Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.4:621.395
DOI: https://doi.org/10.24867/12BE04Jovic

REACT KOMPONENTA ZA KONTROLU PRISTUPA BAZIRANU NA KORISNIČKIM ULOGAMA

REACT COMPONENT FOR ROLE-BASED ACCESS CONTROL

Vukašin Jović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu će biti opisan sistem za dinamičko generisanje komponenti na korisničkoj strani veb aplikacije implementiranoj korišćenjem React radnog okvira i RBAC kontrole pristupa.*

Ključne reči: *Kontrola pristupa, Veb aplikacija, React, RBAC*

Abstract – *This paper will present a system for dynamic component generation on the client side of a web application using React framework and RBAC.*

Keywords: *Access control, Web application, React, RBAC*

1. UVOD

Veb aplikacije često sadrže stranice i podatke koji se prikazuju samo određenim korisnicima. Svim podacima koji ne zahtevaju određeno pravo pristupa može se pristupiti direktno prilikom posećivanja određene stranice, kao što su na primer početna stranica, „o nama“ stranica i slično. Sa druge strane, stranice koje su specifične za pojedinačnog korisnika zahtevaju da se korisnik na neki način identifikuje sistemu, kako bi mu bile dostupne.

Ovaj rad bavi se konceptom kontrole pristupa u veb aplikacijama na osnovu uloga u sistemu. Prikazuje specifikaciju i implementaciju sistema za dinamičko generisanje odnosno aktivaciju/deaktivaciju komponenti na klijentskoj strani veb aplikacije koristeći korisničke role tj. uloge. Korisniku se u zavisnosti od uloge prikazuju određene stranice kao i akcije na stranicama koje može izvršiti.

2. OSNOVNI POJMOVI I DEFINICIJE

Za bolje razumevanje rada potrebno je prvo razumeti koncepte veb aplikacija, korisnikovom identifikovanju sistemu – autentifikaciji, kao i kontroli pristupa baziranoj na korisničkim ulogama.

2.1. Veb aplikacija

Veb aplikacije predstavljaju aplikacije kojima se pristupa putem interneta koristeći internet pretraživač (eng. web browser) i koje su razvijene koristeći jezike prihvatljive od strane samog pretraživača poput *HTML*-a (Hyper Text Markup Language), *JavaScript*-a i slično.

Da bi mogle da funkcionišu, oslanjaju se na internet pretraživače i mnoštvo poznatih aplikacija poput veb prodavnica, aukcija i mejl klijenata [1]. Pružaju mogućnost pokretanja aplikacije na različitim platformama i operativnim sistemima, pružaju istu verziju aplikacije za sve korisnike, ne zahtevaju prethodnu instalaciju što isključuje ograničenja memorijskog prostora i slično. Pojedine aplikacije su dinamičke, što znači da zahtevaju procesiranje na serverskoj strani, dok su druge statičke i ne zahtevaju nikakvo procesiranje već samo prikazuju statički sadržaj.

Veliki broj današnjih veb sajtova sadrži značajne količine koda na klijentskoj strani s obzirom da postoje kompleksni korisnički interfejsi i dosta podataka mora da se premešta i menja u toku vremena [2]. Sa druge strane, aplikaciju je potrebno napraviti brzom kako korisnik ne bi morao mnogo da čeka prilikom njenog učitavanja. Iz tog razloga potrebno je koristiti određenu arhitekturu koja pruža sve te mogućnosti kao i mogućnost jednostavnog dodavanja novih opcija, otklanjanja bagova i skaliranja cele aplikacije.

Za razvoj ovakve aplikacije mogu se koristiti razne javascript biblioteke i radni okviri, kao što je *React*. Radni okviri veb aplikacija su dizajnirani da pojednostave proces programiranja i promovišu kod postavljanjem biblioteka, različitih struktura, dokumentacije i smernica.

Za React se može reći da predstavlja *View* iz *MVC* šablona, međutim nije striktno određeno kako se mora koristiti. Kreira apstraktnu reprezentaciju pogleda i razdvaja delove pogleda na celine koje se nazivaju komponentama. Ove komponente sadrže i logiku za rukovanje pogledom ali i sam pogled i mogu biti nadograđenje i ponovo iskorišćene.

