UNIVERZITET U BANJOJ LUCI
ELEKTROTEHNIČKI FAKULTET

Marija Vasić

SIGURNOST SAVREMENIH INFORMACIONIH SISTEMA NA INTERNETU

diplomski rad

Banja Luka, septembar 2024.

Tema: **SIGURNOST SAVREMENIH INFORMACIONIH SISTEMA NA INTERNETU**

Ključne riječi:
Informacioni sistem
Arhitektura sistema
Sigurnost na Internetu
Identitet
OAuth2
OpenID Connect
SSO
Provajder identiteta
Keycloak

Komisija:
prof. dr Miloš Ljubojević, predsjednik
doc. dr Mihajlo Savić, mentor
Danijela Banjac, ma, član

Kandidat: Marija Vasić

UNIVERZITET U BANJOJ LUCI
ELEKTROTEHNIČKI FAKULTET
KATEDRA ZA RAČUNARSTVO I INFORMATIKU

Predmet: INFORMACIONI SISTEMI

Tema: SIGURNOST SAVREMENIH INFORMACIONIH SISTEMA NA INTERNETU

Zadatak: Opisati arhitekture savremenih informacionih sistema u Internet okruženju. Dati pregled oblasti sigurnosti na Internetu uz poseban osvrt na uticaj na informacione sisteme u Internet okruženju i specifičnosti u zavisnosti od domena primjene. Opisati i uporediti relevantne savremene standarde i protokole iz oblasti sigurnosti informacionih sistema u Internet okruženju. Odabrati i detaljno opisati jedno moguće rješenje, te realizovati sistem koji ilustruje upotrebu prethodno opisanih tehnologija.

Mentor: doc. dr Mihajlo Savić

Kandidat: Marija Vasić (1161/16)

Banja Luka, septembar 2024.

LISTA SKRAĆENICA I OZNAKA

(D)DoS	(Distributed) Denial of Service
ABAC	Attribute-Based Access Control
CIA	Confidentiality, Integrity, Availability
CSRF	Cross-Site Request Forgery
DAC	Discretionary Access Control
GDPR	General Data Privacy Regulation
HMAC	Hash-Based Message Authentication Code
HTML	HyperText Markup Language
HTTP(S)	Hypertext Transfer Protocol (Secure)
IP	Internet Protocol
JSON	JavaScript Object Notation
JWT	JSON Web Token
LAN	Local Area Network
MAC	Mandatory Access Control
MAN	Metropolitan Area Network
MD5	Message Digest algorithm 5
MFA	Multi-Factor Authentication
MVC	Model-View-Controller
PCI	Payment Card Industry
PII	Personal Identifiable Information
PIN	Personal Identification Number
PKCE	Proof of Key Code Exchange
RBAC	Role-Based Access Control
RuBAC	Rule-Based Access Control
SECaaS	Security as a Service
SOA	Service-Oriented Architecture
SQL	Structured Query Language
SSL	Secure Sockets Layer
SSO	Single-Sign On

SSOAC	Single-Sign On
SWT	Simple Web Token
TCP/IP	Transmission Control Procotol/Internet Protocol
TLS	Transport Layer Security
WAN	Wide Area Network
XML	Extensible Markup Language
XSS	Cross Site Scripting

Sadržaj

1 UVOD

Danas je teško zamisliti život bez tehnologije. Razvijaju se noviji i moderniji sistemi, a prilagođenje postojećih novim zahtjevima je moguće samo ako je razvojni tim od samog početka uzeo u obzir da se mijenjaju i ljudi i procesi koje obavljaju i za koje su ti sistemi osmišljeni. No, na propadanje sistema ne utiče samo nemogućnost praćenja potreba tržišta već i ne razmišljanje o sigurnosti istog. Ovaj rad nudi pregled i opis informacionih sistema, pouzdanih arhitektura i sigurnosti sve zastupljenijih Internet aplikacija. Organizovan je u šest glava.

Prva glava predstavlja kratak uvod u tematiku kojom se rad bavi i opis ostalih glava.

Druga glava se bavi informacionim sistemima, ulogom u životu kako pojedinca tako i kompanije. Sa razvojem računara, a posebno baza podataka, je došlo do evolucije načina razmišljanja i organizacije poslovanja. Informacioni sistemi olakšavaju kompanijama da upravljaju podacima, korisnicima i procesima. Kako bi se poboljšalo održavanje i spriječilo pomenuto propadanje sistema, osmišljene su dobre prakse i arhitekture. Objašnjene su najpoznatije. Poželjno je da se arhitektura formira nakon prikupljanja zahtjeva, kako bi se identifikovalo koja najviše odgovara ispunjenju istih i budžetu. Druga glava ujedno skreće pažnju na sigurnosne kontrole i izazove kada je riječ o informacionim sistemima. Iako je upotreba tehnologije skoro nezaobilazan dio modernog poslovanja, sigurnost je često zanemaren aspekt. Koliko god baza podataka bila napredna i način rada ubrzan, ukoliko sigurnost nije implementirana na adekvatnom nivou, informacioni sistem se ne može nazvati ni dobrim ni pouzdanim. Ne samo da su na taj način ugroženi rad kompanije i priliv sredstava, već je vrlo često ugrožena tuđa privatnost.

Treća glava se bavi ključnim problemima sigurnosti i odbranama. Internet je otvorio nova vrata svijeta informacionim sistemima, a samim tim i napadačima. Napadi preko mreže su češći nego drugi tipovi napada, a stručnjaci iz oblasti sigurnosti su identifikovali najčešće napade i metode odbrane. Jedna od tih metoda odbrane jeste kontrola pristupa. Kada se govori o korisnicima aplikacije, u oblasti sigurnosti se govori i o identitetu, bilo osobe, bilo drugog sistema. Identitet, kao i softver, ima svoj životni ciklus koji nam pomaže da razumijemo kako zaštititi podatke koje sadrži i resurse kojim pristupa.

Četvrta glava se bavi različitim načinima na koje se mogu implementirati autentifikacija i autorizacija. Postoje brojni protokoli, od kojih su najpoznatiji OpenID, OAuth 2.0, SAML i OpenID Connect. Danas je najviše u upotrebi OIDC protokol, jer nudi naprednu specifikaciju i autentifikacije i autorizacije. Većina provajdera identiteta nudi podršku implementiranju ovog protokola. Provajderi identiteta nude funkcionalnosti vezane za kreiranje identiteta, održavanje i uništavanje sesije, implementacije poznatih protokola i mogućnost SSO pristupa. Besplatan provajder identiteta sa javnom dostupnim izvornim kodom jeste Keycloak.

U petoj glavi je opisan praktični dio, koji se bavi implementacijom višeslojne i mikroservisne arhitekture korištenjem .NET i Angular tehnologija, podešavanju i upotrebi Keycloak servera u svrhu implementacije OpenID Connect protokola, SSO autentifikacije i kontrole pristupa bazirane na ulogama. Odvojen je u dvije cijeline: korisnička aplikacija koja nudi upravljanje sadržajem i administratorska aplikacija koja predstavlja prilagođenje administratorskog panela samog Keycloak servera.

Šesta glava opisuje značaj informacionih sistema i upotrebe sigurnosti, autentifikacije i autorizacija od samog početka životnog ciklusa softvera.

Dodatno je navedena literatura korištena prilikom izrade diplomskog rada.

Uz rad je priložen CD.

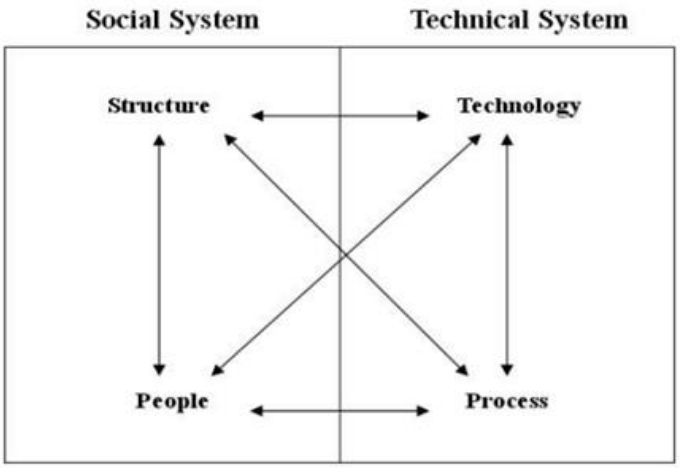
2 INFORMACIONI SISTEMI

2.1 Definicija informacionog sistema

Kada se govori o informacionim sistemima, važno je naglasiti razliku između informacionog sistema i informacione tehnologije. Informaciona tehnologija prenosi, obrađuje ili skladišti informacije, ne nužno u digitalnom obliku. Informacioni sistem je integrisani i usklađeni set informacionih tehnologija koji služe postizanju individualnih, grupnih, organizacionih ili društvenih ciljeva. [1]

2.1.1 Uloga informacionih sistema

Informacioni sistem se posmatra kao socio-tehnološki sistem, sa društvenom i tehnološkom komponentom, kao što je prikazano na slici 2.1 .



