Autorizacija korisnika web aplikacije temeljena na ulogama

Petković, Ante

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

DIPLOMSKI RAD

AUTORIZACIJA KORISNIKA WEB APLIKACIJE TEMELJENA NA ULOGAMA

Ante Petković

Split, rujan 2022.

Voljenoj supruzi i mom malom kotačiću sreće

Temeljna dokumentacijska kartica

Diplomski rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za Informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

AUTORIZACIJA KORISNIKA WEB APLIKACIJE TEMELJENA NA ULOGAMA

Ante Petković

SAŽETAK

Osnovne komponente procesa upravljanja identitetom i kontrole pristupa čine autorizacija i autentikacija. Osim utvrđivanja korisničkih prava na korištenje određenog resursa, sustav kontrole pristupa može definirati kada i kako se određeni resurs može koristiti. Uklapanjem autorizacije unutar sustava postiže se ograničavanje radnji na samo dozvoljene akcije, kontrola pristupa resursima s obzirom na korisničke uloge i privilegije i povećanje sigurnosti samog sustava.

Ključne riječi: autorizacija, autentikacija, kontrola pristupa, sigurnost

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 71 stranicu, 25 grafičkih prikaza, 5 tablica i 26 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: **Dr. sc. Goran Zaharija,** *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Ocjenjivači: **Dr. sc. Goran Zaharija,** *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Dr. sc. Saša Mladenović, *izvanredni profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Dr. sc. Divna Krpan, *viši predavač Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen: Rujan, 2022.

Basic documentation card

Thesis

University of Split
Faculty of Science
Department of Informatics
Ruđera Boškovića 33, 21000 Split, Croatia

ROLE-BASED AUTHORIZATION IN WEB APPLICATIONS

Ante Petković

ABSTRACT

Basic components of the identity management and access control process are authorization and authentication. Except determining user rights to use a particular resource, an access control system can define when and how a particular resource can be used. By integrating authorization within the system, limiting actions to only permitted actions, controlling access to resources regarding user roles and privileges, and increasing the security of the system itself is achieved.

Key words: authorization, authentication, access control, security

Thesis deposited in library of Faculty of science, University of Split

Thesis consists of: 71 pages, 25 figures, 5 tables and 26 references

Original language: Croatian

Mentor: **Goran Zaharija, Ph.D.** *Assistant Professor of Faculty of Science, University of Split*

Reviewers: **Goran Zaharija, Ph.D.** *Assistant Professor of Faculty of Science, University of Split*

Saša Mladenović, Ph.D. *Assistant Professor of Faculty of Science, University of Split*

Divna Krpan, Ph.D. *Senior Lecturer of Faculty of Science, University of Split*

Thesis accepted: September 2022.

IZJAVA

kojom izjavljujem s punom materijalnom i moralnom odgovornošću da sam diplomski rad s naslovom Autorizacija korisnika web aplikacije temeljena na ulogama izradio samostalno pod voditeljstvom doc. dr. sc., Goran Zaharija. U radu sam primijenio metodologiju znanstvenoistraživačkog rada i koristio literaturu koja je navedena na kraju diplomskog rada. Tuđe spoznaje, stavove, zaključke, teorije i zakonitosti koje sam izravno ili parafrazirajući naveo u diplomskom radu na uobičajen, standardan način citirao sam i povezao s fusnotama s korištenim bibliografskim jedinicama. Rad je pisan u duhu hrvatskog jezika.

Student

Ante Petković

SADRŽAJ

Uvod

Ubrzanim razvojem aplikacija i računalne tehnologije općenito svijet je uvidio važnost i potrebu za nekom vrstom sigurnosti. Sigurnost je vitalni dio svake web aplikacije i kao takva ne bi smjela biti naknadna misao prilikom implementacije. Autentikacija i autorizacija čine kritičnu komponentu upravljanja identitetom i kontrolom pristupa te kibernetičke sigurnosti općenito. Autentikacija je u osnovi proces provjere identiteta osobe. S druge strane, autorizacija je proces provjere da li je korisniku dopušten pristup određenom resursu.

Rad je podijeljen u pet osnovnih cjelina. U prvom poglavlju susrest ćemo se s pojmom sigurnosti i s nekoliko najpopularnijih modela za razvoj sigurnosnih sustava. U drugom poglavlju osvrnuti ćemo se na pojam autentikacije. Poglavlje vezano za autorizaciju predstavlja osnovnu tematiku i većinski sadržaj rada. Unutar spomenutog poglavlja navest ćemo i objasniti svaki od modela kontrole pristupa, njihove prednosti i nedostatke, a poseban osvrt dat će se na model kontrole pristupa temeljenog na uloga. Četvrto poglavlje, kompromitirana kontrola pristupa, predstavlja najveći sigurnosni rizik današnjih web aplikacija. S obzirom na to, pokazat ćemo najbolje prakse i načine za obranu integriteta aplikacije. Zadnje poglavlje poslužilo nam je da spomenute koncepte provedemo u djelo i predstavlja izradu web aplikacije temeljenu na ulogama.

1. Sigurnost

Sigurnost web aplikacija, također poznata kao *Web AppSec*, ideja je izgradnje web mjesta da funkcioniraju prema očekivanjima, čak i kada su napadnuta. Koncept uključuje zbirku sigurnosnih kontrola ugrađenih u web aplikaciju kako bi se zaštitila njezina imovina od potencijalno zlonamjernih agenata. Web aplikacije, kao i sav softver, neizbježno sadrže nedostatke. Neki od ovih nedostataka predstavljaju stvarne ranjivosti koje se mogu iskoristiti, unoseći rizike za organizacije. Sigurnost web aplikacije štiti od takvih nedostataka. Uključuje korištenje sigurnih razvojnih praksi i implementaciju sigurnosnih mjera tijekom životnog ciklusa razvoja softvera (engl. *Software Development Life Cycle*, skraćeno SDLC), osiguravajući da se nedostaci na razini dizajna i greške na razini implementacije rješavaju [1].

Sigurnost se u svojoj suštini oslanja na sljedeće elemente [2]:

- autentikacija – osigurava da je subjekt ono za što se predstavlja i jamči
- autorizacija – osigurava da određenim informacijama i operacijama smiju pristupiti samo ovlašteni pojedinci
- povjerljivost – osigurava da osjetljive informacije nisu dostupne niti otkrivene neovlaštenim pojedincima
- integritet (cjelovitost) – osigurava da informacijska imovina nije izmijenjena na bilo kakav od neovlaštenih načina
- sljedivost (eng. *traceability*) – osigurava da se provedene radnje subjekta mogu pripisati isključivo tom korisniku
- dostupnost – osigurava da ovlašteni korisnici imaju pristup aplikaciji kad je to potrebno

1.1. Prijetnje i modeli

Svaka potencijalna pojava koji bi mogla naštetiti imovini, bilo opipljivoj, poput web stranice ili baze podataka, ili manje opipljivoj, poput ugleda web aplikacije ili organizacije koja ju razvija, naziva se prijetnja. U savršenom svijetu svaka prijetnja bi trebala biti

tretirana jednako ozbiljno, te ako postoji i najmanja šansa da bi napadač mogao kompromitirati korisnika, odluka o puštanju (engl. *release*) koda trebala bi biti odbačena. Ipak, ovakav pristup u većini slučajeva nije moguć. S obzirom na to razvijene su različite metode klasifikacije prijetnji uz pomoć kojih možemo odrediti prioritete i odlučiti na koje probleme vrijedi utrošiti najviše vremena. U nastavku teksta opisat ćemo neke od najpoznatijih načina kategorizacija prijetnji.

1.1.1. STRIDE

STRIDE je model klasifikacije prijetnji kojeg su izvorno dizajnirali Microsoft-ovi sigurnosni inženjeri. STRIDE ne pokušava rangirati ili dati prioritet ranjivostima već klasificirati ranjivosti prema njihovim potencijalnim učincima [3]. Samo ime STRIDE označava šest sigurnosnih prijetnji a to su:

- *Spoofing*
- *Tampering*
- *Repudiation*
- *Information disclosure*
- *Denial of Service*
- *Elevation of Privilege*

Spoofing predstavlja prijetnju krađe identiteta drugog korisnika.

Tampering predstavlja prijetnju nedopuštene promjene podataka. Ranjivost nedopuštene promjene podataka omogućuje napadačima promjenu podataka bilo to unutar baze podataka ili unutar web aplikacije .

Repudiation predstavlja prijetnju poricanja izvršenja određene radnje.

Information disclosure predstavlja prijetnju čitanja podataka i otkrivanja informacija kojima napadač ne bi trebao imati pristup.

Denial of Service (skraćeno DoS) predstavlja prijetnju uskraćivanja usluge. DoS napadi predstavljaju jednu od najstarijih vrsta napada kojima je cilj onesposobiti određenu web aplikaciju.

Elevation of Privilege (skraćeno EoL) smatra se najozbiljnijim tipom od svih STRIDE kategorija i predstavlja prijetnju povećanja privilegija. Ranjivost povećanja privilegija omogućuje napadačima izvođenje radnji koje inače ne bi mogli učiniti.

STRIDE se koristi kao odgovor na pitanje „Što može poći po zlu u sustavu na kojem radimo?“, a svaka prijetnja predstavlja povredu poželjnog svojstva za sustav. Na sljedećoj tablici dan je prikaz prijetnji koje STRIDE predstavlja te svojstava koje pojedina prijetnja ugrožava.

PRIJETNJA	**ŽELJENO SVOJSTVO SUSTAVA**
Spoofing	Autentičnost
Tampering	Integritet
Repudiation	Neporicanje radnje
Information disclosure	Povjerljivost
Denial of Service	Dostupnost
Elevation of Privilege	Autorizacija

Tablica 1 Povrede poželjnog svojstva sustava s obzirom na prijetnju

1.1.2. CIA

CIA označava trijadu povjerljivost, integritet i dostupnost (engl. *Confidentiality, Integrity and Availability*). Uobičajeni je model koji čini osnovu za razvoj sigurnosnih sustava. Kao i prethodno spomenuti koristi se za pronalaženje ranjivosti i metoda za izradu sigurnosnih rješenja.

Povjerljivost uključuje osiguravanje podataka tajnima ili privatnima. Upravo zbog toga, pristup informacijama mora biti kontroliran kako bi se spriječilo neovlašteno dijeljenje podataka, bilo namjerno ili slučajno. Učinkovit sustav trebao bi osigurati da osobe bez odgovarajućeg ovlaštenja budu spriječene u pristupu imovini i da osobe koje trebaju imati pristup imaju potrebne privilegije [4].

Cjelovitost s druge strane sprječava neovlaštenu izmjenu imovine. Pod izmjenu imovine uključujemo pisanje novih podataka te modifikaciju postojećih.

U konačnici dostupnost znači da sustav funkcionira kako bi trebao i kada bi trebao. Pojedinci s pristupom određenim informacija moraju moći biti u mogućnosti da ih konzumiraju kada im je potrebno, a dolazak do podataka trebao bi se izvršiti u razumnom vremenskom roku. Također, sustav ne bi trebao biti ranjiv na napade uskraćivanjem usluge.

1.1.3. CWE

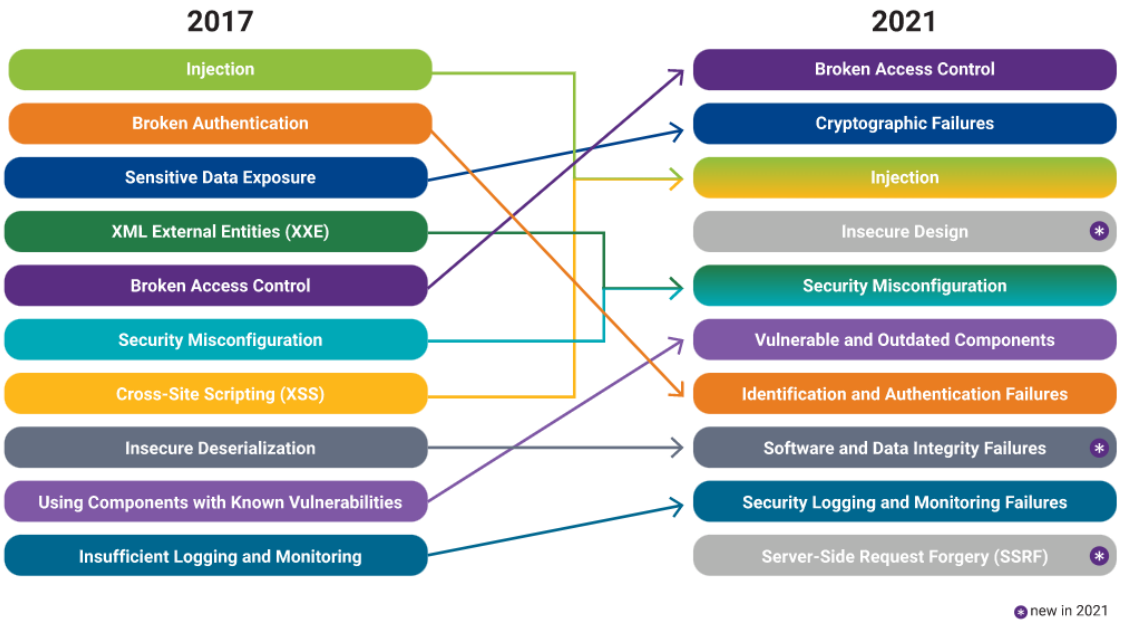
Uobičajeno nabrajanje slabosti (engl. *Common Weakness Enumeration*, skraćeno CWE) je popis općih vrsta ranjivosti softvera kao što su [5]:

- SQL injekcije (engl. *injection*) (CWE-89)
- nepropisna validacija (CWE-20)
- nedostatak enkripcije osjetljivih podataka (CWE-311)
- *Cross-Site Request Forgery* (CSRF) (CWE-352)
- korištenje pokvarenog ili riskantnog kriptografskog algoritma (CWE-327)

CWE lista je specifičnija od općih koncepata prethodno spomenutih modela, gdje svaka prijetnja sadrži opis, mogući utjecaj na sustav unutar kojeg se uspješno izvrši te potencijalna rješenja za nadilaženje ili ublažavanje same prijetnje [6]. Lista je održavana i ažurirana od strane korporacije MITAR, a svake godine objavljuju listu dvadeset i pet najčešćih CWE problema.

1.2. OWASP Top Ten

Jedan od najuglednijih autoriteta u području sigurnosti web aplikacija čiji je cilj poboljšanje sigurnosti i sprječavanje napada na naziva se *Open Web Application Security Project* (skraćeno OWASP). OWASP Top Ten predstavlja listu i smjernice za sanaciju deset najkritičnijih sigurnosnih rizika web aplikacija. Lista se ažurira svake dvije do tri godine u skladu s napretkom i promjenama na *AppSpec* tržištu.



Slika 1.1 OWASP Top Ten

Pokvarena kontrola pristupa (engl. *Broken Access Control*) nalazi se na prvom mjestu kao najkritičniji rizik web aplikacija. Uspješni napadi obično dovode do neovlaštenog otkrivanja informacija, izmjene ili uništavanja svih podataka i obavljanja poslovne funkcije izvan korisničkih ograničenja. Uobičajene ranjivosti kontrole pristupa uključuju EoL, pogrešnu CORS konfiguraciju koja omogućuje pristup API-ju iz nepouzdanih izvora, manipulaciju metapodacima poput tokena kontrole pristupa, kolačića ili skrivenih polja forme koji služe za određivanje privilegija korisnika.

Kriptografski kvarovi (engl. *cryptographic failures*), prethodno nazvani kao izloženost osjetljivim podacima (engl. *sensitive data exposure*) pomaknuli su se s trećeg na drugo mjesto. Osnovni uzrok ovih kvarova povezan je s kriptografijom ili nedostatkom iste, što dovodi do izlaganja osjetljivih podataka.

Injektiranje (engl. *injection*) u koje spadaju SQL, LDAP i OS injekcijski napadi te izvršavanje skripte s drugog računala (engl. *Cross-Site Scripting*, skraćeno XSS) nalazi se na trećem mjestu. Ranjivosti uključuju izravno korištenje neprijateljskih podataka unutar SQL naredbi i neispravna validacija i sanacija istih.

Nesiguran dizajn je nova kategorija usredotočena na rizike povezane s nedostacima u dizajnu i arhitekturi, uz poziv na veću upotrebu modeliranja prijetnji, sigurnih obrazaca dizajna i referentnih arhitektura.

Pogrešne sigurnosne konfiguracije odnose se na samo postavljanje aplikacijskog okruženja te neispravno konfigurirana dopuštenja na *cloud* servisima.

Ranjive i zastarjele komponente odnose se na sve dijelove sustava (OS, poslužitelj, baza podataka, aplikacija te pripadajuća razvojna okruženja i biblioteke) koje su ranjive, nepodržane ili zastarjele.

Pokvarena autentifikacija, odnosno identifikacija i autentifikacijski kvarovi (engl. *identification and authentication failures*) pomakla se s drugog na sedmo mjesto i odnosi se na kvarove vezane uz potvrdu korisničkog identiteta, autentifikaciju i upravljanje sesijom. Ranjivosti pokvarene autentifikacije uključuju dopuštanje slabe ili poznate lozinke, nedostatak provjere autentičnosti s više faktora, upotreba neučinkovitih procesa oporavka zaboravljene lozinke, izloženost identifikatora sesije unutar URL-a.

Pogreške u integritetu softvera i podataka odnose se na kôd i infrastrukturu koji ne štite od narušavanja integriteta. Specifičan primjer je oslanjanje aplikacije na dodatke, biblioteke ili module iz nepouzdanih izvora, repozitorija i mreža za isporuku sadržaja.