Komponenta može sadržati podatke koji se koriste kako bi se generisalo određeno stanje aplikacije. Stanje predstavlja objekat koji sadrži informacije koje pripadaju samo toj komponenti. Prilikom promene stanja, izvršavaju se algoritmi koji odlučuju da li je neophodno opet učitati komponentu i na koji način je to najefikasnije uraditi. S obzirom da je React JavaScript biblioteka, ona manipuliše *DOM*-om (*Document Object Model*).

Kako je ta operacija dosta skupa po pitanju resursa, *React* umesto pravim *DOM*-om manipuliše virtualnim. Promene i provera unutar virtuelnog *DOM*-a je brža i efikasnija što i sam *React* čini brzim i efikasnim [3, 4].

2.2. Autentifikacija

Autentifikacija je proces u kom se utvrđuje da je korisnik, odnosno entitet, ono što tvrdi da jeste. Autentifikacija kao takva predstavlja osnovni oblik pristupa veb aplikaciji. Da

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

ovaj mehanizam ne postoji, aplikacija bi sve korisnike tretirara kao goste (eng. guest) odnosno anonimne korisnike [5]. Postoje dva tipa autentifikacije: korisnička autentifikacija i entitetska autentifikacija. Korisnička identifikacija je najčešće predstavljena korsničkim nalogom, gde se korisnik sistemu identifikuje putem jedinstvenog identifikatora (uglavnom korisničkog imena ili elektronske pošte - *e-mail*) i lozinke.

Nakon unetih podataka, aplikacija proverava njihovu validnost te ukoliko je ona uspešno izvršena, sesijski token (eng. session token) se smešta u korisnikov pretraživač.

Na taj način korisnički pretraživač svaki put prilikom slanja novog zahteva šalje i sesijski token što predstavlja entitetsku identifikaciju. Imajući to u vidu, entitentska autentifikacija vrši se prilikom slanja svakog zahteva, dok se korisnička autentifikacija vrši samo jednom u toku sesije [6].

2.3. Kontrola pristupa

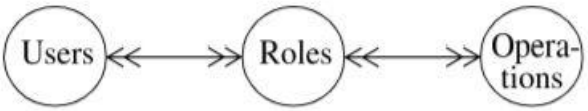
Kontrola pristupa i njeni mezanizmi čine ključni element pri projektovanju bezbednosti svake veb aplikacije. Podaci sa klijentske (eng. front-end) i sa serverske (eng. back-end) strane kao i sistemski resursi bi trebalo da budu zaštićeni od strane aplikacije. Kontrola pristupa bi trebalo da zaštiti podatke od neautorizovanog gledanja, menjanja i kopiranja ograničavanjem korisnikovih radnji, pristupa resursima kao i funkcijama koje manipulišu tim resursima [5].

Pojam koji se često zamenjuje kontrolom pristupa jeste pojam autorizacije. Ona predstavlja proveru odrovarajuće dozvole korisnika da pristupi određenom podatku ili izvrši određenu akciju, pod pretpostavkom da se korisnik prethodno uspešno autentifikovao. Autorizacija se zasniva na specifičnim pravilima liste kontrole pristupa koje unapred određuju administratori aplikacije ili vlasnici podataka.

Neke od čestih provera autorizacije jesu: ispitivanje članstva u određenoj grupi korisnika, pristustvo korisnika na listi odobrenih korisnika resursa ili posedovanje odgovarajućeg odobrenja. Svaki mezanizam kontrole pristupa koji se korisni za autorizaciju neposredno zavisi od efikasnih zaštićenih kontrola autentifikacije [6].

Kontrola pristupa zasnovana na ulogama (*Role-based access control – RBAC*) predstavlja kontrolu pristupa gde su odluke pristupa zasnovane na korisnikvim individualnim ulogama i odgovornostima unutar organizacije [7].