Slika 2.1 - Šema socio-tehnološke organizacije informacionog sistema [1]

U modernom vremenu, tehnološku komponentu informacionog sistema čine hardver, softver, baze podataka i telekomunikacioni sistemi ili veze. Softver predstavlja set instrukcija koji upravljaju hardverom u cilju izvršavanja nekog zadatka. Telekomunikacioni sistemi su mrežni elementi koji omogućavaju ljudima i uređajima da komuniciraju, prevazilazeći udaljenost, pri čemu se dijele na LAN, MAN i WAN mreže.
Proces predstavlja set akcija koje individua, grupa ili organizacija mora da obavi da bi se završila neka aktivnost. U slučaju promjene neke od komponenti informacionog sistema, i druge komponente treba da se prilagode. Promjena ili unapređenje tehnologija često zahtijeva obuku ljudi, kao i prilagođenje samog poslovnog procesa.
Ljudi koji čine informacioni sistem su uglavnom menadžeri koji definišu ciljeve sistema, korisnici, kao i ljudi koji održavaju pomenuti sistem i obučavaju korisnike. Kada se dizajnira informacioni sistem, potrebno je uzeti u obzir znanje, vještine, sklonosti, mogućnosti i ostale stavove ljudi koji će biti dio tog sistema. Organizacija ili struktura informacionog sistema označava veze između ljudi u ljudskoj komponenti tog sistema. [1]

2.1.2 Podjela informacionih sistema

Informacioni sistemi se mogu podijeliti na:

- Sistemi kao podrška odlučivanju (DSS)
- Sistemi kao podrška grupnom odlučivanju (GDSS)
- Strateški informacioni sistemi (EIS)
- Sistemi za upravljanje znanjem (KMS)
- Sistemi za operativnu podršku (MSS)
- Sistemi za poslovnu obaviještenost (BI)

Iako su jedni namijenjeni da pomognu pojedincu, drugi grupi, treći menadžmentu i slično, svi su dizajnirani da pomognu pri donošenju odluka u sklopu neke organizacije. Kada neki matematički sistem odgovara realnosti, moguće je konstruisati informacioni sistem koji će automatizovati donošenje odluke (DSS sistemi). Velike kompanije zapošljavaju veliki broj ljudi koji mogu dijeliti znanje i iskustvo, ali lična komunikacija između zaposlenih nije uvijek praktična, pa se desi da kada iskusni ljudi napuste kompaniju, njihovo znanje ode sa njima. Informacioni sistemi koji pomažu u rješavanju problema komunikacije i dijeljenja znanja i iskustva među ljudima su KMS sistemi. Poslovna obaviještenost opisuje informacioni sistem koji pomaže donosiocima odluka da uoče trendove i veze unutar velike količine podataka. Ipak, većina ih se danas koristi za generisanje specijalizovanih izvještaja, ad-hoc pretraživanje i analizu trendova. [1]

2.2 Sigurnost informacionih sistema

Sigurnost informacionih sistema je uvijek bila važan dio svake organizacije, ali je sa ubrzanim rastom tehnologije i poznatim propustima banaka, kompanija i Internet stranica[1] [2], još više dobila na značaju. Dakle, na razvoj sigurnosti informacionih sistema uticali su, i utiču i danas, brojni društveni faktori i razvoj tehnologije. Kao pojedinci, ljudi se oslanjaju na tehnologiju pri izvršavanju svakodnevnih zadataka kao što je zakazivanje leta. Danas se teško može zamisliti organizacija koja se ne oslanja na tehnologiju za komunikaciju, marketing, upravljanje procesima, unapređenje produktivnosti, pa samim tim i korištenje tehnologije za sticanje prednosti u odnosu na konkurenciju. Zahvaljujući tehnologiji, sve više poslova se obavlja sa udaljenih lokacija. Globalna povezanost, Internet i elektronsko poslovanje su drastično promijenili poslovanje. Ukoliko su vezane za određeno mjesto, kompanijama opada strateška prednost.

Održavanje povjerljivosti, integriteta i dostupnosti su bila tri glavna cilja upravljanja sigurnošću informacionih sistema. Pored tog, sigurnost treba da napreduje ka uspostavljanju osjećaja odgovornosti, integriteta ljudi i njihovoj pouzdanosti, kao i moralu. Zbog toga, prilikom dizajna informacionog sistema, organizacija treba da minimizuje efekte oslanjanja na nepovjerljive ili nesigurne sisteme i entitete i da poveća otpornost od tehnoloških kvarova.

Naravno, ostati zatvoren za napade, prevare i druge sigurnosne propuste i istovremeno ostati otvoren za podjelu informacija sa partnerima, nije jednostavan posao. Tom ne pomaže sve veće napredovanje tehnologije i uporedo potreba za organizacijskom fleksibilnošću. [1]

2.2.1 Definicija sigurnosti informacionih sistema

Prilikom dizajna i upravljanja sigurnošću informacionih sistema, organizacije treba da obrade brojne faktore, od tehnoloških do onih koji se tiču poslovnog okruženja, kulture organizacije, odgovornosti i poslova različitih uloga zaposlenih. Zato se sigurnost informacionih sistema

[1] Neki od najpoznatijih napada na Internet sigurnost su sigurno napad na elektronski sistem države Estonije, kao i napad na Yahoo 2014. godine. [2]

može definisati kao "smanjivanje rizika koji nastaje usljed nekoherentnog i nekonzistentnog ponašanja prilikom rukovanja informacijama organizacije". Pored prirodnih katastrofa, neželjena dejstva na podatke mogu biti uzrokovana namjernim ili slučajnim postupcima pojedinca bilo u obliku ljudske greške, kršenja sigurnosnih mjera ili napada na sistem i podatke. [1]

2.2.2 Sigurnosne kontrole

Računarska sigurnost treba da ispuni 5 funkcija: identifikacija, zaštita, detekcija, odgovor i oporavak. Potrebno je realizovati kontrolu pristupa, tako da se korisniku daju najmanja moguća prava da bi izvršio određeni zadatak. To može da se odnosi i na trajanje sesije korisnika u sistemu, pravo upisa samo za određene uloge korisnika i slično. Pored principa najmanjeg prava pristupa, važno je ispuniti princip zaštite sistema na više nivoa, jer napadač može da probije jedan sloj zaštite i ugrozi sistem. Kada govorimo o sigurnosnim kontrolama, moguće je da sistem spriječi napad prije nego što se desi, da detektuje napad i obavijesti administratora sistema da se napad dešava ili da omogući odgovor za vrijeme napada u vidu ponovnog podešavanja sistema, rezervnih kopija podataka i sličnog. [3]
Dakle, da bi spriječile, detektovale ili reagovale na date događaje, organizacije primjenjuju set mjera poznatih kao sigurnosne kontrole. S obzirom da upravljanje informacijama u sklopu organizacije postoji na tehničkom, formalnom i neformalnom nivou, sigurnost informacionih sistema može da se postigne samo koordinacijom integriteta i sigurnosti na sva tri nivoa.
Na tehničkom nivou, organizacija može da usvoji niz sigurnosnih kontrola, kao što su antivirusni softver, zaštitni zidovi (*engl. firewalls*), sistemi za otkrivanje upada, uređaji za kontrolu pristupa, kriptografski uređaji, ali i jednostavan uređaj kao što je sjekač papira.
Na formalnom nivou, sigurnosne kontrole bi bile sigurnosne polise, šema strukture odgovornosti i šema nepredviđenih slučajeva.
Na neformalnom nivou, sigurnosne kontrole koje mogu da se primjene su programi svijesti, prihvatanje dobrih menadžmentskih praksi i razvoj kulture koja njeguje zaštitu podataka. [4]

2.2.3 Sigurnosni izazovi

Tradicionalne organizacije su kreirale sisteme koji su smatrani sistemi sa svrhom, pri čemu sigurnosni sistem tog sistema nije smatran njegovim svrsishodnim dijelom. Organizacije ne polažu dovoljno pažnje na sigurnost, kao ni na oporavku u slučaju napada. Kada se govori o sigurnosti informacionih sistema, govori se i o izazovima koje treba da riješi uspostavljanje:

- dobrih praksi menadžmenta u geografski raštrkanom okruženju, i mogućnosti kontrole organizacionih operacija [2],
- sigurnosnih procedura koje odražavaju kontekst organizacije i poslove kojima se bavi,
- stroge i korektne strukture odgovornosti, zavisno od strukture same organizacije i aktivnosti obrade informacija na različitim nivoima[3],
- planova oporavka u slučaju sigurnosne katastrofe.