Sigurnosno bilježenje i praćenje pomaže u otkrivanju, eskaliranju i odgovoru na aktivne napade. Najčešće ranjivosti ove kategorije rizika jesu nebilježenje događaja koji se mogu revidirati te neadekvatne ili nejasne poruke upozorenja ili greški sustava.

Na posljednjem mjestu nalazi se također nova kategorija rizika, *Server-Side Request Forgery* (skraćeno SSRF). Prijetnja ovoj kategoriji jest dohvaćanje udaljenog resursa bez provjere URL-a kojeg je naveo korisnik. Time omogućujemo napadaču da prinudi aplikaciju na slanje izrađenog zahtjeva na neočekivano odredište, čak i kada je zaštićena nekom vrstom popisa za kontrolu pristupa mreži.

2. Autentikacija

Kontrola pristupa je mehanizam koji regulira pristup podataka ili funkcionalnosti određivanjem je li subjektu dopušteno obavljanje operacija na ciljnom objektu. Kako bi se to utvrdilo ona se oslanja na dva povezana procesa, autentikaciju i autorizaciju.

Autentikacija u suštini dokazuje da smo ono što tvrdimo da jesmo. Autorizacija je, s druge strane, postupak utvrđivanja ima li potvrđeni identitet prava da radi ono što želi. Budući da autorizacija traži dopuštenja na temelju potvrđenog identiteta, mora uslijediti odmah nakon autentikacije. Drugim riječima, možemo imati autentikaciju bez autorizacije, ali ne možemo imati autorizacija bez provjere autentičnosti.

2.1. Vrste autentikacije

U pravilu, korisnik dokazuje svoj identitet davanjem dogovorenih informacija koje dijele korisnik i sustav. Tri su faktora provjere autentičnosti koji se klasificiraju na sljedeći način [7] :

- „nešto što znamo“ – u najčešćem slučaju lozinka ili identifikacijski broj koji se dijeli između korisnika i sustava za upravljanje pristupom identitetu
- „nešto što imamo“ – jednokratna lozinka, pametna kartica ili sigurnosni token
- „nešto što jesmo“ – biometrijske karakteristike poput otiska prsta ili glasa te prepoznavanja lica

S obzirom na količinu faktora koje koristimo unutar samog procesa razlikujemo nekoliko tipova autentikacije.

Osnovna autentikacija (engl. *Basic authentication* ili *Single-factor authentication*, skraćeno SFA) najosnovniji je tip autentikacije. Osnovna provjera autentičnosti zahtjeva od korisnika unos jedne kombinacije informacija poput korisničkog imena i lozinke gdje se potom lozinka uspoređuje s onom koja je pohranjena unutar baze podataka. Prednost korištenja osnovne autentikacije jest jednostavnost koju pruža prilikom implementacije te korištenja krajnjih korisnika. Iako ovaj oblik provjere autentičnosti univerzalno podržavaju svi web preglednici, ona je inherentno nesigurna i podložna napadima grubom silom (engl.

Brute-force attack). Prilikom napada napadač pogađa nepoznate vrijednosti korištenjem svih mogućih kombinacija i analizom povratnih odgovora od poslužitelja.

Dvofaktorska autentikacija (engl. *Two-factor authentication*, skraćeno 2FA) prilikom provjere autentičnosti od korisnika zahtjeva dva faktora, u većini slučajeva faktore nečega što znamo i imamo. Dodajući još jedan faktor u sam proces autentikacije osiguravamo veću sigurnost aplikacije i korisnika jer ako napadač ukrade lozinku, ne može se autenticirati kako mu nedostaje drugi faktor.

Google je korporacija koja je prva uvela dvofaktorsku autentifikaciju 2010. godine i *de facto* unijela standard kako bi proces autentikacije trebao izgledati.

Više-faktorska autentikacija (engl. *Multi-factor authentication*, skraćeno MFA) je autentikacija koja provjeru autentičnosti vrši pomoću dva ili više faktora. Više-faktorska provjera autentičnosti daleko je najbolja obrana od većine napada povezanih s lozinkama, a analiza koju je proveo Microsoft sugerira da bi zaustavila 99,9% kompromitacija računa. Kao takva, trebala bi se provoditi gdje god je to moguće [8]. Osnovni faktori koji se koriste u ovom procesu autentifikacije jesu jednokratne lozinke ili PIN-ovi, pametne kartice i biometrijske informacije poput otiska prsta ili lica.

Glavni problemi s kojima se susreće MFA jesu upotrebljivost i zahtjevna implementacija rješenja. MFA ponekad zahtjeva i do tri faktora provjere autentičnosti pritom povećavajući vrijeme potrebno za samu autentifikaciju i šansu za većim brojem pokušaja prijave od očekivanog.

Iako se provjera autentičnosti korištenjem više faktora općenito smatra sigurnom, postoje slučajevi upotrebe u kojima se ne smatra najboljom opcijom ili čak sigurnom. Primjeri toga su aplikacije trećih strana (engl. *third-party*) koje se žele povezati s web aplikacijom. Kada se to dogodi, ne smatra se sigurnim dopustiti aplikaciji treće strane da pohrani kombinaciju korisnika i lozinke, budući da se tada širi površina napada koja nije u našoj kontroli. Za ovaj slučaj upotrebe postoji nekoliko protokola provjere autentičnosti koji vas mogu zaštititi od izlaganja podataka vaših korisnika napadačima.

- **OAuth** – skraćeno od *Open Authorization,* je protokol koji aplikaciji omogućuje autentikaciju prema poslužitelju kao korisniku, bez potrebe za lozinkama ili bilo kojim poslužiteljem treće strane koji djeluje kao pružatelj identiteta. Koristi token koji je generirao poslužitelj i pruža način na koji se tokovi autorizacije najčešće

odvijaju, tako da klijent, kao što je web aplikacija, može reći poslužitelju koji korisnik koristi uslugu [8].

- **OpenId** – protokol temeljen na HTTP-u koji koristi pružatelje identiteta za provjeru autentičnosti korisnika. Protokol davatelju usluga omogućuje pokretanje jednostruke prijave (engl. *Single sign-on*, skraćeno SSO). Jednostrukom prijavom korisnik ostaje isti na većem broj web aplikacija, bez potrebe da bilo kojoj od njih da lozinku, osim pružatelju identiteta. Zbog svoje jednostavnosti i zaštite lozinki *OpenId* je dobro prihvaćen [8].

2.2. Upravljanje sesijom

Protokol HTTP je *stateless*, u smislu da je svaki poziv prema poslužitelju u potpunosti izoliran od aplikacijskog stanja te ne posjeduje ugrađene mehanizme za praćenje promjena od jednog zahtjeva do drugog. Bez postojanja sesija, sigurnost web aplikacije bila bi krajnje nepraktična, ako ne i nemoguća. Poslužitelj ne bi imao mogućnost odrediti da jedan niz zahtjeva predstavlja radnje jednog korisnika, dok drugi niz zahtjeva, koji je moguće da se događa u isto vrijeme, predstavlja drugog korisnika.

Možemo samo zamisliti, recimo, kad bi dva različita kupca otišla u bankovnu aplikaciju za internetsko bankarstvo u isto vrijeme, a poslužitelj ne zna koji saldo računa poslati nazad kojem korisniku.

Sesije su zapravo još jedan sloj interakcije između web preglednika i poslužitelja, izgrađen na vrhu protokola HTTP, kojeg korisnik može predstaviti uz svaki zahtjev tvrdeći svoj identitet [5]. Ustrajnost (engl. *persistence*) stanja sesije postiže se nekom od uobičajenih strategija:

- kolačići *(*engl. *cookies)* – prednost kolačića jest jednostavnost implementacije. S druge strane oni su u potpunosti nesigurni ako nisu šifrirani. Ograničeni su na samo 4KB te se prenose sa svakim zahtjevom, čak i kada to nije potrebno
- URL upitni (engl. *query*) parametri
- *Web storage* – naziv specifikacije za novi model pohrane uveden s HTML5. Dizajniran je s ciljem prevladavanja nedostataka kolačića. *Web storage* nalazi se na strani klijenta, postojan je, ne prenosi se sa svakim zahtjevom te funkcionira kao datotečni sustav klijenta iz kojeg web aplikacija može čitati i pisati podatke. Iako je

Web storage pružio poboljšanje, još uvijek nije nadvladao najočigledniju prijetnju sigurnosti, a to je, da ne možemo vjerovati korisniku.

U modernom razvoju web aplikacija postoje dva različita pristupa upravljanju sesijom, a to su: pristup temeljen na sesiji ili kolačićima i pristup temeljen na *JSON Web Token-u* (skraćeno JWT). U nastavku teksta osvrnuti ćemo se na potonji te objasniti prednosti i nedostatke svakog od njih.

2.2.1. JWT

JWT se najčešće koriste za identifikaciju autenticiranog korisnika. Koristi se za dijeljenje informacija između dva entiteta, obično klijenta i poslužitelja. JWT sadrži JSON objekte s odgovarajućim informacijama, a svaki od njih je kriptografski potpisan kako bi se osiguralo da sadržaj ne može promijeniti klijent ili zlonamjerna strana [9].

JWT se sastoji od tri dijela:

- zaglavlje (engl. *header*) – sastoji se od dva dijela
    - algoritam potpisa koji se koristi
    - tip tokena koji je uglavnom JWT
- teret (engl. *payload*) – definirani objekt s informacijama
- potpis (engl. *signature*) – niz koji se generira putem kriptografskog algoritma koji se može koristiti za provjeru integriteta JSON podataka.

Neke od glavnih prednosti JWT-a zbog čega u posljednje vrijeme sve češće zamjenjuju kolačiće u svrhu upravljanja sesijom su:

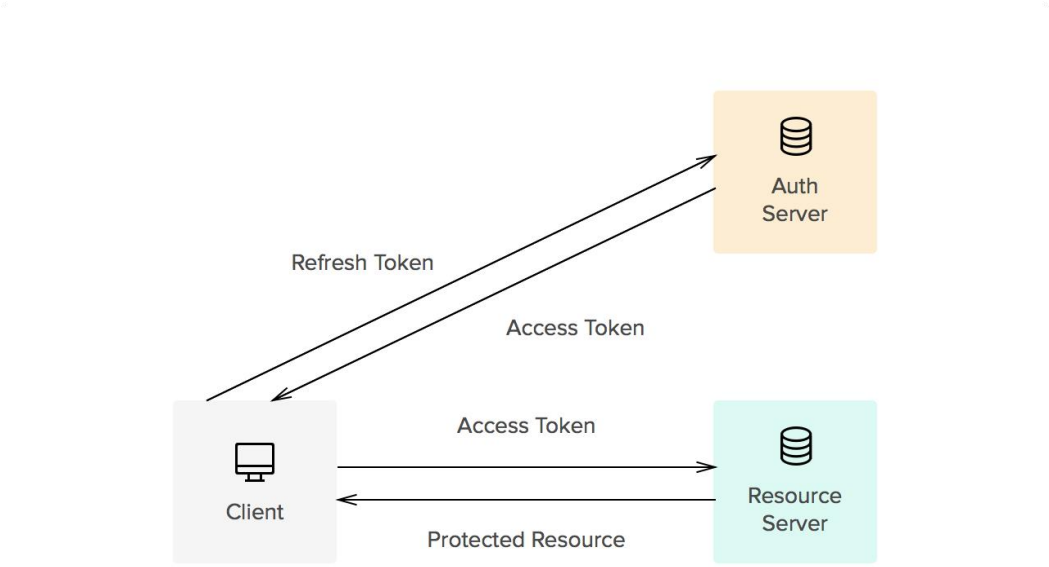
- sigurnost – digitalno su potpisani pomoću tajnog ključa koji ih štiti od izmjene od strane napadača
- pohranjeni su samo na klijentskoj strani – JWT pristup ponekad se naziva i *stateless* pristup upravljanja sesijama budući da se nikakvo stanje ne sprema na poslužitelju ili unutar baze podataka
- učinkovitost – potvrda samog JWT-a vrlo je efikasna budući da ne zahtijeva nikakvo pretraživanje baze podataka. Također isti token može se koristiti za autentifikaciju na različitim poslužiteljima što omogućava razvoj mikro-servisne arhitekture i razvoj odvojeno autentikacijskog poslužitelja

Neki od nedostataka JWT-a su:

- neopozivost – zbog njihove prirode i *stateless* procesa verifikacije, može biti teško opozvati JWT prije nego što prirodno istekne. Stoga se akcije poput trenutne zabrane korisnika ne mogu jednostavno provesti
- ovisni o jednom tajnom ključu – stvaranje JWT-a ovisi o jednom tajnom ključu. Ako je tajni ključ ugrožen, napadač može lažirati identitet bilo kojeg korisnika. Taj rizik možemo smanjiti povremenom promjenom tajnog ključa

U web terminologiji JWT se naziva i token za pristup (engl. *access token*). Kao što je spomenuto, iz sigurnosnih razloga, tokeni za pristup trebali bi biti valjani kratko vrijeme. Nakon što isteknu, aplikacija može iskoristiti token za osvježavanja (engl. *refresh token*) za osvježavanje tokena za pristup. Odnosno, token za osvježavanje je artefakt koji klijentskoj aplikaciji omogućuje dohvaćanje novih tokena za pristup bez potrebe da se korisnik ponovno prijavi [10].

Stavka na koju bi trebali pripaziti je također sigurnost tokena za osvježavanje. Budući da svatko tko posjeduje token za osvježavanje ima mogućnost dobiti novi pristupni token moramo imati strategiju koja ograničava ili sužava njihovu upotrebu. U tu svrhu provodi se rotacija tokena za osvježavanje, odnosno svaki put kad aplikacija razmijeni *refresh token* za novi *access token*, trenutni *refresh token* se invalidira te se korisniku vraća novi.



Slika 2.1 Tijek razmjene tokena

3. Autorizacija

Nadovezujući se na prethodno poglavlje, veliki dio kontrole pristupa predstavlja autentikacija, odnosno dokazivanje korisnika da je netko poznat web aplikaciji pružanjem jednog ili više faktora autentičnosti kao što je zaporka, sigurnosni token ili biometrijski otisak prsta. Drugi dio kontrole pristupa predstavlja autorizacija. Autorizacija pojednostavljenim riječima znači smije li korisnik pristupiti određenom resursu. Kontrola pristupa može imati razne oblike. Osim utvrđivanja hoće li korisnik imati prava na korištenje resursa, sustav kontrole pristupa može ograničiti kada i kako se resurs može koristiti.

Autorizacija se odnosi na odluku hoće li korisniku biti dopušten pristup resursu sustava. U tom procesu sustav treba održavati odnos između identifikatora korisnika i samih resursa. Nužno je da autorizacija ovisi o ispravnoj autentifikaciji jer ako sustav ne može biti siguran u identitet korisnika, ne postoji valjan način da se utvrdi treba li korisniku odobriti pristup određenom resursu.

Tri su temeljna cilja zbog koji autorizaciju uklapamo unutar aplikacije [11]:

- ograničavanje korisničkih radnji na samo dozvoljene akcije
- kontrola pristupa resursima s obzirom na korisničke uloge i privilegije
- povećanje sigurnosti

3.1. Temeljni pojmovi kontrole pristupa

Tijekom posljednja tri desetljeća razvila se dosljedna terminologija za opisivanje modela i sustava. Gotovo svaki model kontrole pristupa može se formalno izraziti korištenjem pojmova korisnika (engl. *users*), subjekata (engl. *subjects*), objekata (engl. *objects*), operacija (engl. *operations*) i dopuštenja (engl. *permissions*) te međusobnih odnosa između tih entiteta [12].

Pojam korisnika odnosi se na osobe koje se povezuju s računalnim sustavom. Mehanizam provjere autentičnosti omogućuje spajanje identifikatora s pripadajućim korisnikom. Instanca korisničkog dijaloga sa sustavom naziva sesija.

Računalni proces koji djeluje u ime korisnika naziva se subjekt. Sve radnje korisnika na računalnom sustavu izvode se putem nekog programa, a sam korisnik može istovremeno imati više subjekata u radu.

Objekt može biti bilo koji resurs dostupan u računalnom sustavu, uključujući datoteke, periferne uređaje kao što su pisači, baze podataka i detaljne entitete kao što su pojedinačna polja u zapisima baze podataka.

Operacija je aktivni proces kojeg poziva subjekt.

Dopuštenja ili privilegije su ovlaštenja za izvođenje neke operacije unutar sustava.

3.2. Načelo najmanjih privilegija

Načelo najmanjih privilegija (engl. *Least privilege*), kao što i ime kaže, označava praksu dodjeljivanja minimalnih privilegija korisnicima. Načelo najmanjih privilegija izbjegava problem pojedinca koji ima sposobnost obavljanja nepotrebne i potencijalno štetne radnje i važna je stavka za održavanje povjerljivosti i integriteta sustava [12].

Osigurati pridržavanje načela najmanjih privilegija u velikoj je mjeri administrativni izazov koji zahtijeva identifikaciju radnih funkcija, specifikaciju skupa dopuštenja potrebnih za obavljanje svake funkcije i ograničenje korisnika na domenu specifičnu za dane privilegije.

Primjerice, korisnički računi baze podataka koje koriste web aplikacije često imaju privilegije izvan potrebnih ili preporučljivih. Dopuštanje web aplikacijama da koriste administratorske ili druge privilegirane račune baze podataka uništava sposobnost poslužitelja da se obrani od pristupa ili izmjene neovlaštenih resursa.

3.3. Modeli kontrole pristupa

S obzirom na to kako kontrola pristupa dodjeljuje prava korisniku razlikujemo tri osnovna modela [11].