Uloga se pravilno posmatra kao semantička konstrukcija oko koje se formuliše politika kontrole pristupa. Određeni skup korisnika i dozvola koje okuplja jedna uloga je privremeni. Sa druge strane, uloga je stabilnija jer se aktivnosti ili funkcije organizacije ređe menjaju. Uloga može predstavljati kompetenciju za obavljanje određenih zadataka, a može predstavljati autoritet i odgovornost. Ima mogućnost da održava određene zadatke dužnosti koji se rotiraju kroz više korisnika. Uloge se razlikuju od grupa jer se tretiraju kao skup dozvola, a ne kao skup korisnika. One u stvari predstavljaju skup korisnika s jedne strane i skup dozvola sa druge, odnosno služe kao posrednik za spajanje ova dva skupa što je prikazano na slici 2 [7].



Slika 2. *Korisnici, uloge i dozvole [8]*

Dvostruke strelice prikazane na slici 2 ukazuju na vezu više prema više. To znači da jedan korisnik može biti u asocijaciji sa jednom ili više uloga, i isto tako jedna uloga može biti u asocijaciji sa više jednim ili više korisnika. Isti scenario se odvija i između operacija i uloga [8].

Kako bi se definisala uloga, potrebno je izvršiti analizu ciljeva i strukture organizacije što je obično povezano sa bezbednosnim normama. Koristeći *RBAC* metod kontrole pristupa administratori veb aplikacije imaju mogućnost određivanja operacija nad podacima, odnosno ko može da vrši koju akciju, kada, kojim redosledom i pod kojim relacionim okolnostima. Dakle operacije koje su povezane s ulogama ograničavaju članove uloge na određeni skup akcija. Na taj način korisnici sistema imaju mogućnost da pristupe samo onim informacijama koje su im potrebne kako bi izvršili svoj posao. Iako *RBAC* model ne promoviše ni jednu politiku zaštite, pokazalo se da podržava nekolicinu poznatih principa i politika koji su važni za bezbednost komercijalnih i državnih preduzeća koji rade sa osetljivim informacijama [8].

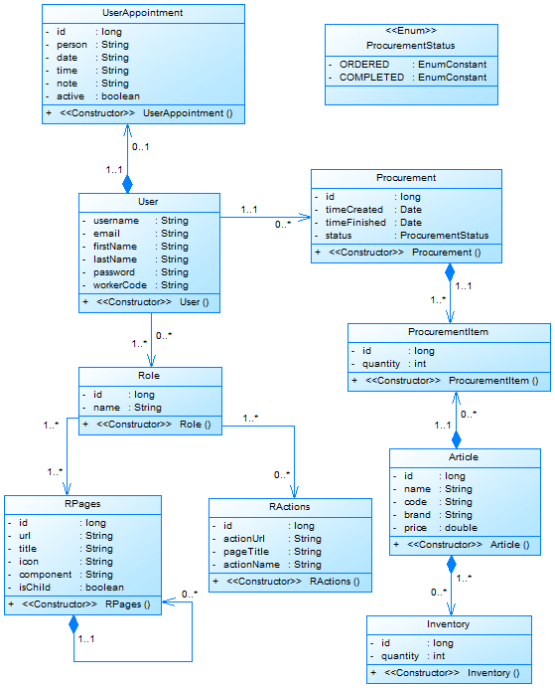
3. MODEL SISTEMA KONTROLE PRISTUPA

Prilikom kreiranja uloga sistema statički su im dodeljene odgovarajuće dozvole odnosno akcije. Kako se radi o front-end delu aplikacije, u pitanju su komponente (stranice) i različite akcije i pogledi na stranicama. Te stranice i akcije su sačuvane u bazi podataka kao liste objekata *RPages* i *RActions*, i u bazu se upisuju nakon što se takva komponenta implementira na klijentskoj strani. Dodavanjem korisnika u sistem dodeljuje mu se i odgovarajuća uloga, ili više njih. Korisniku se prikazuju samo komponente i akcije za koje mu je dozvoljen pristup, dok se ostale komponente ne generišu. To funkcioniše tako što se, nakon što se korisnik uspešno prijavi na sistem, sa back-end dela aplikacije na osnovu njegove uloge u vidu liste dobave sve stranice i akcije koje su mu dozvoljene. Postoji komponenta koja primi ovu listu te na osnovu nje generiše ostatak aplikacije. Na taj način korisnik ima pristup samo onim operacijama koje su dozvoljene u okviru njegove uloge te ne može vršiti neautorizovane operacije niti videti neautorizovani sadržaj.