Prema istraživanju IBM-a iz 1996. godine, 293 od 300 kompanija je pretrpjelo sigurnosne incidente u prethodnoj godini. Gubitak je procijenjen na 500.000 čovjek-sati. Skoro 25% kompanija je čuvalo više od 60% svojih podataka na računarima, a više od 76% nisu pravili rezerve podataka. I nakon brojnih terorističkih napada, ponavljajućih DDOS napada, čak četvrtina britanskih kompanija nije čuvala rezerve podataka, a zabrinjavajući broj nije imao plan oporavka od napada, prema istraživanju iz 2006. godine.

[2] Primjer lošeg rješavanja je skandal Barings banke.
[3] Primjer lošeg rješavanja je skandal Daiwa banke.

Sigurnost informacionih sistema zavisi od ljudi koji čine taj sistem. Ipak su ljudi ti koji dizajniraju, implementiraju i izvršavaju mjere sigurnosti. Ljudi su ti koji pristupaju, koriste, upravljaju i održavaju informacione resurse organizacije. Istraživanje Jozefa Džeta na temu neetičkog korištenja računara pokazuje važnost stvaranja kulture povjerenja i odgovornosti. Princip je sljedeći:
"Trening, buđenje svijesti i edukacija su važni, ali nedovoljni za upravljanje sigurnošću informacionog sistema. Fokus treba da bude na stvaranju kulture sigurnosti."
Neki autori tvrde da su integritet, povjerljivost i dostupnost podataka previše striktni da bi se implementirali u modernom informacionom sistemu, i da su odgovornost, integritet zaposlenih, povjerenje i moral vrijednosti koje treba da se promovišu da bi se uspostavio siguran informacioni sistem. Jedan od principa koji se mogu pratiti za uspostavljanje sigurnosti na formalnom nivou jeste:
"Važno je uspostaviti granicu između pisanih i nepisanih pravila ili normi. Sama pravila nemaju mnogo smisla bez konteksta u kome se ta pravila primjenjuju."
Kako je svaka organizacija drugačija, sama forma sigurnosnih polisa treba da bude zavisna od situacije, a ne generalizovana. [1]

2.3 Softverska arhitektura sistema

Nakon opsežnog opisa informacionih sistema i sigurnosti koja se tiče istih, bitno je opisati moguće arhitekture softverske komponente sistema.

2.3.1 Definicija arhitekture sistema

Teško je definisati arhitekturu sistema, ali je nesumnjiva uloga arhitekture na cjelokupni sistem, održavanje, testiranje pa i implementaciju tehničkih sigurnosnih kontrola. Neka od tumačenja su:

- Arhitektura softvera se može posmatrati kao metafora za razvoj softvera. [5]
- Arhitektura sistema se može posmatrati kao subjektivna stvar, dijeljeno razumijevanje dizajna sistema od softverskih profesionalaca koji na tom sistemu rade. Dijeljeno razumijevanje je u formi glavnih komponenti sistema i načina na koji one sarađuju. [6]
- Arhitektura sistema je most između apstraktnih poslovnih ciljeva i finalnog implementiranog sistema. Softverska arhitektura je skup struktura, kao što su softverski elementi[4], i veza između njih i njihovih atributa, potrebnih za razumijevanje sistema. Arhitektura je takođe apstrakcija softverskih elemenata i njihovih veza, jer su pojedinosti implementacije izostavljeni. Svaki sistem, bez obzira od čega se sastoji, ima arhitekturu. Razlikuju se arhitektura sistema i poslovna arhitektura, pri čemu obje utiču na arhitekturu softvera, koja je u modernom vremenu podsegment obje. [7]

Martin Fowler vidi informacione sisteme kao sisteme sa perzistentnim podacima, i naziva ih poslovnim aplikacijama [5]. Iako ljudi poslovne aplikacije vide kao ogromne sisteme, to nije nužno slučaj. Izbor odgovarajuće arhitekture podrazumijeva poznavanje domena problema sistema. [6] Softverski obrazac predstavlja potencijalno rješenje ili početnu tačku ponavljajućeg problema. Najbitnije karakteristike su ime, alijas, namjena i dijagram klasa. [6] [8][6]. Neizostavan dio arhitekture softvera su strukture softverskog sistema. Neke od njih su sljedeće strukture:

[4] Autor termin softverska komponenta koristi samo u kontekstu dinamičkih softverskih elemenata. Sofverski element može biti i set linija koda za sortiranje niza, ali se ne smatra arhitekturnom strukturom.
[5] Termini poslovna aplikacija i informacioni sistem mogu da se koriste naizmjenično. [6]
[6] U radu će biti pomenuti i obrasci koji se ne smatraju strogo rečeno arhitekturom sistema, ali omogućavaju bolje razumijevanje dizajna sistema i sigurnosnih aspekata.

- Dekompozicija: jedinice su moduli koji su povezani relacijom "je-podmodul-od", prikazujući na taj način kako su moduli dekomponovani na manje cjeline koje se mogu lako shvatiti, često nazivane segmenti ili podsistemi. Dekompozicija određuje u velikom stepenu mogućnost modifikacije sistema, pri čemu su cjeline podložne promjeni lokalizovane, enkapsulaciju i skrivanje informacija.
- Koristi: jedinice su takođe moduli ili klase, povezani relacijom "koristi". Korisno je pri nadogradnji sistema.
- Sloj: jedinice su moduli koji predstavljaju apstrakciju virtuelne mašine čije su funkcionalnosti izložene preko interfejsa. Jedan sloj može da koristi drugi sloj koji je ispod, ali ne iznad njega, što omogućava laku promjenu sloja iznad i ponovnu upotrebu sloja ispod. Omogućava inkrementalni razvoj i prenosivost.
- Klasa: jedinice su klase, koje su povezane relacijom nasljeđivanja ili kompozicije. Koristi se u objektno-orjentisanom pristupu. Omogućava arhitekti da razmišlja o elementima kao klasama koje mogu nadograditi, bilo ponašanjem bilo podacima, klasu koja ima slično ponašanje ili mogućnosti. Korisno je za unapređenje nadogradnje i modifikacije sistema.
- Model podataka: opisuje strukturu statičkih informacija u vidu entiteta i njihovih veza. Na primjer, u bankovnom sistemu, entiteti su uglavnom *Klijent, Nalog, Banka*. *Nalog* može imati više atributa, a *Klijent* može da ima više naloga u više različitih banaka. Prilično utiče na mogućnost modifikacije sistema i na performanse.
- Servis: omogućuje da se dizajnira sistem koji se sastoji od komponenti koje se razvijaju i mijenjaju nezavisno jedna od druge. Servisi obično komuniciraju sa drugim servisima. Utiče na interoperabilnost i mogućnost modifikacije sistema.
- Struktura konkurentnosti: ako posmatramo ovu strukturu kao komponenta-konektor strukturu, komponente su organizovane u logičke niti ili procese, a konektor je način komunikacije između njih. Logička nit jeste niz izračunavanja koji se može rezervisati na fizičkoj niti. Poželjno je koristiti da bi se upravljalo zahtjevima za paralelnim izvršavanjem.

Arhitektura sistema omogućava lakšu komunikaciju sa klijentima, jer je dovoljno apstraktna, a takođe omogućava lakši rad unutar organizacije i tima, jer se upotrebljavaju jasno definisani arhitekturni termini i sa dobrom arhitekturom se sam posao može lakše podijeliti na više osoba. [7]

2.3.2 Tipovi arhitekture sistema i zahtjevi po pitanju sigurnosti

2.3.2.1 Monolitna arhitektura

U monolitnoj arhitekturi, sistem je kreiran kao jedna velika cjelina usko povezanih, slabo definisanih komponenti ili interfejsa. Komponente nisu izolovane, teško se mogu nezavisno razvijati, ili dijeliti na različite uređaje. Manji tim se može odlučiti za ovu arhitekturu, što često rezultuje u sistemima koji se teško održavaju. Sistem sa monolitnom arhitekturom se pokreće kao jedan proces i takav sistem je teško podijeliti ili unaprijediti da se pokreće kao više procesa na više računara ukoliko se takva potreba javi. S druge strane, komunikacija između komponenti monolitne aplikacije je vrlo brza i jeftina. Takvu aplikaciju je lako instalirati na klijentskoj mašini. [9] Monolitna aplikacija vremenom raste, teško je održavati, kvalitet koda često opada, a samim tim i sigurnost postaje narušena. Površina podložna napadu, odnosno

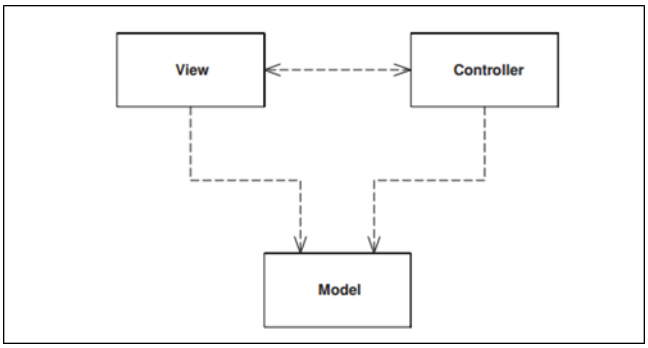
suma tačaka ranjivosti, kod monolitne aplikacije je zbog njenih karakteristika prilično velika, kada se uporedi sa izolovanim mikroservisima. [10]

2.3.2.2 Model-Pogled-Kontroler arhitektura

Korisnički interfejs je najpodložniji promjenama kod interaktivne aplikacije. Zato je važno korisnički interfejs odvojiti od ostatka aplikacije. Ujedno, korisnički interfejs treba da se dinamički mijenja da bi zadovoljio različite zahtjeve korisnika i prikazao različite oblike podataka. MVC je arhitekturni stil koji razdvaja funkcionalnosti aplikacije na tri vrste komponenti:

- Model - sadrži podatke aplikacije, kao i poslovnu logiku i ne zna za postojanje kontrolera.
- Pogled - prikazuje i stilizuje podatke.
- Kontroler- posreduje između modela i pogleda, korisničke akcije transformiše u akcije nad jednim ili nad drugim, i upravlja notifikacijama o promjeni stanja tako da se pogled mijenja sa promjenama modela. [6]

Slika 2.2 ilustrativno prikazuje veze između komponenti.