RBAC, skraćeno od *Role-Based Access Control*, je model koji je temeljen na ulogama. Svakom subjektu je pridijeljena jedna ili više uloga koja dopušta određene operacije. Kontrola pristupa temeljena na ulogama ograničava pristup sustavu na temelju uloge subjekta unutar organizacije i postala je jedna od glavnih metoda za naprednu kontrolu

pristupa. Subjektima je dopušten pristup samo onim informacijama koje su nužne za učinkovito obavljanje njihovih radnih dužnosti. Pristup se može temeljiti na nekoliko čimbenika kao što su autoritet, odgovornost i kompetencija za posao. Osim toga, pristup resursima može biti ograničen na određene zadatke kao što je mogućnost pregledavanja, stvaranja ili izmjene datoteke. Rezultat toga je da zaposlenici niže razine obično nemaju pristup osjetljivim podacima ako im nisu potrebni za ispunjavanje njihovih odgovornosti.

DAC, skraćeno od *Discretionary Access Control*, je model kontrole pristupa koja odobrava ili ograničava pristup resursu putem politike pristupa koju određuju subjekt. DAC je primjer diskrecijskog modela jer subjekt može prenijeti autenticirane objekte ili pristup informacijama drugim korisnicima. Drugim riječima, vlasnik određuje privilegije pristupa objektu.

MAC, skraćeno od *Mandatory Access Control*, je model kontrole pristupa gdje operativni sustav korisnicima omogućuje pristup na temelju povjerljivosti podataka i razina korisničkog odobrenja. Pristup se odobrava na temelju potrebe za informacijama, odnosno, korisnici moraju dokazati potrebu za informacijama prije nego li dobiju pristup. Smatra se najsigurnijim modelom kontrole pristupa. Pravila pristupa u ovom modelu ručno definiraju administratori sustava, a obični korisnici ne mogu mijenjati sigurnosne atribute čak ni za podatke koje su sami stvorili.

3.4. RBAC

RBAC je model koji privlači veliku pozornost, osobito za komercijalne primjene, zbog svog potencijala za smanjenje složenosti i troškova sigurnosne administracije unutar velikih umreženih aplikacija. Sigurnosna administracija je uvelike pojednostavljena korištenjem uloga, hijerarhija i ograničenja za organiziranje privilegija. RBAC smanjuje troškove unutar organizacije vodeći računa o samim korisničkim računima zaposlenika, koji se mijenjaju mnogo češće nego dužnosti unutar određenih pozicija. Primjerice ako zaposlenik promijeni svoju poziciju unutar organizacije mijenja se samo njegova ili njezina uloga. Sukladno tome, nepotrebno je opozvati postojeće privilegije i dodijeliti potpuno nove [12].

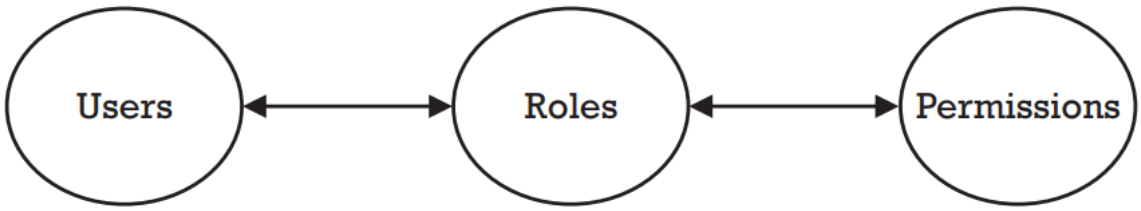
Ljudi su koristili uloge i privilegije za upravljanje pristupa komercijalnim računalnim sustavima barem od 1970-ih. Međutim, ti su postupci bili *ad hoc* i često su se morali redizajnirati za svaki novi sustav ili pojedinačni slučaj. Tek su 1992. istraživači s

Američkog nacionalnog instituta za standarde (NIST) počeli formalizirati sustav koji poznajemo kao kontrolu pristupa temeljenu na ulogama. Te su godine Ferraiolo i Kuhn postavili temelje za model koji danas koristimo u radu i koji opisuje metodologiju kontrole pristupa opće namjene prikladnu za civilnu i komercijalnu upotrebu [12].

Osnovni (engl. *core*) RBAC model može poslužiti kao samostalna metoda kontrole pristupa, ali je ujedno i osnova za hijerarhijske modele i modele razdvajanja dužnosti o kojima ćemo nešto više u nastavku teksta. Svi RBAC modeli moraju se pridržavati sljedećih pravila:

- **dodjela uloga** (engl. *role assignment*) – subjekt može koristiti privilegije samo kada je istom dodijeljena uloga
- **autorizacija uloge** (engl. *role authorization*) – sustav mora autorizirati aktivnu ulogu subjekta
- **autorizacija dopuštenja** (engl. *permission authorization*) – subjekt može primijeniti samo dopuštenja dodijeljena aktivnoj ulozi subjekta

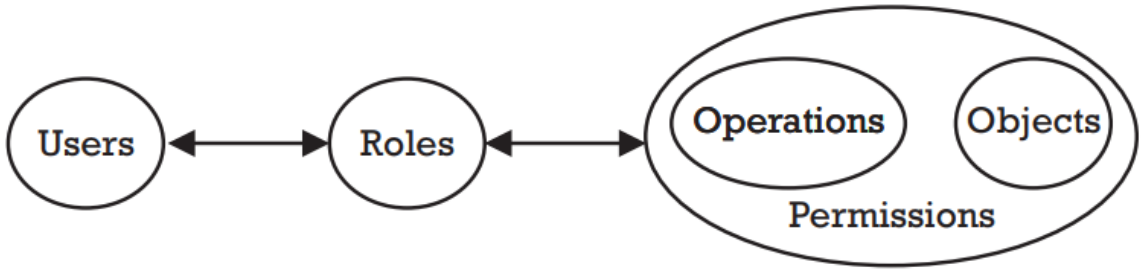
Osnovni koncept RBAC-a navodi da su korisnici dodijeljeni ulogama koje sadrže skup dopuštenja. Korisnici dobivaju dopuštenja dodjeljivanjem uloga. Dodjeljivanje korisničke uloge i dopuštenja određena je relacijom više-naprema-više (engl. *many-to-many*), što znači da se istom korisniku može dodijeliti veći broj uloga, a isto tako uloga može imati veći broj korisnika. Slično tome, dopuštenje se može dodijeliti mnogim ulogama, a jedna uloga može imati mnogo dopuštenja [12].



Slika 3.1 Shema RBAC modela

Prilikom modeliranja sustava kontrole pristupa, administratori mogu tretirati dozvole kao apstraktni koncept koji se odnosi na proizvoljno vezanje računalnih operacija i objekata resursa. Vrsta operacija i objekti koje RBAC kontrolira su ovisni o vrsti sustava u koji će se implementirati. Primjerice, unutar operativnog sustava, operacije mogu uključivati čitanje, pisanje i izvršavanje, dok unutar sustava za upravljanje bazom podataka (DBMS), operacije mogu uključivati umetanje, brisanje, dodavanje i ažuriranje određenih redaka.

Skup objekata obuhvaćenih sustavom RBAC uključuju sve objekte kojima operacije mogu pristupiti, no oni ne moraju biti uključeni u samu shemu modela. Na sljedećoj slici dan je prikaz proširene sheme modela koji uključuje relaciju između operacije i objekta.



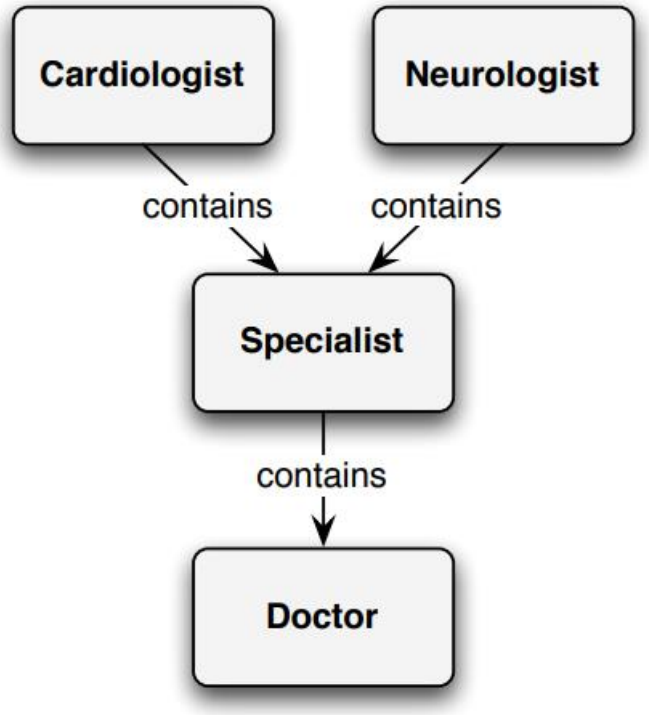
Slika 3.2 Proširena shema RBAC modela

3.4.1. Model hijerarhije uloga

Hijerarhije uloga (engl. *Role hierarchies*) nadilaze osnovne strukture RBAC-a, a glavna motivacija za ovaj model je zapažanje da se pojedinačne uloge unutar organizacije često preklapaju, odnosno, korisnici s različitim ulogama ponekad trebaju izvršiti iste operacije. Hijerarhija uloga način je strukturiranja uloga koje odražavaju složenu organizacijsku strukturu i omogućuju dijeljenje i nasljeđivanje dopuštenja između uloga. Jednostavan primjer hijerarhijskog RBAC-a je niz uloga, u kojem svaka uloga nasljeđuje dopuštenja prethodne i dodaje nova dopuštenja [12].

Kako bismo dočarali potencijal preklapanja dozvola razmotrimo sljedeći primjer. Neka unutar bolnice postoje četiri uloge: kardiolog, neurolog, specijalist i liječnik. Budući da su i kardiolog i neurolog specijalisti, prikladno je pretpostaviti da bi sva dopuštenja dodijeljena ulozi specijalista također bila dodijeljena ulogama kardiologa i neurologa. Osim toga, budući da specijalist obavlja mnoge dužnosti liječnika, dopuštenja koja su dodijeljena ulozi liječnika također bi trebala biti dodijeljena ulozi specijalista.

Najmoćnije (engl. *senior*) uloge s najvećim brojem dozvola predstavljene su na vrhu stabla, a manje moćne (engl. *junior*) smještene su na dnu. *Senior* uloge sadrže više dopuštenja od *junior* uloga. U našem primjeru, uloge neurolog i kardiolog nasljeđuju sva dopuštenja od uloga specijalista i liječnika. Zbog činjenice da su dopuštenja naslijeđena, svaki korisnik koji je dodijeljen ulozi kardiolog ovlašten je za dopuštenja koja su povezana s ulogom kardiolog kao i dopuštenja povezana s ulogama specijalist i liječnik.



Slika 3.3 Model hijerarhije uloga

3.4.2. Model ograničenja

Još jedan od modela koji nasljeđuje i proširuje osnovne strukture RBAC-a jest model ograničenja. On se temelji na odvajanju dužnosti (engl. *Separation of Duty*, skraćeno SoD) i navodi da su kritične operacije podijeljene između dvoje ili više korisnika, tako da niti jedan pojedinac ne može samostalno ugroziti sigurnost [12]. Najvažnija dva tipa podjele dužnosti jesu:

- statičko odvajanje dužnosti (engl. *Static Separation of Duty*, skraćeno SSD) – korisnik ne može imati uloge koje se međusobno isključuju. To osigurava, na primjer, da korisnik ne može izvršiti i odobriti kupnju
- dinamičko odvajanje dužnosti (engl. *Dynamic Separation of Duty*, skraćeno DSD) – korisnici mogu biti članovi sukobljenih uloga. Međutim, isti korisnik ne može obavljati obje uloge u jednoj sesiji. Ovo ograničenje pomaže u kontroli internih sigurnosnih prijetnji, na primjer, provođenjem pravila za dvije osobe koje zahtijeva da dva različita korisnika odobre radnju

3.4.3. Usporedba s DAC i MAC modelima

RBAC predstavlja veliki napredak u smislu fleksibilnosti i detaljnosti kontrole u odnosu na postojeće standarde DAC i MAC.

DAC, kao što sam naziv implicira, prepušta odobravanje i opoziv privilegija diskreciji pojedinačnih korisnika. DAC omogućuje korisnicima da dodijele ili opozovu pristup bilo kojem od objekata pod njihovom kontrolom bez posredovanja administratora sustava. Za mnoge sustave, krajnji korisnici ne posjeduju informacije za koje im je dopušten pristup kao što pretpostavlja DAC politika. U njihovom slučaju organizacija ili korporacija je stvarni vlasnik objekata sustava i možda nije prikladno dopustiti korisnicima da daju prava pristupa objektima. S RBAC, odluke o pristupu temelje se na ulogama koje pojedini korisnici imaju kao dio organizacije. To uključuje specifikaciju dužnosti, odgovornosti i kvalifikacije. Na primjer, uloge koje pojedinac povezan s bolnicom može preuzeti uključuju liječnika, medicinsku sestru i pacijenta. Korisnici inače ne mogu prenijeti svoja dopuštenja drugim korisnicima na njihovu diskreciju. Na primjer, liječnik koji možda posjeduje dopuštenje za izvođenje operacije ne bi trebao moći prenijeti to dopuštenje na medicinsku sestru. Kao takav, RBAC se ponekad opisuje kao oblik MAC-a u kojem su korisnici neizbježno ograničeni i nemaju utjecaja na provođenje zaštitnih politika organizacije [12].

Ipak, RBAC je različit od MAC-a po pitanju nekoliko stvari. MAC podržava zahtjeve i propise koji se odnose na neovlašteni pristup povjerljivim podacima, a posebice na njihovu zaštitu povjerljivosti. Kontrola pristupa nad operacijama bavi se samo sprječavanjem nezakonite dostupnosti osjetljivih informacija, a ne njihovim integritetom, neovlaštenom izmjenom ili uništenjem. Kako bismo razlikovali RBAC od specifičnosti politike MAC-a, RBAC je često karakteriziran kao ne-diskrecijska kontrola pristupa. RBAC omogućuje provođenje raznih politika zaštita koje se mogu prilagoditi za svaku pojedinačnu organizaciju [12].

3.4.4. Prednosti i nedostatci

U ovom poglavlju opisat će se glavne prednosti i neki od nedostatka modela kontrole pristupa temeljenog na ulogama [11].

Prednosti RBAC-a:

- **povećava stupanj sigurnosti** – RBAC ograničava korisnički pristup na minimalnu potrebnu razinu za obavljanje posla. Nastavno na to, organizacije su u povoljnom položaju da provedu sigurnosne prakse poput načela najmanje privilegije, što umanjuje rizik od povrede i curenja podataka. Ako i dođe do proboja, RBAC ograničava površinu napada gdje je pristup zaštićenim informacijama ograničen na ulogu koju je napadač koristio kao ulaznu točku. Načelom razdvajanja dužnosti u još većoj mjeri poboljšavamo sigurnost onemogućujući da bilo koji korisnik ima isključivu ovlast za obavljanje zadatka
- **pojednostavljuje tijek rada** – RBAC korisnicima daje pristup potreban za njihove uloge, što pomaže u uklanjanju uskog grla (engl. *bottleneck*). Zaposlenici se više ne moraju, u velikoj mjeri, oslanjati na administratore u cilju pristupa određenim podacima i sustavima. Također, administratori mogu jednostavno ažurirati dopuštenja za postojeće zaposlenike koji mijenjaju uloge unutar organizacije ili za korisnike treće strane koji trebaju privremeni pristup vašoj mreži. Navedene stavke pružaju ekonomsku korist i povećavaju zadovoljstvo zaposlenika
- **poboljšava usklađenost** – sve organizacije, bilo one pružatelji zdravstvenih usluga ili financijske institucije, moraju se pridržavati određenih propisa o privatnosti i povjerljivosti podataka. Osim toga, certifikati sukladnosti kao što je SOC 2 mogu poboljšati reputaciju same organizacije i ponuditi konkurentsku prednost na tržištu. Dokazivanje usklađenosti odražava predanost osiguravanju sigurnosti korisničkih podataka, kao i sposobnost zaštite osjetljivih informacija općenito

Nedostatci RBAC-a:

- **zahtijeva poslovno znanje** – organizacije moraju biti u stanju uspješno koordinirati među odjelima prilikom kategorizacije uloga i definiranja prava kontrole pristupa. To zahtijeva jasno razumijevanje idealne strukture organizacije kao i tehničke infrastrukture koja je podržava
- **zahtijeva promišljenu implementaciju** – dodjeljivanje uloga može biti izazovan zadatak. Primjerice, trebaju li sigurnosni timovi pristup podacima koje pokušavaju zaštititi i na kojoj razini, treba li korisniku dodijeliti ulogu izvan njegovog odjela kako bi se osigurao privremeni pristup povlaštenim datotekama, neka su od pitanja koja se mogu pojaviti

- **nedostatak fleksibilnosti** – RBAC drži reputaciju rigidnog modela kontrole pristupa. Organizacije rastu, timovi se šire, a potrebe pristupa se mijenjaju. Uloge koje smo definirali na početku možda više neće odgovarati ciljevima tvrtke. Povrh toga, administratori se često suočavaju s pritiskom da što brže uključe nove zaposlenike, čak i one s nedefiniranim ulogama
- **dovodi do eksplozije uloga** – neke organizacije pokušavaju zaobići gore navedene probleme definiranjem sve preciznijih uloga, te stvaranjem novih kako se pojavi potreba. Iako to kratkoročno može ublažiti problem, čini RBAC zbunjujućim i teškim za upravljanje. Problem se često naziva eksplozija uloga i jedan je od najčešćih prigovora ovom modelu kontrole pristupa

4. Kompromitirana kontrola pristupa

Pokvarena kontrola pristupa po OWAPS Top Ten listi najkritičnijih rizika web aplikacija pomaknula se s petog na prvo mjesto se na prvo mjesto. Kvarovi obično dovode do neovlaštenog otkrivanja informacija, izmjene ili uništavanja svih podataka ili obavljanja poslovne funkcije izvan korisničkih ograničenja. Uobičajene ranjivosti kontrole pristupa uključuju [13]:

- kršenje načela najmanje privilegije, gdje bi se pristup trebao odobriti samo za određene mogućnosti, uloge ili korisnike, ali je dostupan svima
- zaobilaženje provjera kontrole pristupa mijenjanjem URL-a ili internog stanja aplikacije
- dopuštanje pregledavanja ili uređivanja tuđeg korisničkog računa pružanjem njegovog jedinstvenog identifikatora (nesigurne izravne reference na objekte)
- pristup API-ju s nedostajućim kontrolama pristupa za POST, PUT i DELETE radnje
- povećanje privilegija gdje napadač djeluje kao korisnik bez prijave ili administrator kada je prijavljen kao korisnik
- manipulaciju meta podacima, kao što je ponovno reproduciranje ili neovlašteno mijenjanje JWT-a ili kolačića kojima se manipulira radi povećanja privilegija
- pogrešnu konfiguraciju CORS-a koja omogućuje pristup API-ju iz neovlaštenih ili nepouzdanih izvora

4.1. Napadi

Kontrola pristupa se u svom procesu regulacije pristupa podacima oslanja na dva povezana procesa, autentikaciju i autorizaciju. S obzirom na to, unutar ovog poglavlja navest ćemo neke od najčešćih napada usmjerene upravo na ova dva procesa.

