanation-393dcb19e459
[8] Spotify WEB API Quick Guide, https://developer.spotify.com/documentation/web-api/quick-start/
[9] Spotify WEB API Authorization Flows and Guide, https://developer.spotify.com/documentation/general/guides/authorization-guide/#authorizaton-code-flow
[10] Spotify WEB API Scopes, https://developer.spotify.com/documentation/general/guides/scopes/
[11] Spotify WEB API Object Model, https://developer.spotify.com/documentation/web-api/reference/object-model/
[12] Spotipy Documentation, https://spotipy.readthedocs.io/en/latest/
[13] React Router Documentation, https://reacttraining.com/react-router/web/guides/philosophy

Kratka biografija:

Žana Bilbija rođena je 26.08. 1994. godine u Novom Sadu, Republike Srbije. Završila je Gimnaziju Svetozar Marković 2013. godine, i iste godine je upisala Fakultet Tehničkih Nauka, isto u Novom Sadu, na odsek Računarstvo i automatika. Diplomirala je 2017. godine, završivši usmerenje Primenjene računarske nauke i informatika, i sa radom „Implementacija kontrole pristupa zasnovane na korisničkim ulogama u sistemu platnog prometa." Posle toga je upisala master akademske studije na programu Računarstvo i automatika, takođe na Fakultetu Tehničkih Nauka u Novom Sadu. Ispunila je sve obaveze i položila je sve ispite predviđene studijskim programom.