Slika 2.2 - MVC dijagram [6]

Komponente nisu usko povezane, pa se mogu paralelno razvijati i lakše mijenjati i nadograđivati. Razdvajanje modela od pogleda je najvažnija karakteristika MVC arhitekture. Pored tog što omogućuje odvojen razvoj, odvojenu modifikaciju, omogućava nezavisno testiranje modela i poslovne logike od pogleda. Često postoji više prezentacija istog modela, i kada se promijeni model na jednom pogledu, potrebno je da se ažurira na drugom, pa se često koristi obrazac izdavač-pretplatnik (*engl. publisher-subscriber pattern*). Potrebno je obratiti posebnu pažnju na čišćenje i validaciju korisničkog unosa, kako bi se osiguralo pravilno korištenje aplikacije i spriječili napadi poput ubacivanja SQL koda (*engl. SQL injection*). [7]

2.3.2.3 Klijent-server arhitektura

Klijent-server arhitektura se odlikuje sa separacijom logike između klijentske i serverske komponente. Klijent je taj koji zahtijeva neki servis od serverske komponente. Server je komponenta koja pruža usluge klijentu, pazeći na broj klijenata koji se mogu istovremeno konektovati, kao i na performanse. Klijent i server su dakle povezani vezom "koristi usluge". Server može biti klijent u nekoj drugoj vezi sa nekim drugim serverom, ukoliko se u toj vezi ponaša kao klijent i koristi usluge serverske aplikacije. Klijent treba da zna koji servis da pozove, tako da je potrebno omogućiti identifikaciju servisa servera. Kako je serverska aplikacija odvojena od klijenta, lako je dodati novog klijenta, a broj klijenta teoretski može biti neograničen. Svjetska mreža (*engl. World Wide Web*) je najpoznatiji sistem koji koristi klijent-

server arhitekturu, pri čemu su Internet pretraživači klijenti, a serveri sa uskladištenim Internet stranicama i sadržajem su serveri, a jedan od korištenih protokola je HTTP. Ova arhitektura omogućava višestruku iskoristivost serverske komponente, i razdvaja klijentsku i serversku aplikaciju, pri čemu klijentska aplikacija može biti ili korisnički interfejs ili neki drugi servis. Nedostaci se ogledaju u performansama, jer server može postati usko grlo aplikacije, jedina tačka pada i može biti jako kompleksan. [7] Vrlo je važno osigurati sigurnu komunikaciju između ove dvije komponente, pa se preporučuje korištenje enkriptovanih protokola, kao što su SSL/TLS.

2.3.2.4 Slojevita arhitektura

Slojevita arhitektura rješava problem nezavisnog razvoja određenih komponenti na vrlo elegantan način upotrebom dva ili više slojeva, ujedno podržavajući modifikovanje, prenosivost i ponovnu upotrebu. Kako bi se postiglo sve navedeno, svaki sloj predstavlj a grupu kohezivnih modula, a veze između slojeva moraju biti jednosmjerne. Funkcionalnosti sloja mogu da se koriste preko javnog interfejsa koji sloj pruža. Sloj smije da koristi funkcionalnosti samo sloja neposredno ispod ili još jednog sloja niže (što je poznato kao premoštavanje slojeva). Iako slojevi nude veliku fleksibilnost, ukoliko nisu dobro implementirani, mogu samo da zakomplikuju situaciju i smetaju programeru. Često sloj iznad poziva funkciju sloja ispod, koja poziva drugu funkciju sloja ispod, delegirajući poziv do odgovarajuće implementacije, čime se smanjuje brzina same aplikacije. [7]
Slojevita arhitektura ili raslojavanje je jedna od najkorištenijih arhitekturnih stilova, bilo da je riječ o TCP/IP mrežama, instrukcijama procesora ili Internet aplikacijama. Sloj omogućava klijentu da ga posmatra kao cjelinu, bez nepotrebnog ulaženja u detalje drugih nižih slojeva, sloj se lako može zamijeniti drugim slojem, veze između slojeva su na minimalnom nivou, niži sloj se može koristiti od strane nebrojeno mnogo viših slojeva. Takva fleksibilnost ima svoju cijenu. Slojevi često imaju redundantne podatke i narušavaju performanse aplikacije. Iako slojeva može biti nebrojeno mnogo, Martin Fowler ističe troslojnu arhitekturu kao bazu za svaku ostalu slojevitu arhitekturu:

- Sloj prezentacije prikazuje informacije, prihvata korisničke akcije koje obrađuje i delegira domenskom sloju, bilo da je riječ o izvršavanju skripte ili prikazu Internet stranice. Korisnik može da bude čovjek ili mašina. Sloj prezentacije u ovom slučaju predstavlja interfejs koji pruža servise sistema klijentima.
- Domenski sloj predstavlja poslovnu logiku, što uključuje obradu podataka, validaciju i delegiranje poziva sloju podataka. Ovaj sloj je adekvatno mjesto za implementaciju sigurnosnih mehanizama i validaciju podataka.
- Sloj podataka rješava komunikaciju sa bazama podataka, sistemima sa porukama, transakcionim menadžerima i sličnim eksternim sistemima koji rješavaju neke zadatke u ime aplikacije.

Slojevi se mogu, a ne moraju, izvršavati na različitim mašinama. Uz brojne prednosti takve arhitekture, potrebno je dodatno obratiti pažnju na sigurnu komunikaciju između slojeva i adekvatno implementirane sigurnosne mehanizme da bi se slojevi zaštitili od neautorizovanog pristupa. Postavlja se pitanje na kom sloju čuvati informacije o sesiji korisnika. [6]
Ukoliko se primjenjuje slojevita arhitektura, a slojevi se međusobno isprepliću, ne poštujući pravila sloja, povećava se kompleksnost sistema i ugrožava sigurnost, naročito ako sloj prezentacije direktno komunicira sa slojem podataka. Bitno je isto obratit i pažnju na elemente koji presjecaju slojeve: instrumentalizacija, autentifikacija, autorizacija, obrada izuzetaka, komunikacija, keširanje i logovanje. Autentifikacija, autorizacija i logovanje su ključne

komponente sigurnosne arhitekture, i trebaju da se dizajniraju u arhitekturi tako da postanu dio menadžmenta procjene rizika. Sloj prezentacije postaje sloj preko kog i individua i napadač pristupaju aplikaciji, i oba tipa korisnika treba da se uzmu u obzir prilikom dizajna aplikacije, jer ako komponente nisu dobro povezane, napadač lako može da dođe do sloja podataka preko prezentacionog sloja. Ukoliko se autentifikacija i autorizacija uključe u sloj prezentacije, aplikacija je ranjivija, jer logika autentifikacije može biti potpuno izložena napadaču. Keširanje podataka se često koristi za postizanje veće brzine dobijanja odgovora, ali treba pažljivo i sa sigurnošću u vidu odlučiti koje podatke keširati, a koje ne. Korisnički PIN ne treba da se kešira, kao ni brojevi kreditnih kartica. Komunikacija između slojeva može da bude tačka za DoS napad, ukoliko se dozvoljava ponavljanje zahtjeva. Takođe, prikaz poruka treba da sadrži što manje sadržaja, kako napadač ne bi saznao informacije potrebne za provođenje napada. Obavezno je validirati unos korisnika. Sve ovo treba da se uzme u obzir prilikom odabira arhitekture i prilikom dizajniranja sistema. [11]