4.1.1. Napad grubom silom

Napad grubom silom (engl. *Brute-force attack*) je jednostavna, ali pouzdana taktika za dobivanje neovlaštenog pristupa korisničkim računima, sustavima i mrežama organizacija. Napadač isprobava više korisničkih imena i zaporki, često koristeći računalo za testiranje širokog spektra kombinacija, sve dok ne pronađe točne podatke za prijavu. U procesu napada uvelike mu pomažu propusti načinjeni u implementaciji autentikacije poput povratnih preopširnih poruka o neuspjehu i neograničen broj pokušaja pogađanja korisničkog imena i zaporke.

4.1.2. Otmica sesije

Otmica sesije (engl. *Session hijacking*) ugrožava token sesije krađom ili predviđanjem važećeg tokena kako bi se dobio neovlašteni pristup web poslužitelju [14]. Najčešći načini ugrožavanja tokena su:

- predvidljivost identifikatora sesije – propust koji se javlja u generiranju identifikatora koji napadaču omogućuje jednostavno predviđanje vrijednosti tokena
- napadom skriptnim kodom (engl. *Cross-Site Scripting*, skraćeno XSS)
- napadom čovjeka u sredini (engl. *Man-in-the-Middle*, skraćeno MITM) – komunikacija između dvije druge strane prisluškuje se od strane napadača, bez da biva otkriven dopuštajući mu presretanje informacija

4.1.3. Nesigurna izravna referenca objekta

Nesigurna izravna referenca objekta (engl. *Insecure Direct Object Reference*, skraćeno IDOR) događa se kada aplikacija izloži referencu internom implementacijskom objektu. Koristeći ovaj način, otkriva pravi identifikator i uzorak koji se koristi za element na pozadinskoj strani pohrane [15]. Najčešći primjer jest identifikator zapisa u sustavu za pohranu (baza podataka, datotečni sustav).

```
https://app.com/sensitive-data?id=15
```

Iz primjera moguće je zaključiti da se URL parametar odnosi na redak u bazi s indeksom 15. IDOR ne donosi izravan sigurnosni problem jer, sam po sebi, otkriva samo uzorak koji se koristi za identifikator objekta. IDOR donosi, ovisno o uzorku, mogućnost za napadača da pokrene napad enumeracijom kako bi pokušao ispitati pristup pridruženim objektima.

4.1.4. CSRF

CSRF, skraćeno od *Cross-Site Request Forgery*, vrsta je napada koji se događa kada zlonamjerna web stranica uzrokuje da korisnikov web preglednik izvede neželjenu radnju na pouzdanoj stranici za vrijeme aktive korisničke sesije. CSRF napad funkcionira jer zahtjevi preglednika automatski uključuju sve kolačiće. Stoga, ako je korisnik autenticiran na stranici, ona ne može razlikovati legitimne autorizirane zahtjeve od krivotvorenih. Ovaj napad je osujećen kada se koristi odgovarajuća autorizacija, što implicira da je potreban mehanizam izazov-odgovor koji provjerava identitet i ovlaštenje podnositelja zahtjeva [16].

4.2. Sigurnosni principi

Uvođenje sigurnosnih principa u procesu implementacije mehanizama za kontrolu pristupa uvelike pružaju zaštitu od prethodno navedenih napada.

Implementacija odgovarajuće kontrole jačine zaporke

Ključna briga pri korištenju zaporki u procesu provjere autentičnosti jest njezina „snaga“. Snažna lozinka u velikoj mjeri otežava njezino pogađanje bilo ručnim ili automatiziranim sredstvima. Karakteristike koje definiraju snažnu lozinku jesu [8]:

- minimalna duljina lozinke trebala bi biti zahtijevana od strane aplikacije i ne bi trebala biti kraća od 8 znakova
- maksimalna duljina zaporke ne bi trebala biti postavljana prenisko (uobičajena maksimalna duljina zaporke iznosi 64 znaka)
- složenost zaporke trebala bi biti zahtijevana od stranice aplikacije i uključivati kategorije velikih i malih slova, brojeva i simbola

Implementacija sigurnog mehanizma za oporavak zaporke

Uobičajeno je da aplikacija sadrži mehanizam koji korisniku omogućuje pristup svom računu u slučaju da zaboravi zaporku. Prilikom oporavka zaporke sustav bi trebao vratiti dosljednu poruku i za postojeće i nepostojeće račune. Za najjednostavniju i najbržu implementaciju prikladno je koristiti URL tokene nasumično generirane korištenjem kriptografski sigurnog algoritma. U konačnici tokeni bi trebali biti kreirani za jednokratnu upotrebu s datumom isteka trajanja u svrhu zaštite od napada grubom silom [8].

Implementacija pohrane i usporedbe lozinki na siguran način

Zaporke unutar aplikacije trebale bi se pohraniti korištenjem kriptografski sigurnog algoritma. Isto tako unesenu korisničku zaporku s pohranjenom trebalo bi provjeriti sigurnom funkcijom koju pruža određena biblioteka ili razvojno okruženje [8].

Implementacija povratnih poruka o neuspješnoj autentikaciji

Neispravno implementirane poruke o pogrešci u slučaju provjere autentičnosti mogu se zlonamjerno iskoristiti. U slučaju pogreške aplikacija bi trebala odgovoriti generičkom porukom o pogrešci tipa „Pogrešno korisničko ime ili lozinka“. Ovakvom porukom napadaču ne dajemo na uvid koji je podatak točan, a koji nije [8].

Implementacija više-faktorske autentikacije

Više-faktorska autentikacija je daleko najbolja obrana od većine napada povezanih s autentikacijom, i kao takva, trebala bi se implementirati kad god je to moguće [8].

Implementacija atributa kolačića

Mehanizam razmjene ID-a sesije koji se temelji na kolačićima pruža više sigurnosnih značajki u obliku atributa kolačića koji se mogu koristiti za zaštitu razmjene ID-a sesije [17] :

- *Secure* atribut – atribut nalaže web pregledniku da šalje kolačić samo putem šifrirane HTTPS (SSL/TLS) veze. Ovaj mehanizam zaštite sesije obavezan je kako bi se spriječilo otkrivanje ID-a sesije putem MitM napada
- *HttpOnly* atribut – atribut nalaže web pregledniku da skriptama ne dopusti mogućnost pristupa kolačićima putem DOM objekta *document.cookie*. Ovaj mehanizam zaštite sesije obavezan je kako bi se spriječila krađa ID-a sesije putem XSS napada
- *SameSite* atribut – atribut sprječava preglednike da pošalju kolačić s *cross-site* zahtjevima. Glavni cilj je ublažiti rizik od curenja informacija s drugog izvora i pružiti određenu zaštitu od CSRF napada

Implementacija provjere dozvola sa svakim zahtjevom

Korisničku dozvolu za izvršenje akcije bi trebalo potvrditi na svakom zahtjevu, bez obzira na to tko je pokrenuo zahtjev.

Implementacija CSRF tokena

CSRF token je siguran nasumični token koji se koristi za sprječavanje CSRF napada. Aplikacija dodjeljuje jedinstveni CSRF token za svaku korisničku sesiju. Ovi se tokeni umeću unutar skrivenih parametara HTML formi koji se odnose na kritične operacije na strani poslužitelja. Pozivanjem istih zahtjev koji generira preglednik mora sadržavati pridruženi CSRF token koji služi za provjeru legitimnosti zahtjeva krajnjeg korisnika. Ako se tokeni ne poklapaju zahtjev bi trebao biti odbijen [18].

5. Praktični dio

Za potrebe demonstracije koncepta autorizacije korisnika temeljene na ulogama napravljena je aplikacija „CleanZee“. Glavni cilj je bio upoznati se s osnovama implementacije autorizacije te produbiti postojeće znanje o već spomenutim tehnologijama koristeći moderne standarde kodiranja.

Problem koji aplikacija rješava jest jednostavno pružanje i korištenje usluga pranja rublja. S jedne strane, kupcima je omogućeno naručivanje pranja rublja, a s druge, vlasnicima praonica, dodavanje proizvoda te praćenje rada i dobiti svojeg poslovanja. Također, unutar aplikacije postoji i sučelje namijenjeno za administratora koji ima uvid na cjelokupni rad i procese koje obavljaju kupci i vlasnici praonica.

5.1. Analiza aplikacije

5.1.1. Analiza i procjena tržišta

Trenutno na tržištu postoji poveći broj aplikacija praonica rublja zbog čega možemo preduhitreno zaključiti da bi sam probitak aplikacije imao manje šanse za uspjeh. Ipak svaka aplikacija djeluje i razvijena je za određeno područje ili državu u kojoj broji većinu svojih klijenata.

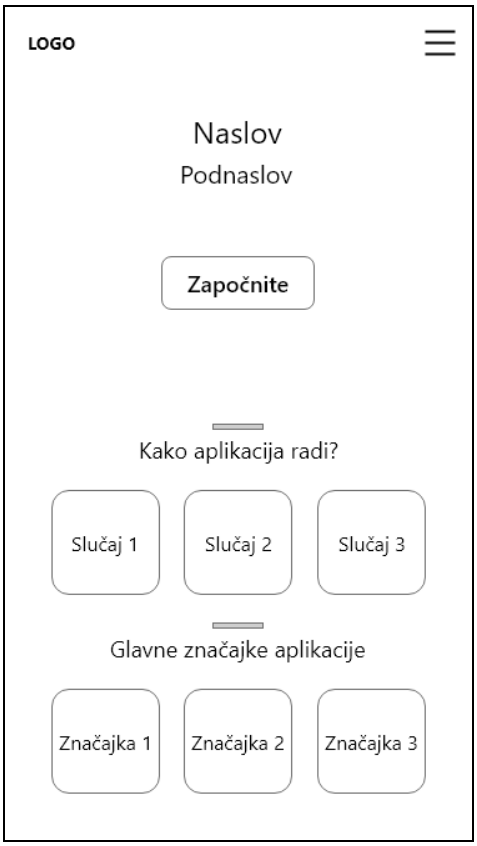
Primjer jedne aplikacije koja pruža cjelokupan asortiman usluga jest CleanCloud. Aplikacija CleanCloud, odnosno softver kao servis (engl. *Softver as a Service*) prilikom prijave unutar svog sustava dodaje trgovine po dogovoru s vlasnicima koje su zatim vidljive klijentima unutar odvojene mobilne aplikacije preko koje se vrši narudžba pranja rublja.

Sama aplikacija CleanCloud poslužila je kao ideja za izradu ovog diplomskog rada. Poboljšana funkcionalnost kojoj smo težili prilikom izrade jest jedinstvena aplikacija unutar koje se mogu prijaviti svi tipovi korisnika, bilo oni vlasnici trgovina ili kupci. Također neke od dodatnih funkcionalnosti jesu pregled najbližih trgovina namijenjen za kupce, te mogućnosti individualne prilagodbe izgleda svojih trgovina i usluga za vlasnike.

5.1.2. Analiza i implementacija ideje

Prvotni zadatak bio je analiza i koncipiranje ideje zajedno sa svim značajkama koje aplikacija pruža. Kroz sljedećih nekoliko stranica opisat ćemo značajke i funkcionalnosti koje bi zapravo trebale predstavljati minimalno održiv proizvod (engl. *Minimum Viable Product*, skraćeno MVP). Za implementaciju same ideje i dizajna značajki koje bi korisničko sučelje trebalo sadržavati koristili smo Adobe XD, alat za vektorski dizajn za web i mobilne aplikacije.

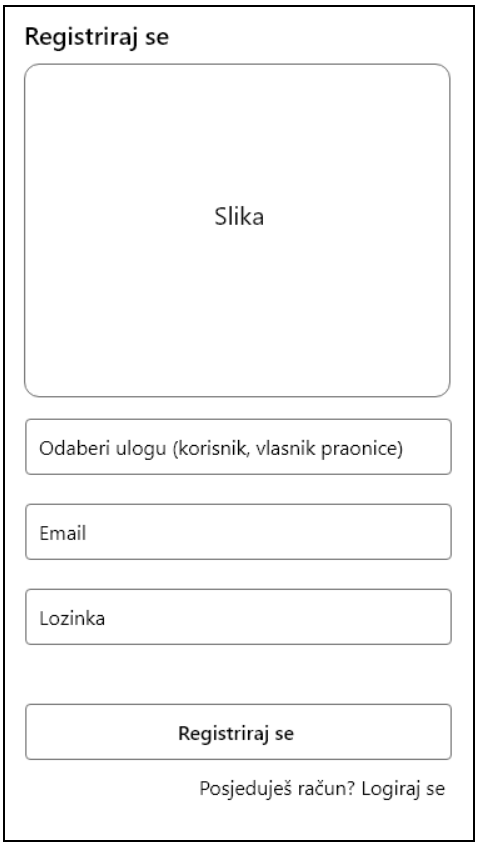
Aplikacija je zamišljena kao spoj statičke web stranice i dinamičke web aplikacije. Statički dio predstavljala bi početna stranica. Kao i većina web stranica sadržavala bi informacije glavnih značajki aplikacije, kako aplikacija funkcionira te podnožja unutar kojeg bi se nalazile informacije o samom vlasniku aplikacije i kontakt forme. Također zamišljeno je da početna stranica sadržava i navigacijsku traku s pregledom svih poglavlja stranice.



Slika 5.1 Planirani izgled početne stranice

Nakon pregleda početne stranice dolazimo do dijela koji se odnosi na autentikaciju korisnika. Sami korak autentikacije zamišljen je podjelom na nekoliko pod-stranica. Na stranici za registraciju korisnik bi unosio svoje podatke te željenu ulogu unutar same

aplikacije (korisnik ili vlasnik praonice). Stranica za prijavu služila bi za konkretnu provjeru identiteta korisnika. Za povrat izgubljene zaporke služile bi dvije stranice: stranica za prijavu zaboravljene zaporke pomoću koje bi korisnik putem unesenog emaila trebao obavijestiti aplikaciju o potrebi za promjenom zaporke i stranica za promjenu zaporke pomoću koje bi korisnik unio novu željenu zaporku.



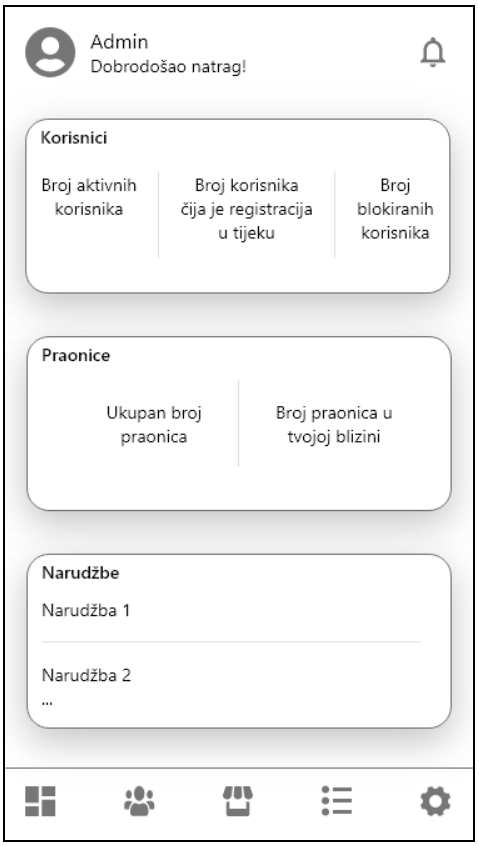
Slika 5.2 Planirani izgled stranice za registraciju

Slika 5.3 Planirani izgled stranice za prijavu

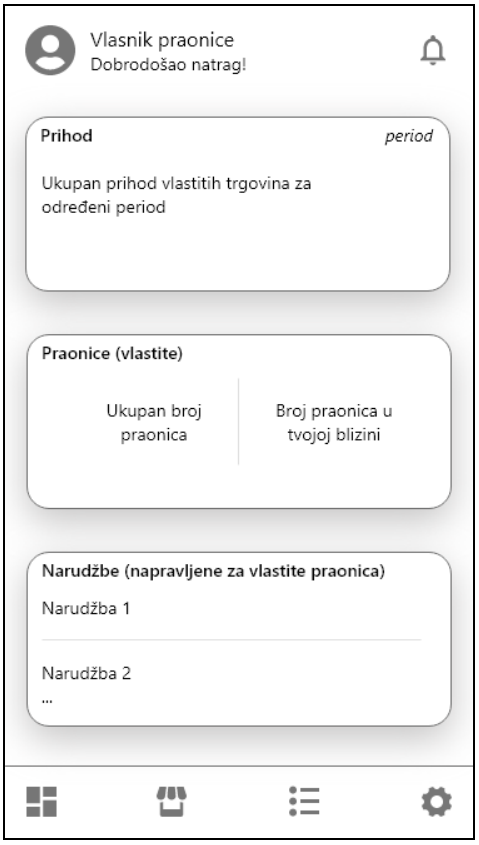
Slika 5.4 Planirani izgled stranice za prijavu zaboravljene zaporke

Slika 5.5 Planirani izgled stranice za promjenu zaporke

Uspješnom prijavom korisnika aplikacija zamišljeno je da se prelazi na dinamički dio aplikacije koji se također sastoji od nekoliko stranica. Prvo na što bi korisnici trebali naići jest nadzorna ploča koja bi sadržavala sve bitne informacije vezane za sam korisnički profil. Svaka od uloga (administrator, vlasnik praonice i korisnik) trebala bi imati različito korisničko sučelje. Tako bi npr. administrator trebao imati uvid na broj svih korisnika, narudžbi i praonica, vlasnik praonice trebao bi imati uvid na prihode poslovanja svojih trgovina, broj trgovina te narudžbe vezane za vlastite praonice dok bi korisnik imao uvid na registrirane praonice te vlastite narudžbe.