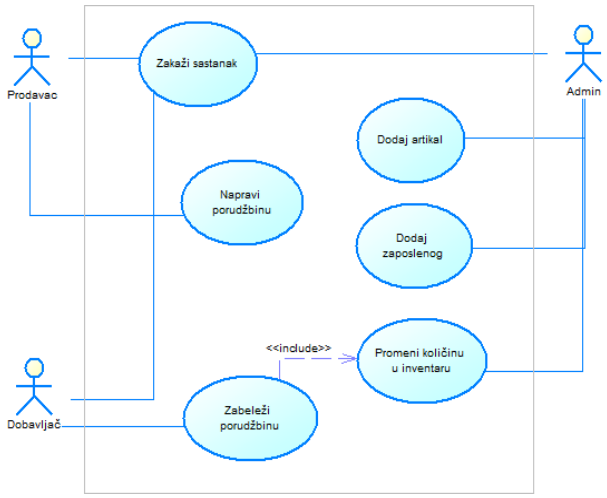
Sistem na kom je verifikovano korišćene kontrole pristupa posredstvom uloga, predstavlja veb aplikaciju magacina za skladištenje robe. Magacin sadrži skup artikala, tj. inventar, koji se trenutno nalazi u njemu i aplikacija služi kako bi se izvršila potražnja artikala koji nedostaju kao i izvršila evidencija artikala koji su dopremljeni. Na slici 3 prikazan je dijagram klasa sistema. Unutar organizacije magacina postoje 3 glavne vrste zaposlenih i to su:

- Prodavac – osoba koja potražuje robu
- Dobavljač – osoba koja evidentira robu koja je pristigla
- Admin – osoba koja dodaje nove zaposlene i ima kontrolu nad sistemom

Sistem podržava dodeljivanje više uloga jednom korisniku, s toga radnik koji ima ulogu menadžera, ima i ulogu prodavca i ulogu dobavljača. Na slici 4 prikazan je dijagram slučeva korišćenja za prethodno navedene tri uloge.



Slika 3. *Dijagram klasa sistema*



Slika 4. *Dijagram slučajeva korišćenja*

4. IMPLEMENTACIJA SISTEMA

U narednom odeljku biće predstavljena veb aplikacija za magacinsko poslovanje i način generisanja komponenti.

4.1. Implementacija aplikacije

Da bi korisnik mogao da pristupi stranicama i podacima, prvo je potrebno da se prijavi na sistem. Neprijavljeni korisnik ima pristup samo *Login* komponenti, i ukoliko pokuša da pristupi nekoj drugoj komponenti putem address bar-a prikazuje se *NotFound* komponenta.

Nakon što je korisnik uneo svoje kredencijalne u formu za prijavu, podaci se šalju back-end sistemu na proveru. Ukoliko su kredencijali tačni, server generiše *JSON Web Token* (*JWT*) koji se vraća klijentu kao odgovor i pomoću kog će se klijent identifikovati serveru ubuduće u toku sesije dok se korisnik ne odjavi sa sistema, ili dok token ne istekne. Ukoliko token istekne, korisnik će ponovo morati da se prijavi na sistem. Kada se korisnik uspešno autentifikuje, biva preusmeren na *PrivateRoutes* komponentu, prikazanu na listingu 1, koja generiše sve dozvoljene stranice i akcije u sistemu. Prilikom generisanja ove komponente, vrši se dodatna provera da li je korisnik prijavljen na sistem kako bi se još više osigurali pristup komponentama.

```
function PrivateRoutes() {
 const match = useRouteMatch('/app');
 const [allowedRoutes, setAllowedRoutes] = useState([]);
 const [hasRoutes, setHasRoutes] = useState(false);

 useEffect(() => {
  UserService.isUserLoggedIn().then(response => {
    let isit = response.data
    if(isit) {
     UserService.getUserRoutes().then(response => {
      if(response != null) {
       setAllowedRoutes(response.data)
       setHasRoutes(true);
      }
     })
    } else {
     return <Redirect to="/" />;
    }
   })
 }, [])

 const handleChild = (parent, route, url) => {
  const Component = Routes[route.component];
  return route.children.length > 0 ?
  route.children.map((child) => {
   const newUrl = url + child.url;
   return handleChild(route, child, newUrl)
  })
   :
  <Route
   exact
   key={route.url}
   path={`${match.path}${url}`}
  >
   <Component allowedActions={route.actions}/>
  </Route>
 }

 if({hasRoutes}) {
  return (<Fragment>
   <Navigation routes={allowedRoutes} path={match.path} />
   <Switch>
    {allowedRoutes.map((route) => {
     if(route.children.length > 0) {
      return (route.children.map( child => {
       var newUrl = route.url + child.url;
       return handleChild(route, child, newUrl)
      }))
     }
     const Component = Routes[route.component];
     return (
      <Route
       exact
       key={route.url}
       path={`${match.path}${route.url}`}
      >
       <Component allowedActions={route.actions}/>
      </Route>
     )
    })}
    <Route component={NotFound} />
   </Switch>
  </Fragment>
  )
 } else {
  return (
   <div style={{textAlign: 'center'}}>
    <h1 style={{marginTop: '15%'}}>Loading...</h1>
   </div>
  )
 }
}
```