2.3.2.5 Servisno-orjentisana arhitektura

Servisno-orjentisana arhitektura predstavlja formu tehnološke arhitekture koja je dizajnirana tako da podrži koncept slabo vezanih servisa kako bi se poboljšala interoperabilnost između sistema i višestruka iskoristivost servisa.
Servisi postoje kao fizički odvojeni programi, pri čemu svaki servis ima svoj funkcionalni kontekst i određen set funkcionalnosti i mogućnosti zavisno od tog konteksta, koji se mogu koristiti preko objavljenog servisnog ugovora. Servisi se lako mogu ponovo iskoristiti, autonomni su, bez stanja, lako se mogu otkriti i mogu se kombinovati zarad kompleksnije funkcionalnosti. Lako se decentralizuju i skaliraju. Iako su autonomni, mogu da dijele resurse, pa pri dizajnu treba uzeti u obzir sigurnost kod pristupanja. [12] Zbog decentralizovane prirode, podaci konstantno putuju u svim smjerovima, trebaju biti zaštićeni konstantno. Da bi se implementirala kontrola pristupa, potrebno je da se definiše negdje, i ostatak sistema treba da bude svjestan pravila i da ih poštuje, a kako postoji mnogo resursa u sistemu, previše je naporno da se korisnik autentifikuje svaki put kada pokuša da pristupi novom resursu, pa se pribjegava SSO pristupu. U SOA su ovi sigurnosni mehanizmi implementirani u XML, što je samo po sebi sigurnosno problematično. Potrebno je koristiti enkripciju. [13]
SOA se koristi u mnogim cloud infrastrukturama. U momentu migracije standardne infrastrukture na cloud-u, običan zaštitni zid više nije dovoljna zaštita. Protok informacija treba da se odvija preko enkriptovanih protokola, ali i digitalni potpisi ključeva za enkripciju se trebaju provjeriti, kako bi se garantovala sigurnost i da bi se spriječili napadi. Idealno, obe strane enkriptovanog tunela su potpisane. SECaaS često nisu primjenjivi u drugim cloud okruženjima, što primorava inženjere da koriste sigurnosne servise koji nisu toliko razvijeni. W. Williams tvrdi da izbor arhitekture softvera uključuje pravljenje kompromisa između funkcionalnosti, performansi i sigurnosti. Funkcionalnosti se uvijek daje prednost, jer niko ne mari koliko je brz ili siguran sistem koji ne ispunjava svoju ulogu.
Dizajnom sistema koji ima nisku povezanost između front-end interfejsa, back-end interfejsa i čuvanja podataka, unapređuje se lakoća održavanja sistema i interoperabilnost sa drugim sistemima. SOA zadovoljava ovaj princip arhitekture (jedan od mnogih). SOA takođe omogućava iterativan razvoj komponenti i mogućnost pojedinačnog dizajna, implementiranja, testiranja i validacije komponenti. Drugi princip dizajna sistema je da svaka komponenta treba da ima jedinstvenu ulogu. To se prenosi i na sigurnost sistema, pri čemu umjesto uloge, govorimo o jedinstvenosti dužnosti. Još jedan princip dizajna arhitekture sistema koji se često krši jeste dodavanja funkcionalnosti koje su suvišne u sistemu. Koliko god bilo interesantno proširivati sistem, to povećava sigurnosne propuste, a sistem postaje nepotrebno veći i teži za održavanje. Takođe, vrlo je bitno sigurnosni aspekt aplikacije analizirati i primjenjivati u

svakom koraku, a ne dodavati na kraju implementacije. Sam dizajn arhitekture sistema treba da uključi i sigurnosne zahtjeve i ranjivosti. [11]

2.3.2.6 Mikroservisna arhitektura

Mikroservisna arhitektura je arhitektura bazirana na mikroservisima. Mikroservisi se mogu definisati kao nezavisni servisi građeni oko poslovne logike, koji enkapsuliraju neku funkcionalnost i čine je dostupnom preko mreže. Iako su tip SOA arhitekture, granice servisa su drugačije nego u SOA arhitekturi i mogu nezavisno da se raspoređuju na mašine. Nezavisni su od tehnologije. Izvana je mikroservis crna kutija, što znači da je korisnicima dostupan preko interfejsa, dok su ostale informacije, kao detalji implementacije sakriveni. Svaki mikroservis ima svoje stanje, nezavisno od stanja drugih mikroservisa. Jedna od prednosti mikroservisa u odnosu na troslojnu arhitekturu se ogleda u raspoređivanju poslovne logike. Za razliku od slojevite arhitekture gdje imamo horizontalnu podjelu poslovne logike, tako da promjena u poslovnoj logici zahtijeva promjenu na sva tri sloja, kod mikroservisne arhitekture svaki mikroservis predstavlja funkcionalnost, poslovna logika može vertikalno da se podijeli i znatno je lakše održavati promjene poslovne logike. Lako se skaliraju. Kao i kod SOA arhitekture, i kod mikroservisa je važno osigurati sigurnu komunikaciju između korisnika servisa i servisa i samih servisa. Mehanizmi zaštite su najčešće korištenje sigurnih protokola, kao što je uzajamni TLS (*engl. mutual TLS*), enkripcije i dekripcije na različitim nivoima, implementiranje autentifikacije i autorizacije za među komunikaciju servisa, kao i korištenje kontejnera za izolaciju servisa. Iako mikroservisi povećavaju površinu napada[7], oni omogućuju razvojnom timu da zaštite servis na više nivoa.

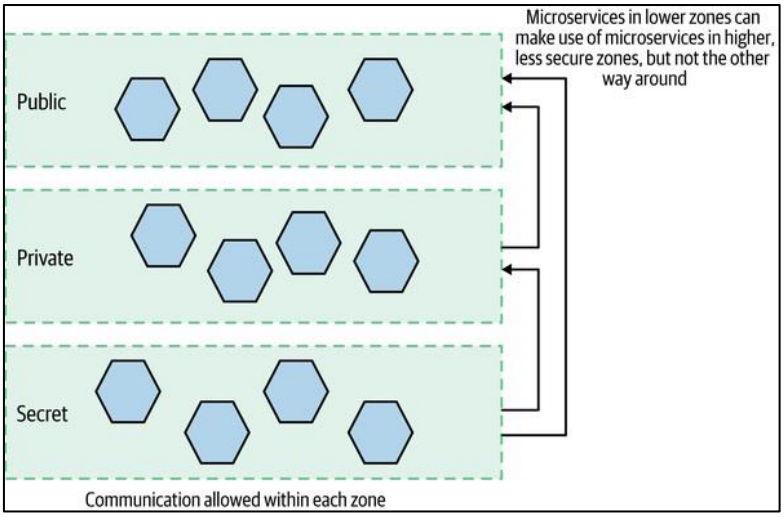
"Nula povjerenja" jeste pretpostavka da se radi o okruženju koje je kompromitovano, sve konekcije mogu da potiču od neprijateljskih servisa, podaci mogu da se pročitaju od strane napadača. Kada se takva pretpostavka uvodi u mikroservisnu arhitekturu, svi dolazeći pozivi od mikroservisa moraju biti provjereni, podaci se moraju sigurno skladištiti, kao i ključevi enkripcije. Ovdje se naravno govori o komunikaciji unutar privatne mreže, što znači da se komunikacija unutar privatne mreže tretira isto kao da dolazi sa Interneta, što može da bude kontraproduktivno. Implicitno povjerenje znači da ako pozivi dolaze unutar obima servisa, oni su implicitno sigurni. Problem je ako napadač uđe u privatnu mrežu, time ugrožavajući servise. Zato je poželjno napraviti balans. Recimo, ukoliko mikroservis radi sa PII podacima, koristi se nula povjerenja, a u ostalim oblastima se koriste blaže restrikcije. Ako se posmatra primjer medicinske ustanove, koja radi sa osjetljivim podacima korisnika, mogu se posmatrati 3 zone (Slika 2.3):

1. Javna zona – podaci koji su svakako dostupni javno.
2. Privatna zona – podaci koji pripadaju korisniku (pacijentu) i dozvoljen je samo autorizovan pristup prijavljenog korisnika (o prijavi će biti riječi kasnije), recimo kakvo zdravstveno osiguranje korisnik posjeduje.
3. Tajna zona – izuzetno povjerljivi podaci dostupni samo na specijalan zahtjev, kao što je medicinski karton pacijenta.

Ideja je da mikroservis koji radi sa javnim podacima bude manje zaštićen, a mikroservis koji radi u dvije zone uzima restriktivniju zonu kao relevantnu.

O mikroservisima će biti više riječi kasnije, s obzirom da će biti obrađeni u praktičnom dijelu rada. [3]

[7] Monolitne aplikacije imaju jednu tačku napada, sa velikom površinom. Mikroservis sa druge strane ima manju površinu napada, ali kada se koristi više mikroservisa, dobija se više tačaka napada sa zajednički velikom površinom.