Slika 5.6 Planirani izgled nadzorne ploče administratora



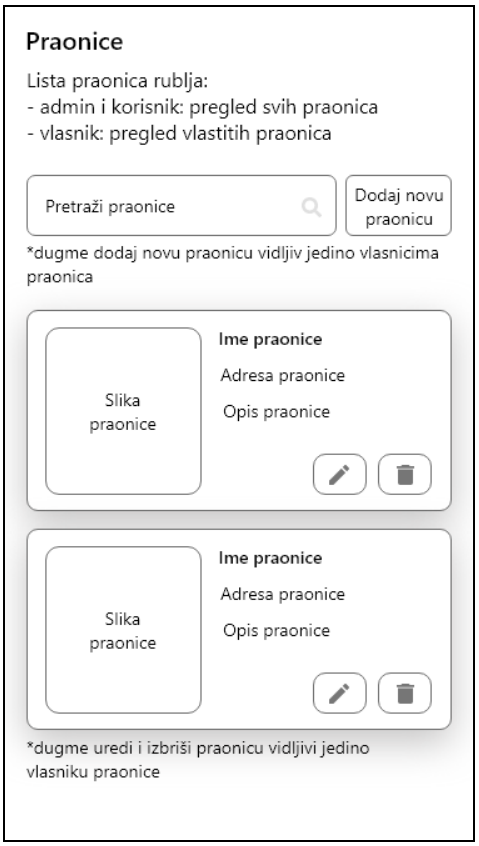
Slika 5.7 Planirani izgled nadzorne ploče vlasnika praonice

Slika 5.8 Planirani izgled nadzorne ploče korisnika

Unutar navigacijske trake aplikacije vidimo da se neke od stranica ponavljaju za sve tipove uloga. Stranice na koje svi imaju pristup jesu postavke, narudžbe i praonice. Stranica postavki služila bi za uređivanje osobnih podataka, definiranje funkcionalnosti značajki notifikacija, odabir jezika te samu odjavu korisnika.

Slika 5.9 Planirani izgled stranice postavki

Stranica praonice sadržavala bi listu svih registriranih praonica. Korisničko sučelje zamišljeno je jednako za sve uloge dok bi se dodatna provjera i filtriranje praonica za određene tipove uloga provodila na strani servera. Stranica bi također trebala biti podijeljena na nekoliko pod-stranica koje bi pružale detaljan uvid u dodatne informacije o praonici, njihovoj ponudi proizvoda te detaljan uvid u sam proizvod. Klik na kartice praonica i proizvoda vodio bi na novu stranicu pregleda samog entiteta. Iako je za ove pod-stranice također zamišljeno jedinstveno korisničko sučelje, svaka od njih trebala bi imati dinamički prikazane komponente u ovisnosti o ulozi korisnika. Tako bi npr. vlasnik praonice trebao imati mogućnost dodavanja, brisanja i uređivanja praonica i proizvoda, dok bi korisnik trebao imati mogućnost dodavanja proizvoda u košaricu.

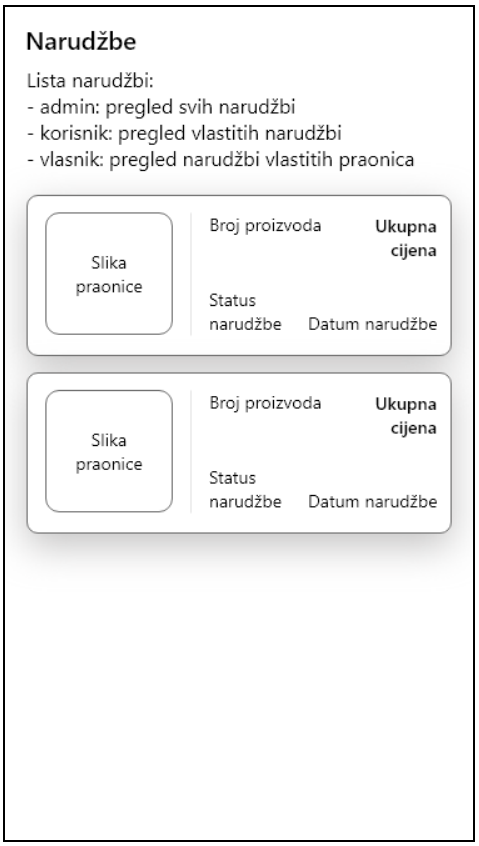


Slika 5.10 Planirani izgled stranice praonica

Slika 5.11 Planirani izgled stranice određene praonice

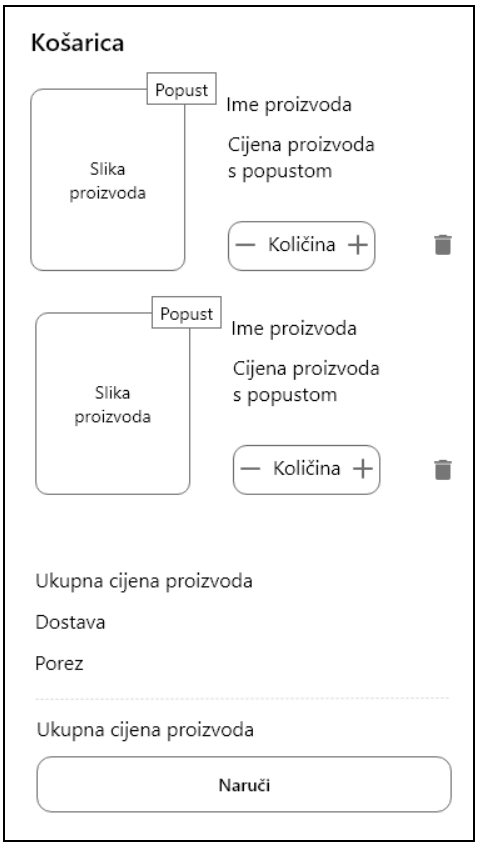
Slika 5.12 Planirani izgled stranice određenog proizvoda

Stranica narudžbe slično kao i stranica praonica trebala bi sadržavati listu svih narudžbi gdje bi se dodatna provjera i filtriranje narudžbi za određene uloge odvijala na strani servera. Tako bi administrator trebao imati na uvidu sve narudžbe, vlasniku praonica bile bi vidljive samo narudžbe koje su napravljene unutar njegove praonice dok bi korisnik trebao vidjeti jedino vlastite narudžbe.



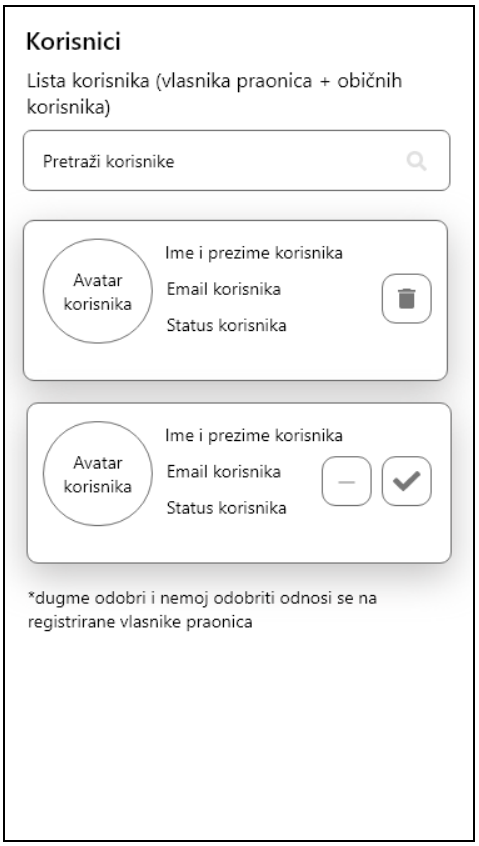
Slika 5.13 Planirani izgled stranice narudžbi

Košarica je stranica koja bi služila za pregled željenih proizvoda za narudžbu te bi bila vidljiva jedino korisnicima. Unutar nje trebalo bi biti moguće mijenjati količinu samih proizvoda te po potrebi omogućiti brisanje proizvoda unutar košarice.



Slika 5.14 Planirani izgled stranice košarica

Za sam kraj zamišljeno je omogućiti administratoru pregled svih korisnika bilo oni vlasnici praonica ili ne. Unutar stranice korisnika administrator bi trebao imati mogućnosti uređivanja i brisanja. Brisanjem samog korisnika ako je on vlasnik praonice trebale bi se izbrisati i sve praonice koje je registrirao, a samim time i sve proizvode koje je praonica posjedovala.



Slika 5.15 Planirani izgled stranice korisnika

5.1.3. Analiza korisničkih uloga i dozvola

Osnovni koncept kojim smo se vodili prilikom određivanja korisničkih uloga i pripadajućih dozvola bio je osnovni model kontrole pristupa temeljen na ulogama, gdje je svako pojedinoj ulozi pridodana dozvola za odgovarajuću operaciju. Glavni razlog zbog čega nije korišten hijerarhijski model kontrole pristupa jest taj što nismo htjeli administratoru omogućiti određene radnje poput dodavanja proizvoda u košaricu i izvršavanja narudžbi.

Terminologija uloga kojom ćemo se voditi u nastavku teksta definirana je ovako:

- administrator
- vlasnik (engl. *shop owner/service*) – ponuditelj usluga
- korisnik (engl. *user*) – naručitelj usluga

Popis svih dozvola, njihov opis i pripadnost određenoj ulozi, odnose uloge koje imaju dozvole za obavljati određene operacije prikazani su sljedećoj tablici.

DOZVOLA	OPIS DOZVOLE	ULOGA
Upravljanje aplikacijom	Pregled nadzorne ploče, uređivanje osobnih podataka	ADMINISTRATOR VLASNIK KORISNIK
Upravljanje korisnicima	Čitanje, uređivanje i brisanje korisnika	ADMINISTRATOR
Čitanje praonica	Čitanje praonica	ADMINISTRATOR VLASNIK KORISNIK
Kreiranje praonica	Kreiranje praonice	VLASNIK
Uređivanje praonice	Uređivanje praonice	VLASNIK
Brisanje praonice	Brisanje praonice	VLASNIK
Čitanje proizvoda	Čitanje proizvoda određene praonice	ADMINISTRATOR VLASNIK KORISNIK
Kreiranje proizvoda	Kreiranje proizvoda unutar određene praonice	VLASNIK
Uređivanje proizvoda	Uređivanje proizvoda	VLASNIK
Brisanje proizvoda	Brisanje proizvoda	VLASNIK
Čitanje narudžbi	Čitanje narudžbi	ADMINISTRATOR VLASNIK KORISNIK
Kreiranje narudžbe	Kreiranje narudžbe	KORISNIK
Uređivanje statusa narudžbe	Uređivanje statusa narudžbe (na čekanju, prihvaćena, …)	VLASNIK

Upravljanje košaricom	Pregled košarice, dodavanje i brisanje proizvoda	KORISNIK

Tablica 2 Popis uloga i dozvola

5.2. Korištene tehnologije i alati

5.2.1. JavaScript

JavaScript je programski jezik visoke razine koji se pravovremeno kompajlira (engl. *just-in-time*, skraćeno JIT) i koji po svojoj klasifikaciji podržava mnoge programske paradigme kao što su programiranje vođeno događajem (engl. *event driven*), funkcionalno i imperativno programiranje, u koje spadaju proceduralno i objektno – orijentirano programiranje [19]. JIT kompajliranje predstavlja prevođenje programa u jezik koji računalo razumije za vrijeme izvršavanja programa. Web preglednici sadržavaju ugrađene dijelove programa (engl. *engine*) nazvane *JavaScript virtual machine*. U preglednicima Chrome i Opera to bi bio V8, dok je u Firefoxu nazvan SpiderMonkey. Proces rada teče u par koraka. U prvom koraku e*ngine* čita (parsira) skriptu. Zatim prevodi (kompajlira) skriptu u strojni jezik te se naposljetku strojni jezik izvršava.

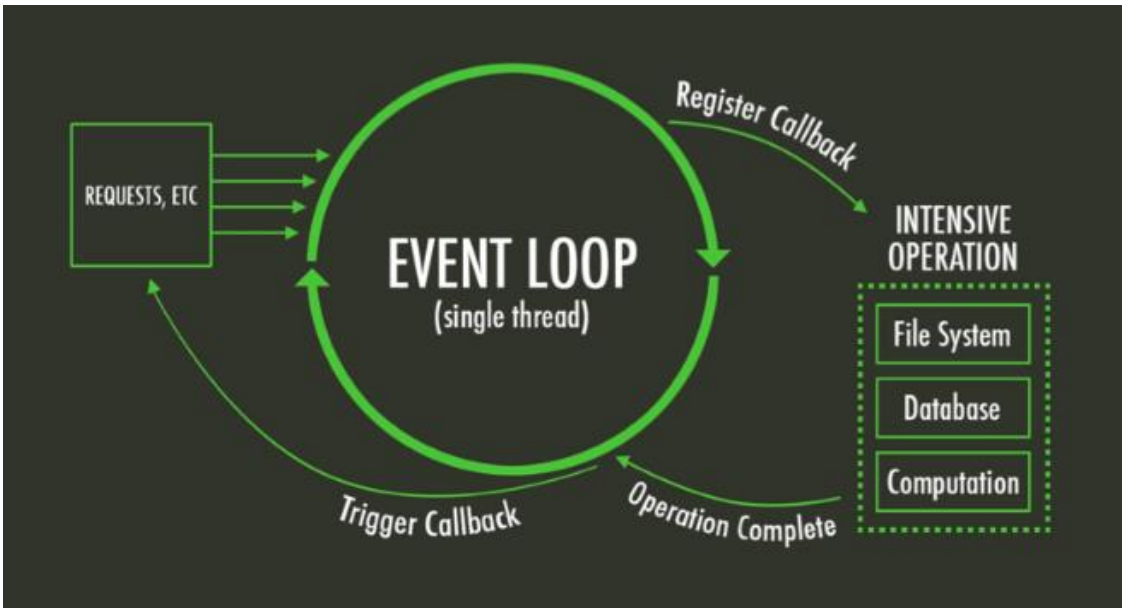
Standard za JavaScript je ECMAScript. Od 2012. svi moderni preglednici u potpunosti podržavaju ECMAScript 5.1. ECMA International je 17. lipnja 2015. objavila šestu glavnu verziju ECMAScript-a, koja se službeno naziva ECMAScript 2015, a koja se u početku nazivala ECMAScript 6 ili ES6. ES6 predstavlja revoluciju u standardizaciji jezične strukture ovog programskog jezika. Uvedene su klasne deklaracije, moduli, iteratori i *for...of* petlje, definiranje funkcijskih izraza pomoću *arrow* operatora, ključne riječi *let* i *const*, nove kolekcije poput mapa i setova te slično. Nadalje od sljedećih verzija bitno je spomenuti uvođenje *async/await* konstrukcija s kojima se koristimo kod asinkronog programiranja [20].

Iako je ovaj skriptni jezik najpoznatiji kao jezik web preglednika i razvoja klijentske strane, ugrađivanje engine-a na strani poslužitelja omogućuje korištenje JavaScripta za potpunu izgradnju aplikacije. U izradi ovog diplomskog rada koristili smo prvenstveno JavaScript. Na strani poslužitelja koristili smo Node.js i Express.js razvojni okvir (engl. *framework*), dok smo na klijentskoj strani koristili programsku knjižnicu (engl. *library*) React.

5.2.2. Tehnologije korištene na strani poslužitelja

5.2.2.1 Node.js

Node.js je izvršavajuće okruženje (engl. *runtime environment*) na strani poslužitelja izgrađeno na Chrome JavaScriptu V8 *engine-u*. Ono što Node.js čini učinkovitim jest neblokirajući *event-driven* ulazno/izlazni model (engl. *non-blocking I/O*). Neblokirajući I/O predstavlja nam glavnu i jedinu nit (eng. *single-thread*) koja nije blokirana I/O operacijama, već poslužitelj konstantno prati i zaprima zahtjeve svih korisnika [21]. Cijela magija ovog procesa obrade događa se unutar petlje događaja (engl. *event loop*). Ona nam predstavlja beskonačnu petlju i jedinu nit unutar Node-a. Kad *event loop* treba izvršiti I/O operaciju ona koristi nit iz bazena operacijskog sustava (pomoću *library-a* „libuv"), dok se povratni poziv (engl. *callback*) kad je posao završen postavlja u red koji čeka na svoje izvršavanje [21].