Listing 1. *PrivateRoutes komponenta*

Pre samog prikaza *PrivateRoutes* komponente, šalje se zahtev *back-end* delu sistema kako bi se pribavile rute odnosno stranice kojima korisnik može pristupiti. Pored stranica, dobavljaju se i akcije koje korisnik može izvršiti. Akcije mogu predstavljati i delove stranice koje će se prikazati korisniku, kao i operacije koje kornisnik može izvršiti (npr. pravljenje nove porudžbine).

Kada na *front-end* kao odgovor stigne lista korisničkih ruta, prikazuju se komponente. Najpre se postavi navigaciona traka (eng. navbar) koja sadrži linkove pomoću kojih korisnik navigira između stranica. Prilikom generisanja komponente *Navigation*, prolazi se kroz svaku rutu i proverava da li je lista dece prazna. Ukoliko jeste, ruta se generiše kao link, dok ukoliko nije, ruta se generiše kao padajući meni (eng. dropdown menu). Nakon što se navigaciona traka generiše, generišu se komponente na koje linkovi navigacije vode.

One se generišu tako što se preuzme naziv komponente iz rute, nakon čega se preuzme komponenta sa istim nazivom koja se izvozi iz komponente *ComponentRoutes*. U ovoj komponenti nalaze se uvezene i potom izvezene sve komponente koje se prikazuju u sistemu kako bi se na osnovu naziva iz rute, mogla pronaći odgovarajuća komponenta sistema. Važan parametar prilikom kreiranja komponente jeste parametar *allowedActions* pomoću kojeg se komponenti koja se generiše prosleđuju akcije koje su dozvoljene korisniku.

Kada je generisanje navigacione trake i svih komponenti završeno, korisniku se prikazuje aplikacija i omogućeno mu je korišćenje. Ukoliko korisnik navigira na neku komponentu, tj. stranicu sistema, potrebno je prilikom prikazivanja proveriti za svaku akciju na toj stranici koja zahteva dozvolu da li je korisnik ima. To se radi tako što sto se uz deo koda koji zahteva proveru postavlja i uslov, koji ukoliko je tačan, prikazuje ackiju. Taj uslov se proverava pomoćnom funkcijom *isActionAllowed* sa listinga 2.

```
isActionAllowed = (actionName = "") => {


let numbb = this.props.allowedActions.map(function(a) { retu
rn a.actionUrl; }).indexOf(actionName);
  return (numbb >= 0) ? true : false;
 }
```

Listing 2. *Pomoćna funkcija isActionAllowed*

4.2. Dodavanje nove komponente ili dozvole u sistem

Kako bi se dodala nova komponenta u sistem, potrebno je nakon kreiranja same komponente uvesti je u komponentu *ComponentRoutes* a potom i dodati u listu izvezenih komponenti. Pri tome treba voditi računa da se komponenta izvozi sa istim nazivom kao što je ona nazvana u sistemu. Ovaj proces potrebno je uraditi na *front-end* strani sistema dok je na *back-end* strani potrebno dodati komponentu u bazu podataka. Na isti način se u bazu podataka dodaju i akcije koje se nalaze na novonastaloj komponenti.