Slika 2.3 - Zone sigurnosti mikroservisa [3]

3 SIGURNOST NA INTERNETU I ŽIVOT IDENTITETA

Sigurnost informacija je u posljednjih nekoliko desetljeća doživjela dvije velike promjene. Razvoj računara je doveo do potrebe zaštite datoteka i drugih informacija koje se skladište na računarima, naročito ako je riječ o dijeljenim informacijama. Generički naziv za kolekciju alata koji su napravljeni da zaštite podatke i odbiju napade hakera jeste računarska sigurnost.
Druga velika promjena sigurnosti informacija jeste nastala usljed upotrebe distribuiranih sistema i korištenje mreže za dijeljenje informacija između krajnjeg korisnika i računara. Kolekcija alata koji su napravljeni da zadovolje potrebe druge velike promjene se naziva internet[8] sigurnost. Naravno, granice između ova dva termina nisu jasno definisane i uključuju jedna drugu. Fokus Internet sigurnosti jeste sprječavanje, detekcija i korigovanje propusta u sigurnosti koji uključuju prenos informacija. Kao što je u prethodnim poglavljima rečeno, sigurnost ima u osnovi tri cilja koja čine CIA trijadu:

- Povjerljivost – podataka i privatnosti, pri čemu se podaci sakrivaju od neautorizovanih individua.
- Integritet – podataka i sistema, pri čemu se garantuje da se podaci ili sistem mijenjaju samo u specifičnom i autorizovanom pristupu.
- Dostupnost – sistem radi i dostupan je autorizovanim korisnicima.

Dodatni koncepti koji se vezuju za računarsku sigurnost jesu:

- Autentičnost – korisnici su ono što tvrde da jesu, i svi podaci potiču od provjerenog izvora
- Odgovornost (*engl. Accountability*) – cilj koji generiše zahtjev da sve akcije entiteta mogu jedinstveno da se povežu sa tim identitetom. Ovo podržava nemogućnost poricanja (*engl. non-repudiation*), detekciju upada, kao i prevenciju, a ujedno i oporavak kao i zakonsko sankcionisanje. [14]

3.1 Uvod u sigurnost Internet aplikacija

3.1.1 Evolucija Internet aplikacija

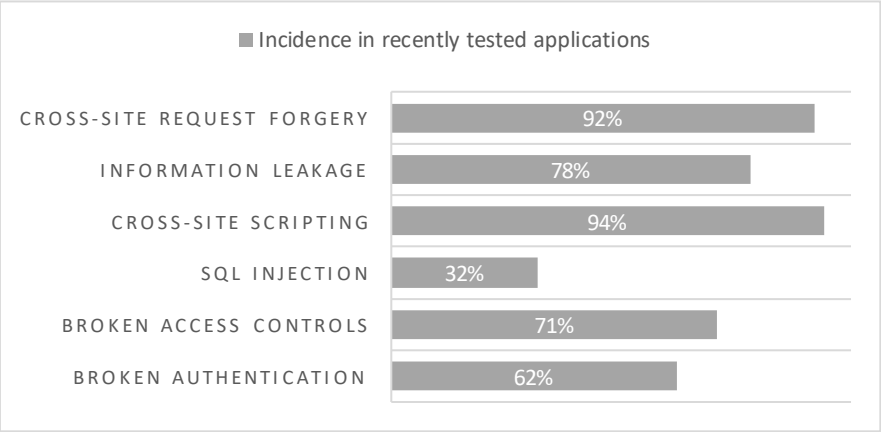
Rani počeci Interneta su vezani za postojanje statičkih Internet stranica, i tok podataka je bio u jednom smjeru, od servera ka Internet pretraživaču. Mnoge stranice nisu autentifikovale korisnike, jer nije bilo potrebe za tim, sve informacije sa servera su već bile javno dostupne. Napadi su se uglavnom svodili na to da napadač kompromituje Web server, i izmijeni sadržaj stranica tako da upućuje na piratski softver koji se nalazi negdje na Internetu ili da ga objavi putem tog Web servera. Danas je situacija neprepoznatljiva. Većina sadržaja na Internetu i većina stranica potiču od Internet aplikacija, koje su uveliko funkcionalne i omogućuju dvosmjernu komunikaciju između pretraživača i Web servera. Informacije koje se pružaju korisniku se generišu dinamički, prilagođavaju se korisniku i često sadrže povjerljive i tajne podatke. Korištenje Internet aplikacija uvodi nove sigurnosne prijetnje, naročito zbog nedovoljne obučenosti programera. Danas napadač može da kompromituje aplikaciju na Internetu, i stekne pristup finansijskim podacima korisnika, privatnim podacima, kao i da izvršni brojne nelegalne akcije. Najčešća primjena Internet aplikacija je u domenima kupovine, društvenih mreža, bankarstva, Web pretrage, kockanja, slanja poruka i elektronske pošte (*engl. email*), kao i interaktivni sadržaj sa različitim temama. Aplikacije kojima se pristupa preko računara su često usko povezane sa mobilnim aplikacijama. Takođe, Internet aplikacije se koriste unutar organizacije kao podrška poslovanju, kao što su aplikacije kao podrška

[8] Termin "internet" je iskorišten za povezane mreže, a termin "Internet" je jedan od načina kako se mreže mogu povezati na globalnom nivou.

kadrovskoj, administraciji, kolaborativni softver u cilju olakšavanja komunikacije unutar organizacije, kao i kompleksne poslovne aplikacije kao podrška odlučivanju. Dakle, dosta aplikacija koje se koriste interno u organizaciji se nalaze ili na nekom privatnom serveru ili na nekom eksternom serveru. Internet aplikacije omogućuju brojne prednosti u odnosu na klasične aplikacije, a neke od njih su: korištenje standardizovanih protokola kao što je HTTP za komunikaciju, korištenje pretraživača kog svaki korisnik ima već instaliran, pa nema potrebe za zasebnim instaliranjem klijentskih aplikacija, lijep korisnički interfejs i moderni Internet pretraživači sa velikim spektrom mogućnosti, kao i veliki broj tehnologija za Internet programiranje. [15]

3.1.2 Napadi na Internet aplikacije

Najveći broj napada na Internet aplikacije jeste vezan za otkrivanje povjerljivih informacija ili zabranjen pristup back-end dijelu aplikacije. Tvrđenje da je sajt siguran[9] samo zato što koristi SSL i sigurne protokole i sertifikate, ili zato što implementira PCI standard je daleko od dovoljnog da bi sajt stvarno bio otporan na napade. Uprkos širokoj upotrebi navedenih mjera zaštite, dijagram 3.1 prikazuje napade[10] identifikovane tokom testiranja Internet aplikacija 2007. i 2011. godine. Iako SSL štiti integritet i povjerljivost podataka koji se dijele između korisnig pretraživača i aplikacije, ne štiti od napada na komponente klijentskog ili serverskog dijela aplikacije, pa ne štiti ni od navedenih napada. [15]



Dijagram 3.1 - Učestalost vrste napada u istraživanju [15]

3.1.3 Ključni problemi sigurnosti

Korisnicima je dozvoljen unos proizvoljnih podataka. Ovaj problem se manifestuje na nekoliko načina, od tog da se korisnicima dozvoljava da pošalju zahtjeve na back-end u bilo kom redoslijedu, da mijenjaju kolačiće, zaglavlja zahtjeva i tijelo, da pristupe aplikaciji preko druge aplikacije, a ne samo pretraživača (DoS napad), modifikuju tokene sesije, dodaju SQL koda u HTML polje forme, namjerno prave greške kako bi se otkrili propusti.

Nedovoljno je razvijena svijest o sigurnosti Internet aplikacija. Programeri nisu dovoljno edukovani po pitanju sigurnosti, zarad kraćeg vremena razvoja posežu za brojnim eksternim bibliotekama sa čijim sigurnosnim propustima nisu upoznati.

Sopstvena implementacija softvera predstavlja rizik po pitanje sigurnosti, zbog potencijalnog uvođenja novog nesigurnog koda. Često je bolje upotrijebiti najbolje moguće

[9] "This Site is Secure" je vidljiv kod stranica koje komuniciraju preko sigurnog protokola.
[10] Tipovi napada neće biti objašnjeni u radu.

gotovo rješenje specijalizovano za određenu svrhu, kao što je korištenje provajdera identiteta koji su testirani godinama sa stotinama klijenata i programera.

Upotreba eksternih razvojnih okvira (*engl. frameworks*) može biti problematična, jer ubrzava programiranje uvođenjem gotovih komponenti i servisa, tako da početnik može da napiše moćnu aplikaciju po pitanju funkcionalnosti, ali slabu po pitanju sigurnosti, ukoliko nema osnovna znanja sigurnosti. To ne znači da razvojni okviri ne treba da se upotrebljavaju, već da se treba zadržati na jednostavnosti aplikacije i usavršavanje tog spektra funkcionalnosti.

Vidljiv je porast novih prijetnji i napada, tako da za vrijeme razvoja aplikacije naučeni napadi više nisu jedini mogući. Zato je neophodan kontinuitet u unapređenju sigurnosti Internet aplikacije za vrijeme cijelog životnog ciklusa.