Slika 5.16 Node.js Event Loop

Node.js najčešće se koristi u aplikacijama zadužene za prijenos podataka, aplikacijama u stvarnom vremenu, aplikacijama zasnovanim na JSON aplikacijskom programskom sučelju te jednostraničnim web aplikacijama (engl. *Single-page application*, skraćeno SPA). Nekoliko važnih značajki koje Node.js čine prvim izborom programskih arhitekata jesu:

- asinkronost – poslužitelj konstantno prati i zaprima zahtjeve svih korisnika
- brzina – izgrađen na Chrome JavaScriptu V8 *engine-u*

- skalabilna dretva (nit) – može zaprimiti i obraditi puno veći broj zahtjeva od tradicionalnih HTTP poslužitelja
- međuspremnik – poslužitelj ne pohranjuje podatke u međuspremnik već ih iznosi u dijelovima

5.2.2.2 Express.js

Express.js ili Express je mrežni okvir za Node.js koji je dizajniran za izradu mrežnih aplikacija i API-ja. Spada u standardni poslužiteljski framework za Node.js. Autor TJ Holowaychuk opisao ga je kao poslužitelj nadahnut Sinatrom (Ruby web programska knjižnica namijenjena kao alternativa za razvojne okvire poput Ruby on Rails-a [22]). Iako je sam Express prilično minimalistički, stvoreni su kompatibilni paketi s kojim se rješava gotovo svaki problem razvoj web aplikacija. Nekoliko osnovnih značajki Express.js razvojnog okvira jesu:

- pojednostavljeno rukovanje zahtjevima uvođenjem tzv. *middleware* funkcija koje se izvršavaju tijekom zahtjeva
- pojednostavljena integracija *template engine-a* poput handlebars-a, ejs-a
- pojednostavljena integracija s bazama podataka
- posluživanje statičkih dokumenta

Na sljedećem dijelu koda vidimo primjer jednostavne Express aplikacije gdje ruta ispisa poruke odgovara mrežnoj adresi lokalnog poslužitelja s definiranim portom.

```
const express = require("express");
const app = express();
const port = 3000;
app.get("/", (req, res) => {
  res.send("Hello World");
});
app.listen(port, () => {
  console.log("Server listening on port ", port)
});
```

Kôd 1 Primjer Express aplikacije

5.2.2.3 Sustav za upravljanje bazom podataka MySQL

MySQL, skraćeno od *My Structured Query Language*, najpoznatiji je i najrašireniji sustav otvorenog koda (engl. *open source*) za upravljanje bazama podataka razvijen od Oracle korporacije. MySQL upravlja relacijskim bazama podataka (engl. *Relational Database Management System*, skraćeno RDBMS) koje pohranjuju zbirku podataka iz različitih izvora. MySQL također pruža implementaciju za upitni jezik (engl. *Structured Query Language*, skraćeno SQL) dizajniran za upravljanje podacima koji se nalaze u bazi podataka.

RDBMS	**NoSQL**	**RAZLIKA**
Tablica	Kolekcija	U relacijskoj bazi podataka tablica sadrži stupce i retke, dok je u nerelacijskoj bazi ista struktura poznata kao kolekcija sastavljena od dokumenata.
Redak	Dokument	U relacijskoj bazi podataka redak predstavlja jedinstvenu strukturiranu podatkovnu stavku u tablici, dok su podaci u nerelacijskoj bazi pohranjeni u dokumentima u obliku BSON formata.
Stupac	Polje	U relacijskoj bazi podataka stupac označava skup vrijednosti podataka, koji su u nerelacijskoj bazi poznati kao polja.

Tablica 3 Razlike relacijskih i nerelacijskih baza podataka

Samo neke od prednosti koje su nas navele na odabir MySQL bazu su brzina, pouzdanost, mogućnost rada na različitim platformama te podrška za veliku količinu programskih jezika. MySQL kod je u cijelosti napisan od nule s ciljem postizanja što boljih izvedbi. Također MySQL pruža mogućnost rada na različitim operacijskim sustavima kao što su Linux, Solaris, FreeBSD, MacOS, Windows, Irix i drugi. Povezivanje i komunikacija s bazom podataka moguća je s mnoštvom programskih jezika primjerice PHP, Java, Ruby, Python, C/C++ itd. Glavni nedostatak odnosno izazov MySQL baze je skalabilnost. Izvorno je dizajniran kao sustav s jednim čvorom, a ne s konceptom modernog podatkovnog centra. Današnje najveće MySQL instalacije ne mogu se skalirati korištenjem MySQL-a kao jednog sustava i moraju se oslanjati na dijeljenje ili dijeljenje skupa podataka na više čvorova ili instanci. Ipak, za manje web aplikacije s manjom količinom

podataka i tablica, primjerice lokalnih eCommerce trgovina MySQL baza čini se kao izvrstan izbor.

5.2.2.4 Sequelize ORM

Objektno relacijski *mapper* (engl. *Object Relational Mapper*, skraćeno ORM) je alat, odnosno razina apstrakcije koja preslikava podatke u relacijskoj bazi podataka u objekte kojima dalje možemo manipulirati pomoću programskog jezika. ORM-ovi postoje isključivo za mapiranje detalja između dva izvora podataka koji zbog neusklađenosti ne mogu postojati zajedno. Glavna od prednosti upotrebe ORM-a jest ušteda vremena pri pisanju neobrađenih upita strukturnog upitnog jezika (engl. *Structured Query Language*, skraćeno SQL) pritom smanjujući vrijeme razvoja aplikacije. ORM-ovi također pomažu u sprječavanju SQL injekcija te omogućuju apstrakciju operacija trajne pohrane unutar baze podataka [23].

Sequelize je jedan od najpoznatijih ORM-ova, kako za MySQL, tako i za mnoge druge tipove baza (PostgreSQL, SQLite, Microsoft SQL, itd.). Osim navedenih prednosti *Sequelize* također pruža podršku i za transakcije, relacije između modela baze podataka, željno (engl. *eager*) učitavanje, validaciju modela i replikaciju čitanja.

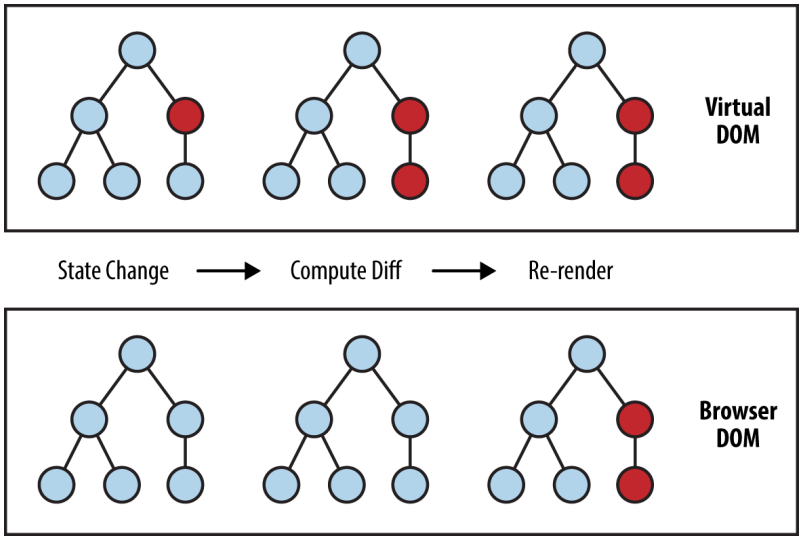
5.2.3. Tehnologije korištene na strani klijenta

5.2.3.1 React

React je JavaScript biblioteka specijalizirana za pomoć u izradi korisničkih sučelja te predstavlja prikaz (engl. *view*) unutar MVC (engl. *Model-View-Controller*) strukture web aplikacije. Prije React-a korisnička sučelja razvija su se ručno pomoću običnog *vanilla* JavaScript-a ili s bibliotekama koje i nisu bile toliko usmjerene na korisničko sučelje poput jQuery-a. To je značilo dulje vrijeme razvoja i obilje mogućnosti za pogreške. Tako je 2011. godine inženjer na Facebook-u Jordan Walke kreirao React. Glavni cilj React-a bio je olakšati manipulaciju nad DOM-om te ubrzati razvoj korisničkih sučelja pomoću osnovnih gradivnih elemenata, komponenti (engl. *components*) [24].

React ima dvije ključne značajke koje ga odlikuju: JSX i Virtual DOM. JSX (engl. *JavaScript eXtension*) je proširenje koje web programerima olakšava modifikaciju svog DOM-a pomoću jednostavnog koda u stilu HTML-a. Ako ne upotrebljavamo React, web aplikacija za ažuriranje svog DOM-a upotrijebit će HTML. Učitavanje cijelog DOM-a za

dinamične web stranice koje uključuju tešku korisničku interakciju može postati problem. React je taj problem riješio stvaranjem Virtual DOM-a. Virtualni DOM je kopija DOM-a web lokacije, a React je koristi da vidi koje dijelove stvarnog DOM-a treba promijeniti kad određeni događaj nastupi (npr. korisnik klikne gumb).



Slika 5.17 Virtualni DOM

Kao što smo već spomenuli, React je baziran na komponentama koje upravljaju vlastitim stanjima koje poslije komponiramo kako bi napravili složena korisnička sučelja. Komponente su poput JavaScript funkcija, one prihvaćaju proizvoljne ulaze (engl. *props*) i vraćaju elemente koji određuju što će se pojaviti na ekranu. Komponente možemo definirati pomoću funkcija i klasa. Osnovna razlika između takvih komponenti jest bila ta da funkcionalne komponente prvotno nisu mogle sadržavati stanja no uvođenjem kuka (engl. *hooks)* to se promijenilo. *Hook useState* omogućila je promjenu vrijednosti stanja komponente pomoću funkcije `setState((previousState) => newState).`

Props ili svojstva i *state* ili stanje jesu zapravo JavaScript objekti čija promjena utječe na ponovni prikaz (engl. *render*) komponente. *Props* je zapravo način na koji komponente komuniciraju međusobno. Koriste se kako bi se podaci prebacili od roditelj komponente do komponente djeteta, te je protok podataka jednosmjeran.

Jedan od dodatnih koncepata React-a jesu metode životnog ciklusa (engl. *lifecycle methods*) koje su također postale dostupne u funkcionalnim komponentama pojavom *hooks-a*. Metode životnog ciklusa jesu metode koje se pokreću u određenom trenutku životnog ciklusa aplikacije te unutar njih možemo izvršiti neku željenu funkciju. Neke od njih se pokreću prije prikaza komponente, neke prilikom izmjene na komponenti, a neke

kada se komponente više ne prikazuje. *Hook useEffect* je jedinstvena metoda koja objedinjuje sva tri scenarija, u kojoj je moguće definirati parametre o kojima metoda ovisi.

5.2.3.2 React Router

React Router je nedvojbeno jedna od najboljih značajki i najkorištenijih biblioteka namijenjenih za React. U suštini biblioteka sadržava funkcionalnosti za dvije vrste aplikacija, web, koje koriste paket *react-router-dom* i mobilne, koje koriste paket *react-router-native*.

React u svojoj suštini stvara SPA aplikaciju, gdje ruter odigrava važnu ulogu u prikazivanju višestrukih pogleda (engl. *views*) bez potrebe za ponovnim učitavanjem web preglednika. Osnovne komponente React Router-a jesu sam ruter, primjerice `<BrowserRouter>`, *route matchers* kao što su `<Route>` i `<Switch>` te komponente navigacije u koje spadaju `<Link>` i `<Redirect>` [25].

- *BrowserRouter* – implementacija usmjerivača pomoću koje uvodimo usmjeravanje u React. Koristi HTML5 History API koji uključuje *pushState*, *replaceState* i *popState* događaje s ciljem sinkronizacije korisničkog sučelja s URL-om. To je nadređena komponenta koja se koristi za pohranu svih ostalih komponenti i koristi regularne URL putanje
- *Route* – uvjetna komponenta koja prikazuje komponentu na temelju definiranog URL-a. Drugim riječima, to je komponenta koja prikazuje neko korisničko sučelje kada se njegova putanja podudara s trenutnim URL-om
- *Switch* – komponenta prekidača koja je zadužena za prikaz samo prve rute koja odgovara lokaciji umjesto prikaza svih odgovarajućih ruta
- *Link* – komponenta veze koristi se za stvaranje poveznica na različite rute i implementira navigaciju unutar web aplikacije

Osim navedenih komponenti biblioteka sadrži i nekoliko kuka koje pružaju dodatne informacije o samoj putanji.

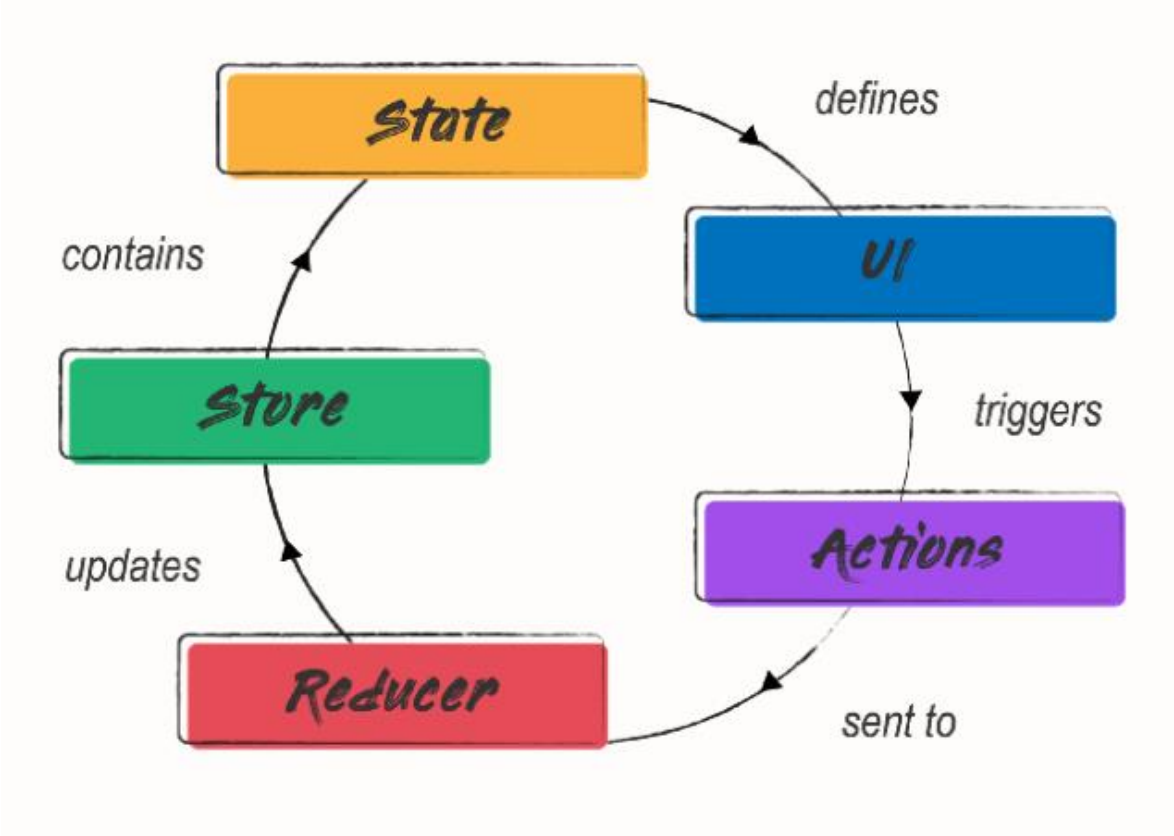
- *useHistory* – dopušta pristup instanci povijesti koju možemo koristiti za navigaciju. Preko objekta povijesti možemo pristupiti i manipulirati trenutnim stanjem povijesti preglednika
- *useLocation* – vraća objekt lokacije koji sadrži trenutnu putanju aplikacije

useParams – vraća objekt parova ključ/vrijednost URL parametara gdje je ključ naziv parametra, a vrijednost trenutna vrijednost parametra. Drugim riječima, omogućuje pristup parametrima pretraživanja unutar URL-a

5.2.3.3 Redux

Redux je spremnik predvidljivog stanja za JavaScript aplikacije. Redux možemo koristiti u kombinaciji s React-om ili s bilo kojom drugom bibliotekom prikaza. Opisujemo ga s tri temeljna principa a to su [26]:

- jedan izvor istine (engl. *single source of truth*) – globalno stanje aplikacije pohranjuje se u objektno stablo unutar jednog spremišta (engl. *store*)
- stanje se može samo čitati (engl. *read-only*) – jedini način da promijenimo stanje jest da emitiramo radnju (engl. *dispatch*), objekt koji opisuje što se dogodilo
- promjene se vrše čistim (engl. *pure*) funkcijama – reduktori su čiste funkcije koje preuzimaju prethodno stanje i radnju te vraćaju sljedeće stanje. Prilikom promjena važno je vratiti nove objekte stanja, umjesto mutacije prethodnog stanja



Slika 5.18 Tok Redux-a

U prvom koraku korisnik putem korisničkog sučelja izvrši radnju. Sama radnja okidač je za emitiranje akcije do reduktora. Unutar reduktora i podataka koje smo dobili iz objekta akcije ažurira se stanje samog *state-a.* Promjena stanja u konačnici definira izgled korisničkog sučelja.

5.3. Implementacija aplikacije

5.3.1. Poslužitelj

5.3.1.1 Kreiranje migracija

Kao što postoji Git, sustav kontrole verzija za upravljanje promjenama u izvornom kodu, za upravljanje promjenama unutar bazi podataka postoje migracije. Pomoću njih možemo prenijeti postojeću bazu podataka u drugo stanje i obrnuto. Ti prijelazi stanja spremaju se u datoteke migracije, koje opisuju kako doći do novog stanja i koje promjene je potrebno napraviti da biste se vratili u staro stanje.