Ukoliko se ulozi u sistemu dodaje dozvola za pristup određenoj komponenti ili akciji, potrebno je dodati željenu komponentu/akciju u listu već postojećih komponenti/akcija koje su u vezi sa ulogom.

5. ZAKLJUČAK

Tema ovog rada jeste kreiranje sistema za dinamičko generisanje komponenti na klijentskoj strani veb aplikacije, implementirane korišćenjem React radnog okvira. Predstavljen je sistem koji na osnovu kontrole pristupa zasnovanoj na ulogama (*RBAC*) vrši aktivaciju i deaktivaciju odgovarajućih komponenti sistema.

Objašnjeni su neki od osnovnih koncepata veb aplikacija, autentifikacije, kontrole pristupa, kao i kontrole pristupa zasnovanoj na ulogama. U trećem poglavlju prikazan je model celokupnog sistema, zajedno sa objašnjenjem rada sistema propraćenim dijagramom klasa. Sama implementacija prikazana je u poglavlju četiri. Sistem na kom je verifikovana kontrola pristupa zasnovama na ulogama predstavlja veb aplikaciju magacina gde korisnici sa odgovarajućom ulogom imaju dozvolu da pristupe različitim stranicama i obavljaju različite zadatke. Pored implementacije, prizakane su fotografije sistema za vreme korišćenja.

Dalji pravci razvoja i unapređenja veb aplikacije može biti implementacija *HTTPS*-a (*Hypertext Transfer Protocol Secure*) kako bi se osigurala sigurna komunikacija između klijenta i servera. Pored toga, moguće je sistem implementirati korišćenjem React Redux-a koji predstavlja mesto za čuvanje stanja unutar cele aplikacije, te bi se korisničke rute i dozvole mogle tu sačuvati, a ne prosleđivati među komponentama. Na kraju, moguće je implementirati funkcionalnost i grafički korisnički interfejs za promenu dozvola među ulogama, kojoj bi imao pristup admin sistema. Na taj način nakon kodiranja klijentskog i serverskog dela, odnosno dodavanje nove komponente u sistem, moguće je prepustiti adminu sistema da odredi koja uloga će imati dozvolu da pristupi novonastaloj komponenti, kao i da promeni prethodno postavljene veze uloga, komponenta i akcija.

6. LITERATURA

[1] Sabah Al-Fedaghi, *Developing Web Applications*, Computer Engineering Department - Kuwait University

[2] Marin Šili, Jakov Krolo, Goran Dela, *Security Vulnerabilities in Modern Web Browser Architecture*, Faculty of Electrical Engineering and Computing, University of Zagreb, 2010.

[3] Vipul A M, Prathamesh Sonpatki, *ReactJS by Example – Building Modern Web Applications with React*, Packt Publishing, 2016.

[4] Mark Piispanen, *Modern architecture for large web applications*, Bachelor's Thesis in Information Technology, 2017. [Online]. Доступно: https://jyx.jyu.fi/bitstream/handle/123456789/54129/URN%3aNBN%3afi%3ajyu-201705272524.pdf?sequence=1&isAllowed=y

[5] Muzafer Saračević, Muhedin Hadžić, *Upravljanje bezbednošću u cyber prostoru i mehanizmi zaštite veb apliakcija*, Septembar 2019.

[6] Stevan Džigurski, Bezbednost veb aplikacija, Novi Sad, 2003.

[7] Ravi S. Sandhu, *Role-based Access Control. Advances in Computers*, Laboratory for Information Security Technology, Academic Press 1998.

[8] David F. Ferraiolo, Janet A. Cugini, Richard Kuhn, *Role-Based Access Control (RBAC): Features and Motivations*, National Institute of Standards and Technology, 2003.

Kratka biografija:

Vukašin Jović rođen je u Novom Sadu 06.09.1996. godine. Osnovne akademske studije upisao je na Fakultetu Tehničkih nauka Univerziteta u Novom Sadu 2015. godine. Diplomirao je 2019. godine nakon čega upisuje master akademske studije na istom fakultetu.