Nažalost, mnogi timovi i projekti su ograničeni vremenom razvoja i resursa. Potrebno je da kompanija zaposli stručnjaka za sigurnost, ali ne na kraju razvoja, što je često slučaj, već na samom početku dizajna i odabira arhitekture sistema, što je objašnjeno i u prethodnim poglavljima.

Tehnologije nekad razvijene za potrebe Internet programiranja se sada modifikuju koliko je moguće da bi se prilagodile novim zahtjevima tržišta i sigurnosti. To nažalost dovodi do nepredviđenih sigurnosnih propusta.

Još jedan od ključnih problema je porast funkcionalosti aplikacija. Nekada je bilo dovoljno da forma za prijavu sadrži samo korisničko ime i lozinku. Danas sadrže i mogućnosti za oporavak lozinke, korisničkog imena, opciju da se zapamte jedno i drugo. Ovakva stranica aplikacije nesumnjivo promoviše postojanje brojnih sigurnosnih funkcionalnosti, a ipak svaka od njih može biti samostalni servis koji dodaje još jednu tačku za napad aplikacije. [15]

3.2 Osnovni mehanizmi zaštite

Pretpostavka da se nijednom korisniku ne može vjerovati je dovela do porasta broja sigurnosnih mehanizama Internet aplikacija. Iako detalji implementacije i sprovođenja mehanizama variraju, osnovni mehanizmi zaštite su isti i to su:

- kontrola pristupa korisnika aplikaciji (podacima i funkcionalnostima),
- validacija i obrada korisničkog unosa kako bi se aplikacija sačuvala od nedefinisanog ponašanja za određeni unos koji može ugroziti sigurnost podataka i aplikacije,
- preduzimanje odgovarajućih odbrambenih i napadačkih mjera da bi se usporio ili iznervirao napadač,
- omogućavanje monitoringa aplikacije administratorima.

Ukoliko se primjene osnovni mehanizmi zaštite, sam napadač mora da ih poznaje kako bi imao šansu da izvede uspješan napad, čiji je cilj pristup podacima ili izvršavanje neovlaštenih transakcija u poslovnoj Internet aplikaciji. [15]

3.2.1 Kontrola pristupa

Centralni zahtjev po pitanju sigurnosti svake poslovne aplikacije na Internetu jeste kontrola pristupa. Tipična situacija jeste veći broj različitih kategorija korisnika, kao što su anonimni korisnici, obični registrovani korisnici i administratori. Većina Internet aplikacija sprovode kontrolu pristupa kroz autentifikaciju, autorizaciju i upravljanje sesijom. Budući da su prethodne komponente povezane, sama sigurnost jeste jednaka sigurnosti najslabije karike u lancu. Fokus ovog poglavlja jeste opis različitih kontrola pristupa. [15]

Autentifikacija je proces potvrđivanja legitimiteta korisničkog identiteta. Autentifikacija je bazirana na jednom ili više sljedećih faktora: nešto što korisnik zna (lozinka, PIN), nešto što korisnik ima (pametna kartica, ključ) i nešto što korisnik jeste (otisak prsta, lice). [16]

Svi korisnici bi bili anonimni bez ovog koraka. Najčešće se koristi nešto što korisnik zna, obično spoj korisničkog imena i lozinke. Danas je takođe popularna multifaktorska autentifikacija. O autentifikaciji će biti više riječi poslije.
Upravljanje sesijom je bitno kod Internet aplikacija, jer nakon što se korisnik prijavio[11], počinje slanje većeg broja HTTP zahtjeva. Istovremeno, tipična Internet aplikacija prima veći broj zahtjeva od različitih korisnika (prijavljenih ili anonimnih). Da bi se uopšte implementirala kontrola pristupa, potrebno je naći način da se identifikuje i obradi serija zahtjeva jednog jedinstvenog korisnika. Internet aplikacije rješavaju ovaj problem dodjeljivanjem jedinstvenog tokena svakom korisniku. Sesija sama po sebi je set struktura podataka koje prate trajanje korisničke interakcije sa aplikacijom. Token je obično jedinstveni niz karaktera koji klijentska aplikacija automatski treba da šalje prilikom svakog zahtjeva (kroz zaglavlje ili kolačić). Sesija se može prekinuti ukoliko korisnik određeno vrijeme nije poslao nijedan zahtjev, a da bi se uspostavila nova sesija, potrebno je proći kroz proces prijave na sistem. Sigurnost sesije najviše zavisi od sigurnosti tokena i upravljanja tokenom. Token može da se čuva na klijentskoj ili na serverskoj strani. U poglavlju o protokolima za autentifikaciju i autorizaciju biće više riječi o tokenima sesije.
Treći korak u prethodno navedeno lancu kontrole pristupa jeste sama kontrola pristupa, odnosno, autorizacija. Potvrđen je identitet korisnika, aplikacija vodi računa o sesiji korisnika, ali sve to ne bi imalo smisla ako svako može da pristupi svim podacima. Autorizacija ili kontrola pristupa ograničava korisnika na pristup samo dozvoljenim resursima, bilo da su to akcije ili podaci. Aplikacija može da podržava različite uloge korisnika koje sadrže različite kombinacije specifičnih privilegija. Zbog kompleksne prirode dizajna i implementacije autorizacije često je tačka napada. [15]

3.2.2 Vrste kontrole pristupa

Skoro svaka kontrola pristupa se može formulisati upotrebom termina korisnik, subjekat, objekat, operacija i privilegija, i entiteta između ovih termina. [16]
Neke od vrsta kontrola pristupa koje će biti ukratko objašnjene su kontrola pristupa bazirana na ulogama, atributima, pravilima, jedinstvenoj prijavi, diskreciona i obavezna kontrola pristupa.

3.2.2.1 Diskreciona i obavezna kontrola pristupa (DAC, MAC)

Definisani su još 1983. godine kao dio vojnog programa. Osnovna osobina DAC kontrole pristupa jeste restrikcija pristupa na osnovu vlasništva nad objektom. Vlasnik objekta je taj koji određuje privilegije pristupa objektu i otuda naziv diskreciona kontrola pristupa. Jedan od funkcionalnih nedostataka DAC jeste to što mnogi korisnici nisu vlasnici podataka kojima pristupaju i nije efikasno da im korisnici odobravaju pristup pojedinačno.
MAC kontrola pristupa je način ograničenja pristupa objektima na osnovu osjetljivosti informacija[12] sadržanih u istim i oslobađanje subjekta od informacija takve osjetljivosti, odnosno, centralni sistem je taj koji određuje prava pristupa korisnicima. Osjetljivost i povjerljivost podataka su od najvećeg značaja i zbog toga se koristi u vladinim i vojnim sistemima. Postoji više MAC modela, ali neće biti obrađeni u radu. [16]

[11] Prijaviti se na sistem ili se autentifikovati.
[12] Oznaka može da bude u tekstualnoj formi (*engl. label*).

3.2.2.2 Kontrola pristupa zasnovana na ulogama (RBAC)

RBAC predstavlja veliko unapređenje u fleksibilnosti i detaljnosti u odnosu na DAC i MAC, jer je prilagođeniji komercijalnoj upotrebi. Ipak, RBAC se često posmatra kao posebna forma MAC kontrole pristupa.
Pristup resursima Internet aplikacije zavisi od uloge korisnika u organizaciji. Uloge imaju različite privilegije, odnosno, permisije i odgovornosti. Permisije se mogu posmatrati kao atomska jedinica rada, kao što je recimo pravo čitanja podataka. Kod RBAC, permisije se ne dodjeljuju korisnicima, već ulogama, a uloge se dodjeljuju korisnicima. Prednost je lakše ažuriranje permisija. Druga prednost RBAC jeste mapiranje uloga aplikacije na uloge koje postoje u organizaciji i biznis logici. Postoje tri osnovna pravila:

1. Dodjeljivanje uloga: Subjekt može da izvrši operaciju samo ako je korisnik selektovao ulogu ili mu je uloga dodijeljena. Proces autentifikacije i registracije se ne smatra operacijom, dok se sve druge aktivnosti korisnika smatraju. Ovo pravilo osigurava da svaki korisnik ima neku aktivnu ulogu.
2. Autorizacija uloga: Aktivna uloga subjekta mora biti dodijeljena datom korisniku. Time se garantuje da sa pravilom 1, korisnik može da uzme samo uloge koje su mu provjereno dodijeljene.
3. Autorizacija operacije: Subjekt može da izvrši operaciju samo ako je ta operacija dozvoljena za datu ulogu. Ovo pravilo sa pravilima 1 i 2 osigurava da korisnik može da izvrši samo one operacije koje su podržane njegovom ulogom.