Sequelize migracija je JavaScript datoteka koja izvozi dvije funkcije, *up* i *down*. Te funkcije definiramo ručno te pozivamo unutar Sequelize sučelja naredbenog retka pomoću naredbi `npx sequelize-cli db:migrate` i `npx sequelize-cli db:migrate:down`. Unutar tijela funkcija migracija komuniciramo s bazom podataka uz pomoć pripadajućeg korisničkog sučelja QueryInterface.

```
module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.createTable("Permissions", {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER,
      },
      title: {
        allowNull: false,
        type: Sequelize.STRING,
      },
      description: {
        type: Sequelize.STRING,
      },
```

```
      createdAt: {
        allowNull: false,
        type: Sequelize.DATE,
      },
      updatedAt: {
        allowNull: false,
        type: Sequelize.DATE,
      },
    });
  },

  down: async (queryInterface) => {
    await queryInterface.dropTable("Permissions");
  },
};
```

Kôd 2 Kreiranje migracije

Konkretno ova migracija služi za kreiranje nove tablice unutar baze podataka. Pomoću QueryInterface sučelja i njegovih naredbi kreiramo tablicu s imenom „Permissions“ te strukturu i tipove podataka njezinih stupaca. Za povrat u prethodno stanje baze unutar tijela funkcije potrebno je izbrisati tablicu s imenom „Permissions“.

5.3.1.2 Kreiranje seedersa

Pretpostavimo da želimo unutar baze podataka unijeti već predefinirane zadane podatke. U tom procesu pomažu nam *seeders*. Proces kreiranja i pokretanja njezinih funkcija jednak je kao i kod migracija. Pomoću naredbe `npx sequelize-cli seed:generate --name "seed name"` kreira se datoteka s predefiniranom strukturom i funkcijama za uvoz i izvoz Funkcije pozivamo pomoću naredbi `npx sequelize-cli db:seed:all`, odnosno `npx sequelize-cli db:seed:undo` zavisno o tome želimo li doći u novo stanje ili se vratiti na prethodno.

```
module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.bulkInsert(
      "Permissions",
      [
        {
          title: "MANAGE_APP",
```

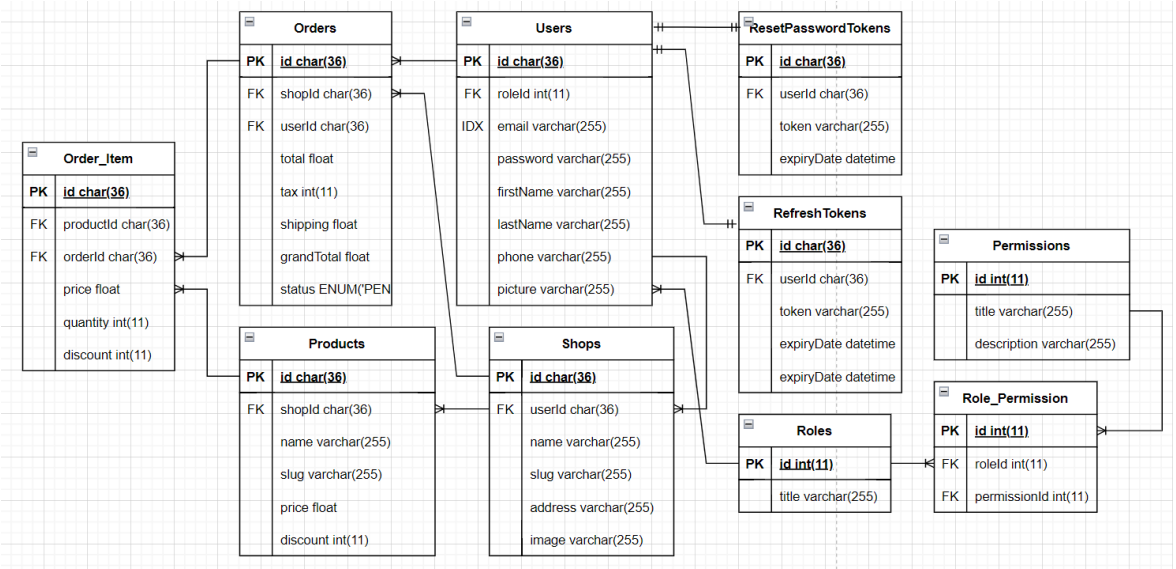
```
          description: "Can view dashboard, edit personal
settings",
          createdAt: new Date(),
          updatedAt: new Date(),
        },
        {
          title: "MANAGE_USERS",
          description: "Can read, update and delete users",
          createdAt: new Date(),
          updatedAt: new Date(),
        },
        …
      ],
      {}
    );
  },

  down: async (queryInterface, Sequelize) => {
    await queryInterface.bulkDelete("Permissions", null, {});
  },
};
```

Kôd 3 Kreiranje seedera

Konkretno ovaj *seeder* služi za popunjavanje tablice “Permissions” s danim podacima, odnosno brisanje svih podataka unutar istoimene tablice.

5.3.1.3 Prikaz strukture baze podataka

Sljedeća slika prikazuje strukturu baze podataka i vrstu relaciju između pojedinih tablica. Nama posebno zanimljive jesu relacije između tablica korisnika, uloga i dozvola. Kao što je i prethodno navedeno unutar poglavlja RBAC relacija između tablica uloga i dozvola je više-naprema-više. Izvedena je pomoću tablice spoja (engl. *join*) „Role_Permission“. Zbog jednostavnost implementacije relacija između tablice uloga i korisnika je jedan-naprema-više.



Slika 5.19 Struktura baze podataka

5.3.1.4 Upravljanje rutama

Način komunikacije i djelovanje koji se odvija između različitih komponenti računalnog sustava, u našem slučaju klijenta i poslužitelja, naziva se aplikacijsko programsko sučelje (engl. *application programming interface*, skraćeno API). Na strani poslužitelja izlažemo „usluge“ koje su klijentu dostupne putem HTTP protokola te vraćaju odgovarajuće podatke iz baze u JSON formatu.

```
const router = express.Router();

router.post(
  "/",
  authenticateUser,
  authorizeUser([Role.OWNER]),
  validateRequest(schema.createProduct),
  createProduct
);

router.get("/:shopId", authenticateUser, getProducts);

router.put(
  "/:id",
  authenticateUser,
  authorizeUser([Role.OWNER]),
  updateProduct
);
```

```
router.delete(
  "/:id",
  authenticateUser,
  authorizeUser([Role.OWNER]),
  deleteProduct
);
```

Kôd 4 Prikaz ruta vezanih uz proizvode praonica

METODA	PRISTUPNA TOČKA	SVRHA
POST	/api/auth/register	Registracija korisnika
POST	/api/auth/login	Prijava korisnika
POST	/api/auth/refresh-tokens	Generiranje *access* i *refresh* tokena
POST	/api/auth/request-reset-password	Generiranje verifikacijskog linka za generiranje nove lozinke
POST	/api/auth/reset-password	Izmjena lozinke
GET	/api/dashboard/users	Dohvaćanje *dashboard* podataka vezanih za korisnike
GET	/api/dashboard/shops	Dohvaćanje *dashboard* podataka vezanih za praonice
GET	/api/users	Dohvaćanje svih korisnika
DELETE	/api/users/:id	Brisanje određenog korisnika ili zahtjeva za registraciju
POST	/api/users/pending	Prihvaćanje korisničkog zahtjeva za registraciju
GET	/api/shops	Dohvaćanje svih praonica ovisno o korisničkoj ulozi (sve/vlastite)
POST	/api/shops	Kreiranje praonice

PUT	/api/shops	Uređivanje praonice
GET	/api/shops/:slug	Dohvaćanje određene praonice
DELETE	/api/shops/:id	Brisanje određene praonice
POST	/api/products	Kreiranje proizvoda
GET	/api/products/:shopId	Dohvaćanje svih proizvoda određene praonice
PUT	/api/products/:id	Uređivanje određenog proizvoda
DELETE	/api/products/:id	Brisanje određenog proizvoda
GET	/api/orders	Dohvaćanje svih narudžbi
POST	/api/orders	Kreiranje narudžbe

Tablica 4 Popis ruta na poslužitelju

5.3.1.5 Implementacija middleware funkcija

Express.js je mrežni okvir s minimalnom vlastitom funkcionalnosti čiji je koncept temeljen na nizu poziva funkcija preusmjeravanja (engl. *middleware*). *Middleware* funkcije imaju pristup objektu HTTP zahtjeva (*req*), objektu odgovora (*res*) i sljedećoj funkciji u ciklusu zahtjev-odgovor aplikacije. Sljedeća funkcija obično se označava varijablom pod nazivom *next*. Funkcije mogu izvršavati sljedeće zadatke:

- izvršiti kod
- napraviti promjene na objektima zahtjeva i odgovora
- završiti ciklus zahtjev-odgovor
- pozvati sljedeću funkciju u stogu (engl. *stack*) funkcija

Unutar poslužitelja kreirane su tri prilagođene funkcije, funkcija za autentifikaciju korisnika, autorizaciju korisnika i validaciju zahtjeva poslanog od strane korisnika, koje se izvršavaju po potrebi prilikom HTTP zahtjeva prije no što upravljači preuzmu svoj dio posla.

```
const authenticateUser = async (req, res, next) => {
  const header = req.headers.authorization ?? "";
```

```
  const token = header.split(" ")[1];

  if (token) {
    try {
      const decoded = await verifyAccessToken(token);
      req.decoded = decoded;
      return next();
    } catch (err) {
      return res.status(401).send(
        "Failed to authenticate   token!"
      );
    }
  }

  return res.status(401).send("No provided token!");
};
```

Kôd 5 Funkcija za autentikaciju korisnika

```
const authorizeUser = (roles = []) => (req, res, next) => {
  if (roles.length && !roles.includes(req.decoded.roleId)) {
    res.status(403).send("Unauthorized!");
  }

  next();
};
```

Kôd 6 Funkcija za autorizaciju korisnika

```
const validateRequest = (schema) => (req, res, next) => {
  const options = {
    abortEarly: false,
  };

  const { error, value } = schema.validate(
    req.body, options
  );

  if (error) {
    res.status(422).send({
```

```
      error: {
        message: "Invalid input. Please try again!",
      },
    });
  } else {
    req.body = value;
    next();
  }
};
```

Kôd 7 Funkcija za validaciju korisničkog zahtjeva

5.3.1.6 Implementacija toka prijave korisnika

Prijavom korisnika započinje proces kontrole pristupa i kao takva jedna je od najvažnijih funkcija unutar poslužitelja. Započinje slanjem zahtjeva korisnika na odgovarajuću rutu.

```
router.post(
  "/login",
  validateRequest(schema.login),
  authController.loginUser
);
```

Kôd 8 Ruta za prijavu korisnika

Prije negoli upravljač preuzme zahtjev na obradu, poziva se *middleware* za validaciju zahtjeva. Argument kojeg prima funkcija validacije je *joi* objekt. Biblioteka joi jest jezik opisa sheme i validator podataka namijenjen za JavaScript. Očekivani ulaz korisnika definiran je kao objekt s potrebnim poljima email i password od kojeg se očekuje duljina od osam znakova te korišteni znakovi velikih i malih slova i brojeva.

```
login: joi.object({
    email: joi.string().required().email(),
    password: joi
      .string()
      .required()
      .regex(/^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9]).{8,}$/),
  })
```

Kôd 9 Shema objekta za validaciju prijave korisnika

Uspješnom validacijom dolazimo do upravljača koji poslane parametre korisnika prosljeđuje servisu za prijavu korisnika. Ako servis uspješno obavi prijavu, korisniku se vraćaju podaci vezani uz sami korisnički profil i *access token*. Također, unutar kolačića

šalje se i *refresh token* potreban za osvježavanje *access token-a*. U ovom trenutku s krajnjim korisnikom uspostavlja se sesija. Razlog slanja i spremanja tokena na dva različita mjesta (*local storage* i kolačići) jest zaobilaženje mogućnosti kompromitacije oba tokena jednim uspješnim napadom.

```
const loginUser = async (req, res, next) => {
  try {
    const { user, accessToken, refreshToken } =
      await authService.loginUser(
        req.body
      );

    res.cookie("refresh-token", refreshToken, {
      httpOnly: true,
      secure: true,
    });

    return res.status(200).json({ user, accessToken });
  } catch (err) {
    return next({
      status: 400,
      message: err,
    });
  }
};
```

Kôd 10 Upravljač prijave korisnika

Servis za prijavu korisnika zadužen je za provjeru korisničkih podataka, te kreiranje već spomenutih tokena. Unutar funkcije koriste se tri odvojene funkcije: funkcije za usporedbu korisničke zaporke i *hashirane* zaporke unutar baze podataka, funkcije za kreiranje JWT access tokena i funkcije za kreiranje refresh tokena.

```
const checkPassword = async (password, hash) => {
  try {
    const isMatching = await bcrypt.compare(password, hash);
    return isMatching;
  } catch (err) {
    throw Error("Failed to check password.", err);
  }
};
```

Kôd 11 Funkcija za usporedbu zaporki

```
const createAccessToken = (user) => {
  const { id, roleId, email } = user;
  const payload = {
    id,
    roleId,
    email,
  };

  const token = jwt.sign(payload, process.env.SECRET_KEY, {
    expiresIn: "15m",
  });

  return token;
};
```

Kôd 12 Funkcija za kreiranje JWT access token-a

```
const createRefreshTokenPayload = (userId) => {
  const expiredAt = new Date();
  expiredAt.setSeconds(expiredAt.getSeconds() + 86400);

  const refreshTokenUUID = uuidv4();

  return {
    token: refreshTokenUUID,
    expiryDate: expiredAt.getTime(),
    userId,
  };
};
```

Kôd 13 Funkcija za kreiranje refresh-token-a

```
const loginUser = async (params) => {
  try {
    const { email, password: paramPassword } = params;

    const user = await User.findOne({
      where: { email },
      attributes: { exclude: "RoleId" },
      include: [
```

```
    {
      model: Role,
      as: "role",
      attributes: ["title"],
      include: [
        {
          model: Permission,
          as: "permissions",
          attributes: ["title"],
          through: {
            attributes: [],
          },
        },
      ],
    },
  ],
});

const passwordMatch = await checkPassword(
  paramPassword,
  user.password
);

if (!user || !passwordMatch) {
  throw Error(
    "Invalid email or password! Please try again!"
  );
}

if (user.status === "PENDING") {
  throw Error("Account has not been approved yet!");
}

const accessToken = createAccessToken(user);

const refreshToken = await RefreshToken.findOne({
  where: { userId: user.id },
});

if (refreshToken) {
  await refreshToken.destroy();
```

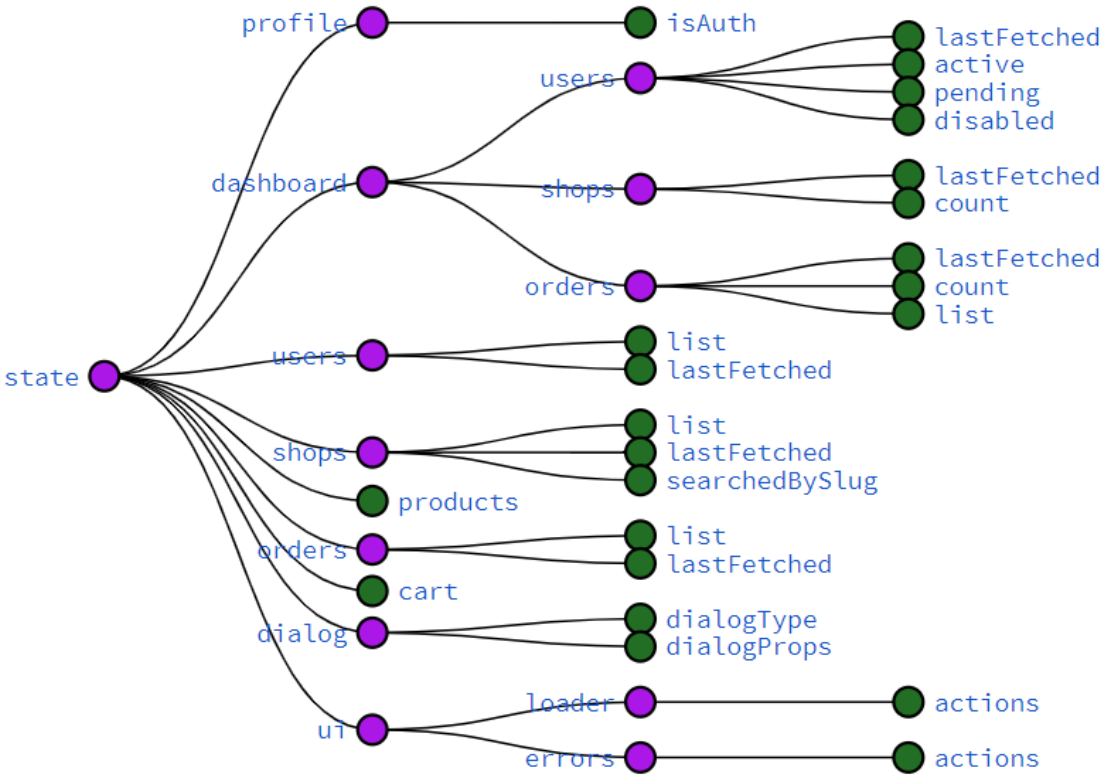
```
      }
      const newRefreshToken = await RefreshToken.create(
        createRefreshTokenPayload(user.id)
      );

      const {
        id,
        password,
        passwordResetToken,
        ...rest
      } = user.dataValues;

      return {
        user: rest,
        accessToken,
        refreshToken: newRefreshToken.token
      };
    } catch (err) {
      throw err.message || "Failed to login user!";
    }
  };
```

Kôd 14 Servis prijave korisnika

5.3.2. Klijent

5.3.2.1 Upravljanje stanjima

Manipulacija globalnim stanjima i njihovo prosljeđivanje odvija se pomoću biblioteke Redux. Osnovni elementi Redux-a jesu akcije (engl. *actions*), reduktori (engl. *reducers*) i skladište podataka (engl. *store*). Vrijednost stanja definirana je običnim objektom s ugniježđenim podacima unutar njega. Na sljedećoj slici vidim početni prikaz globalnog stanja aplikacije.



Slika 5.20 Redux state

S obzirom na strukturu stabla možemo zaključiti da je unutar aplikacije definirano nekoliko vrsta reduktora: profile, dashboard, users, shops, orders, dialog i ui. Reduktori, po definiciji, su funkcije koje za argumente primaju stanje i akciju koja se izvodi, a vraćaju rezultat novog stanja. Pri tom nije im dopušteno mijenjati postojeće stanje već moraju vratiti nove objekte stanja.

```
const INITIAL_STATE = {
  list: [],
  lastFetched: null,
};

const orders = (state = INITIAL_STATE, action) => {
  switch (action.type) {
    case FETCH_ORDERS:
      return {
        ...state,
        list: action.payload.orders,
        lastFetched: action.payload.fetchTime,
      };
```

```
    case CREATE_ORDER:
      return {
        ...state,
        list: [action.payload.order, ...state.list],
      };
    case EDIT_ORDER_STATUS:
      return {
        ...state,
        list: state.list.map((order) =>
          order.id !== action.payload.orderId
            ? order
            : {
                ...order,
                ...action.payload.data,
              }
        ),
      };
    default:
      return state;
  }
};
```

Kôd 15 Orders reduktor

Akcije su objekti s pripadajućim poljem koji definira tip akcije. Možemo ih zamisliti kao događaje koji opisuju što se dogodilo unutar aplikacije. U nastavku teksta dan je primjer akcije za dohvaćanje narudžbi.

```
const fetchOrders = (orders) => ({
  type: FETCH_ORDERS,
  payload: {
    orders,
    fetchTime: new Date().getTime(),
  },
});
```

Kôd 16 Akcija za dohvaćanje narudžbi

U svrhu lakšeg ažuriranja globalnog stanja s podacima vraćenim iz asinkronih HTTP zahtjeva koristili smo Redux Thunk biblioteku. Redux Thunk omogućuje pozivanje kreatora radnji koji vraćaju funkciju umjesto objekta akcije. Ta funkcija prima ugrađenu

metodu *dispatch*, koja se zatim koristi za slanje sinkronih akcija unutar samog tijela funkcije. U nastavku teksta prikazan je API *thunk* za dohvaćanje svih narudžbi.

```
const fetchOrders = (actionName) => async (dispatch) => {
  try {
    dispatch(uiActions.startUILoader(actionName));
    const { data } = await httpClient.get("/orders");
    dispatch(orderActions.fetchOrders(data));
  } catch (err) {
    dispatch(uiActions.setUIError(actionName));
  } finally {
    dispatch(uiActions.stopUILoader(actionName));
  }
};
```

Kôd 17 Redux thunk za dohvaćanje narudžbi

Bitno je još napomenuti da Redux pruža ugrađene kuke (engl. *hooks*) za lakšu manipulaciju operacijama nad samim stanjem. Najpoznatije od njih su `useDispatch()` koji koristimo za emitiranje određene akcije ili *thunk-a* te `useSelector()` koja nam daje pristup objektu stanja.

5.3.2.2 Upravljanje rutama

React sam po sebi nema ugrađenu podršku za upravljanje rutama te je korištena biblioteka React Router. S obzirom na veliki broj zamišljenih stranica, zamišljen je koncept ugniježđenih ruta gdje bi svaka nova razina naslijedila globalne komponente definirane u prethodnoj razini te dodala nove po potrebi.

RUTA	SVRHA PRIPADAJUĆE STRANICE	POTREBNA ULOGA
/	Prikaz početne stranice	/
/auth/sign-in	Prijava korisnika	/
/auth/sign-up	Registracija korisnika	/
/auth/forgot-password	Zahtjev za izmjenom zaporke	/
/auth/reset-passoword	Izmjena zaporke	/

/app	Prikaz nadzorne ploče	ADMINISTRATOR VLASNIK KORISNIK
/app/users	Prikaz korisnika	ADMINISTRATOR
/app/shops	Prikaz praonica	ADMINISTRATOR VLASNIK KORISNIK
/app/shops/create	Kreiranje nove praonice	VLASNIK
/app/shops/edit	Uređivanje postojeće praonice	VLASNIK
/app/shops/:shopSlug	Prikaz određene praonice	ADMINISTRATOR VLASNIK KORISNIK
/app/shops/:shopSlug/create	Kreiranje proizvoda	VLASNIK
/app/shops/:shopSlug/edit	Uređivanje proizvoda	VLASNIK
/app/shops/:shopSlug/:productSlug	Prikaz određenog proizvoda	ADMINISTRATOR VLASNIK KORISNIK
/app/orders	Prikaz narudžbi	ADMINISTRATOR VLASNIK KORISNIK
/app/cart	Prikaz košarice	KORISNIK
/app/settings	Prikaz postavki s navigacijom	ADMINISTRATOR VLASNIK KORISNIK

/app/settings/account	Prikaz i izmjena postavki korisničkog računa	ADMINISTRATOR VLASNIK KORISNIK
/app/settings/security	Prikaz i izmjena sigurnosnih postavki računa	ADMINISTRATOR VLASNIK KORISNIK
/app/settings/notifications	Prikaz i izmjena postavki notifikacija	ADMINISTRATOR VLASNIK KORISNIK
/app/settings/language	Prikaz i izmjena postavki jezika	ADMINISTRATOR VLASNIK KORISNIK
/app/settings/logout	Odjava korisnika	ADMINISTRATOR VLASNIK KORISNIK
/unauthorized	Prikaz stranice u slučaju neautorizirane radnje	/
*	Prikaz stranice u slučaju nepostojećeg URL-a	/

Tablica 5 Popis ruta na klijentu

```
const App = () => (
  <Router>
    <Switch>
      <Route exact path="/" component={Home} />
      <Route
        path="/auth"
        render={({ match }) =>
```

```
          <AuthRoutes basePath={match.path} />
        }
      />
      <Route
        path="/app"
        render={({ match }) =>
          <AppRoutes basePath={match.path} />
        }
      />
      <Route path="/unauthorized" component={NotAuthorized}
/>
      <Route component={NotFound} />
    </Switch>
  </Router>
);

const APP_ROUTES = [
  {
    path: "",
    name: "Dashboard",
    iconName: "dashboard",
    component: Dashboard,
    rule: RULES.MANAGE_APP,
    exact: true,
  },
  {
    path: "/users",
    name: "Users",
    iconName: "groups",
    component: Users,
    rule: RULES.MANAGE_USERS,
    exact: false,
  },
  …
];

const AppRoutes = ({ basePath }) => (
  <AppLayout>
    <Switch>
      {APP_ROUTES.map((route) => (
        <PrivateRoute
```

```
            key={route.path}
            path={`${basePath}${route.path}`}
            component={route.component}
            rule={route.rule}
            exact={route.exact}
          />
        ))}
      </Switch>
    </AppLayout>
  );
```

Kôd 18 Implementacija rutera

5.3.2.3 Implementacija komponente zaštićene rute

Zaštićene rute, ponekad nazivane i privatne, zahtijevaju od korisnika da bude ovlašten za pristup određenoj stranici. Cilj njihove implementacije jest zaštita od napada prisilnim pregledavanjem gdje napadač pokušava zaobići definirane mjere kontrole pristupa pogađanjem URL-a stranice. Iako komponenta djeluje kao zaštitni sloj za kompletan mehanizam autorizacije, za sve zahtjeve poslane sa zaštićenih ruta trebalo bi provesti propisnu validaciju kontrole pristupa.

```
const PrivateRoute = ({ component: Component, rule, ...props
}) => {
  const profile = useSelector((state) => state.profile);
  const { isAuthenticated, role } = profile;

  if (!isAuthenticated) {
    return <Redirect to="/auth/sign-in" />;
  }

  const { permissions } = role;
  const isAuthorized = checkIfUserHavePermission(
    rule,
    permissions
  );

  if (isAuthenticated && !isAuthorized) {
    return <Redirect to="/unauthorized" />;
  }

  return (
```

```
      <Route
        {...props}
        render={(routeProps) => (
          <Component {...routeProps} roleTitle={role.title} />
        )}
      />
    );
};
```

Kôd 19 PrivateRoute komponenta

5.3.2.4 Implementacija komponenti s obzirom na ulogu

Osim zajedničkih komponenti koje su jednake za sve tipove uloga, unutar aplikacije postoji veliki broj komponenti koje pružaju dodatne funkcionalnosti privržene određenom tipu uloge. Primjerice komponenta za prikaz određenog proizvoda, u slučaju prijavljenog vlasnika praonice prikazat će dodatne opcije za uređivanje i brisanje samog proizvoda, dok će u slučaju korisnika, prikazati opciju za dodavanje proizvoda unutar košarice. Ovaj pojam unutar React-a naziva se uvjetno iscrtavanje (engl. *Conditional Rendering*) i dopušta enkapsulaciju ponašanja komponente ovisno o stanju aplikacije.

```
const Container = styled.div`
  display: grid;
  grid-template-columns: 1fr;
  gap: 2rem;
  grid-template-areas: ${(props) => {
    if (props.role === ROLES.ADMIN) {
      return `
      "users"
      "shops"
      "orders"
      `;
    }
    if (props.role === ROLES.SERVICE) {
      return `
        "revenue",
        "shops",
        "orders"
      `;
    }
    return `
```

```
    "shops"
    "orders"
    `;
  }};
`;

const Dashboard = ({ roleTitle }) => (
  <>
    <DashboardHeader />
    <Container role={roleTitle}>
      {roleTitle === ROLES.ADMIN && <DashboardUsers />}
      {roleTitle === ROLES.SERVICE && <DashboardRevenue />}
      <DashboardShops />
      <DashboardOrders />
    </Container>
  </>
);
```

Kôd 20 Dinamičko prikazivanje i stiliziranje komponenti

5.3.2.5 Implementacija HTTP klijenta

Axios je popularan HTTP klijent odgovoran za upućivanje zahtjeva uslugama trećih strana (engl. *third-party services*). S ciljem upravljanja sesijom i slanjem odgovarajućih tokena za provjeru autentičnosti i autorizacije bilo je potrebno implementirati HTTP klijenta. Jedna od njegovih uloga je mogućnost osvježavanja *access tokena* koji se šalje u zaglavlju zahtjeva pomoću *refresh tokena* kojeg smo prethodno definirali na poslužitelju i poslali putem kolačića. U tom naumu, poslužili smo se ugrađenim funkcijama, presretačima (engl. *interceptors*), koji omogućuju izvršavanje koda ili izmjenu zahtjeva i odgovora prije njihovih pokretanja.

```
const instance = axios.create({
  withCredentials: true,
  baseURL: server-url,
});

instance.interceptors.request.use(
  (config) => {
    const jwt = localStorage.getItem(
      "access-token"
    );
```

```
    config.headers.Authorization = `Bearer  ${jwt}`;

    return config;
  },
  (error) => Promise.reject(error)
);

instance.interceptors.response.use(
  (response) => response,
  async (error) => {
    const initialRequest = error.config;

    if (error.response.status === 401 &&
    !initialRequest._retry) {
      initialRequest._retry = true;

      try {
        const url = "/auth/refresh-tokens";
        const { data } = await instance.post(url, {
          token: getCookie("refresh-token"),
        });
        localStorage.setItem(
          "access-token",
          data.accessToken
        );
        return instance(initialRequest);
      } catch (_err) {
        window.location.href = "client-url/auth/sign-in";
      }
    }

    return Promise.reject(error);
  }
);
```

Kôd 21 Implementacija HTPP klijenta

Zaključak

*„**There are only two types of companies**:*

those that have been hacked,

and those that will be.“

Robert Mueller, 2012

Implementacija sigurnosnih rješenja unutar web aplikacija iscrpan je i mukotrpan posao. Implementacija svakog novog „procesa“ unutar aplikacije sa sobom donosi potencijalne prijetnje i probleme. Cijeli niz sigurnosnih principa i rješenja za ublažavanje napada neophodan je za postizanje pozitivnog sigurnosnog položaja web aplikacije. Važno je također napomenuti da sveobuhvatan pristup zahtijeva suradnju među mrežnim, sigurnosnim, operativnim i razvojnim timovima, budući da svaki od njih ima svoju ulogu u zaštiti aplikacija i njihovih kritičnih podataka.

S ciljem izgradnje aplikacije namijenjene široj ciljanoj skupini posebnu pozornost trebalo je pridodati upravo povjerljivosti i integritetu. Uklapanjem autorizacije temeljene na ulogama aplikacija je dobila jednu sasvim novu dimenziju. Određenim ulogama korisnika dopušten je pristup samo onim značajkama koje su im bile potrebne za nesmetan rad s čime se smanjila moguća površina napada i ojačala sigurnost aplikacije. Načelo najmanje privilegije važan je koncept dizajna za poboljšanje zaštite podataka i postizanje bolje stabilnosti i sigurnosti sustava. Dodjelom uloga i dozvola postavili su se temelji skalabilnosti i fleksibilnosti sustava koji služe kao ključna početna točka u daljnjem razvitku projekta.

Literatura

[1] „Web Application Security“. Pristupljeno: ruj. 18, 2022. [Na internetu]. Dostupno na: https://www.synopsys.com/glossary/what-is-web-application-security.html

[2] „Web Application Security Basics 101: where to start“. Pristupljeno: ruj. 18, 2022. [Na internetu]. Dostupno na: https://www.synopsys.com/glossary/what-is-web-application-security.html

[3] „Stride (security)“. Pristupljeno: ruj. 19, 2022. [Na internetu]. Dostupno na: https://en.wikipedia.org/wiki/STRIDE_(security)

[4] „What is the Information Security Triad“. Pristupljeno: ruj. 19, 2022. [Na internetu]. Dostupno na: https://www.fortinet.com/resources/cyberglossary/cia-triad

[5] B. Sullivan i V. Liu, Web Application Security: A beginner's guide. McGraw-Hill Osborne Media, 2011.

[6] „Common Weakness Enumeration“. Pristupljeno: ruj. 19, 2022. [Na internetu]. Dostupno na: https://cwe.mitre.org/

[7] „What is Authentication“. Pristupljeno: ruj. 21, 2022. [Na internetu] Dostupno na: https://auth0.com/intro-to-iam/what-is-authentication

[8] „Authentication Cheat Sheet“. Pristupljeno: ruj. 22, 2022. [Na internetu] Dostupno na: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

[9] „What is JWT?“. Pristupljeno: ruj. 22, 2022. [Na internetu] Dostupno na: https://www.akana.com/blog/what-is-jwt

[10] „What Are Refresh Tokens and How to Use Them Securely?“. Pristupljeno: ruj. 22, 2022. [Na internetu] Dostupno na: https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/

[11] „Understanding Role-Based Access Control (RBAC)?“. Pristupljeno: ruj. 22, 2022. [Na internetu] Dostupno na: https://www.strongdm.com/rbac

[12] D. F. Ferraiolo, D. R. Kuhn, R. Chandramouli, Role-Based Access Control, Second Edition. Artech House, 2007.

[13] „A1:2021 – Broken Access Control“. Pristupljeno: ruj. 23, 2022. [Na internetu] Dostupno na: https://owasp.org/Top10/A01_2021-Broken_Access_Control/

[14] „Session hijacking attack“. Pristupljeno: ruj. 24, 2022. [Na internetu] Dostupno na: https://owasp.org/www-community/attacks/Session_hijacking_attack

[15] „Insecure Direct Object Reference Prevention Cheat Sheet“. Pristupljeno: ruj. 24, 2022. [Na internetu] Dostupno na: https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html

[16] „Cross Site Request Forgery (CSRF)“. Pristupljeno: ruj. 24, 2022. [Na internetu] Dostupno na: https://owasp.org/www-community/attacks/csrf

[17] „Session Management“. Pristupljeno: ruj. 24, 2022. [Na internetu] Dostupno na: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

[18] „Cross-Site Request Forgery Prevention Cheat Sheet“. Pristupljeno: ruj. 24, 2022. [Na internetu] Dostupno na: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

[19] „JavaScript“. Pristupljeno: ruj. 25, 2022. [Na internetu] Dostupno na: https://en.wikipedia.org/wiki/JavaScript

[20] „ECMAScript“. Pristupljeno: ruj. 25, 2022. [Na internetu] Dostupno na: https://en.wikipedia.org/wiki/ECMAScript

[21] „Everything you need to know about Node.js“. Pristupljeno: ruj. 25, 2022. [Na internetu] Dostupno na: https://dev.to/jorge_rockr/everything-you-need-to-know-about-node-js-lnc#nodejsandtheeventloop

[22] „Express.js“. Pristupljeno: ruj. 25, 2022. [Na internetu] Dostupno na: https://en.wikipedia.org/wiki/Express.js

[23] „Introduction to Sequelize ORM for Node.js“. Pristupljeno: ruj. 25, 2022. [Na internetu] Dostupno na: https://www.section.io/engineering-education/introduction-to-sequalize-orm-for-nodejs/

[24] „React (JavaScript library)“. Pristupljeno: ruj. 25, 2022. [Na internetu] Dostupno na: https://en.wikipedia.org/wiki/React_(JavaScript_library)

[25] „ReactJS | Router“. Pristupljeno: ruj. 25, 2022. [Na internetu] Dostupno na: https://www.geeksforgeeks.org/reactjs-router/

[25] „Three Principles“. Pristupljeno: ruj. 25, 2022. [Na internetu] Dostupno na: https://redux.js.org/understanding/thinking-in-redux/three-principles

Popis slika