Ukoliko postoji hijerarhija uloga, uloge djeca nasljeđuju privilegije uloge roditelja. [16]

3.2.2.3 Kontrola pristupa zasnovana na atributima (ABAC)

Kontrola pristupa zasnovana na atributima uvodi fleksibilnost provjeravajući različite atribute prije donošenja odluke o dozvoli pristupa. RBAC nosi svoje prednosti zbog postojanja uloge, ali ukoliko se permisije trebaju zasnivati ne samo na ulozi, već i na drugim karakteristikama, kao što je lokacija, specijalni trening ili slično, onda ABAC nosi svoje prednosti.
Formalna definicija bi bila: ABAC jeste kontrola pristupa gdje subjektovi zahtjevi izvršavanja operacije nad objektom zavise od atributa dodijeljenih subjektu, objektu, uslova okruženja ili skupa polisa[13] koje su specifikovane u terminima tih atributa i uslova. Atributi su karakteristike pomenutih, i sadrže informacije u obliku naziv-vrijednost. U poslovnim aplikacijama bitno je uspostaviti validnu arhitekturu kontrole pristupa, koja će da definiše atribute i polise, održava repozitorijum tih atributa kao i upravlja servisima preko kojih se do tih atributa može doći, bilo da se dodijele korisniku ili da se provjere. Naravno, korisnicima nije dostupno da vide atribute objekata, već samo atribute dodijeljene njima. Kako korisnik može da ponavlja zahtjev, uočavajući pravila koja su dovela do odbijanja pristupa, kompanija treba da vodi računa o praćenju zahtjeva koji pristižu. Zavisno od veličine poslovnog procesa, bitno je omogućiti skaliranje i distribuciju arhitekture ABAC, vodeći naravno računa o sigurnosti takve arhitekture. Zbog toga se često pribjegava gotovim rješenjima. [17]
Primjer ABAC bi bila situacija gdje je medicinskim sestrama dozvoljen pristup sistemu samo unutar radnih sati. [18]

[13] Polisa jeste reprezentacija pravila ili veza koje omogućuju određivanje autorizovanja neke radnje.

3.2.2.4 Kontrola pristupa zasnovana na pravilima (RuBAC)

Kontrola pristupa zasnovana na pravilima, kao što joj samo ime kaže, upravlja pristupom na principu seta predefinisanih pravila i permisija. Ukoliko korisnik ne zadovoljava ta pravila, smatraće se uljezom i biće izbačen sa mreže. Najčešće se koristi kod pristupa lokacijama, bazama podataka i drugim specifičnim resursima kompanije. Može da ima veću snagu od RBAC, jer određena pravila koja administrator postavi mogu biti važnija od uloge pojedinca. Može biti komplikovana za implementaciju i održavanje, ali pruža dobar mehanizam zaštite od napada za situacije koje će se rijetko mijenjati u budućnosti. Pravila mogu biti statička, dinamička i implicitna pravila odbijanja. [16] [19]

3.2.2.5 Kontrola pristupa zasnovana na jedinstvenoj prijavi (SSOAC)

Jedinstvena prijava (*engl. Single Sign-On*) je tehnologija koja koristi postojeće korisničke naloge da bi potvrdio identitet korisnika i kontrolisao pristup podacima za različite servise. [20] Može da se podijeli u tri velike grupe: Web, federalna i desktop SSO. Organizacije obično imaju sva tri tipa, a SSO sam po sebi je uklopljen u postojeću kontrolu pristupa.
Web SSO pruža SSO prijavu između servera zaduženog za sigurnost koji vodi registar korisnika i back-end-a Web aplikacije. Eliminiše se potreba da se korisnik dva puta prijavljuje kada potražuje resurs sa datog sigurnosnog servera.
Federalna SSO pruža SSO prijavu između Web aplikacija od povjerenja koristeći odvojene registre korisnika, bilo u organizaciji ili u kombinaciji više organizacija.
Desktop SSO ili poslovni SSO, dozvoljava jednostavno i transparentno SSO iskustvo mnogim aplikacijama, ne samo Internet. Omogućava korisnicima da se sa jednom lozinkom prijave na postojeće klijentske aplikacije, kao što su email klijenti i .NET aplikacije.
SSO olakšava korisnicima korištenje mnogobrojnih aplikacija upotrebom samo jedne prijave i jedne kombinacije kredencijala. Korisnicima je na ovaj način omogućeno da odaberu jednu kompleksnu lozinku, umjesto mnogih kompleksnih koje bi možda završile zapisane na papiru, na taj način stvarajući brojne ranjivosti. [21] Prema istraživanju iz 2015. godine, najčešće korištene lozinke su: *123456 i password*.[14] SSO može da se implementira uz pomoć protokola kao što su OpenID, OpenID Connect OAuth 1.0, OAuth 2.0. [20] Više riječi o SSO će biti u poglavlju 4.6.

3.2.3 Obrada korisničkog unosa na klijentu i serveru

Zarad sigurnosti, aplikacija treba da obrađuje korisnički unos, pri čemu se pod obradom podrazumijevaju različiti mehanizmi validacije, transformacije i ograničavanja unosa. Klijentska aplikacija može da zahtijeva od korisnika da unese lozinku od 8 karaktera, može da zabrani korištenje određenih JavaScript izraza ili drugih simbola, ali u slučaju da se od korisnika zahtijeva proizvoljan unos, takva ograničenja nisu poželjna. Ukoliko se govori o skrivenim poljima koje korisnik ne može da izmijeni na uobičajen način kroz pretraživač, server može da detektuje te izmjene i reaguje na potencijalni napad bilježenjem u bazu ili zabranom pristupa. Dakle, mehanizmi obrade unosa su sljedeći:

Postojanje crne liste koja sadrži riječi, fraze ili obrasce koji nisu dozvoljeni, jer se često koriste u napadima. Prilikom izrade filtera potrebno je razumijevanje napadača i predviđanje mogućih izmjena riječi koja bi semantički trebala da bude zabranjena, ali nije identifikovana kao jedna od riječi crne liste (npr. “seLEct” umjesto “SELECT”).
Još jedna česta ranjivost ovog pristupa jeste u nultim (*engl. null*) vrijednostima (npr. “%00<script>alert(1)</script>)”.

[14] Istraživanje je sprovedeno na osnovu podataka lozinki koje su provaljene, i to je potrebno uzeti u obzir.

Drugi mehanizam jeste suprotan od prethodnog i podrazumijeva postojanje bijele liste, koja sadrži obrasce koje aplikacija prihvata smatrajući da su benigni. Vrlo je efikasan metod, ali postoje nedostaci, kao sprječavanje unosa apostrofa ili crtice pri unosu imena, koji može da se iskoristi za napad na bazu, iako nečije ime sadrži pomenuti karakter.

Sanitacija unosa podrazumijeva uklanjanje potencijalnih prijetećih karaktera, ostavljajući samo onaj dio unosa koji je siguran, enkodovanje unosa ili upotrebom isključnih karaktera. Uobičajen način za izbjegavanje XSS napada je HTML enkodovanje opasnih karaktera prije nego što se ugrade u dio stranice.

Sigurno manipulisanje i rad sa podacima je drugi način zaštite od lošeg unosa. Na primjer, napad ubacivanja SQL koda može da se izbjegne parametrizacijom upita nad bazom.

Semantičke provjere uskaču gdje ostali mehanizmi obrade unosa padaju, a to je semantička provjera unosa. Ukoliko korisnik promijeni skriveni parametar u zaglavlju zahtjeva, kao što je broj naloga, a pošalje zahtjev za provjerom stanja na računu, potrebno je provjeriti da li je broj korisnika isti kao broj za koji korisnik zahtijeva provjeru stanja računa.

Svaka komponenta aplikacije bi trebala da radi validaciju prije izvršavanja operacije, podrazumijevajući da unos nije siguran, makar bio obrađen u prethodnoj komponenti u lancu. [15]

3.2.4 Obrada greške i održavanje revizionih logova

Aplikacija ne treba nikada da vrati sistemske odgovore korisniku, jer korisnik može da iz poruke dobije dovoljno informacija da izvrši neki napad i ukrade podatke. Zbog toga je bitno uraditi obradu greške prije slanja informacija korisniku.

Revizioni (*engl. audit*) logovi bilježe korisničku aktivnost, koja se možda čini irelevantna, ali je bitna za forenzičku obradu, naročito kod aplikacija koje zahtijevaju visok stepen sigurnosti. Bankovne aplikacije treba da pružaju čitav forenzički zapis aktivnosti korisnika, od prijave na sistem do odjave sa sistema. Najmanje treba da se zapišu aktivnosti kao što su uspješna ili neuspješna prijava, bankovni transferi, svaki pokušaj neautorizovanog pristupa ili svaki zahtjev koji sadrži riječi koje mogu da izazovu sumnju u namjere korisnika. [15]

3.3 Život identiteta

Upravljanje identitetom je jednostavno u teoriji, ali komplikovano u praksi. Ne postoji jedinstveno rješenje za svaki sistem. Ukoliko je riječ o Internet aplikaciji, korisnik može očekivati mogućnost prijave upotrebom nekog poznatog provajdera, kao što je *Facebook*. Ukoliko korisnik treba da pristupi brojnim aplikacijama unutar kompanije, SSO može da mu olakša posao. Aplikacija sa osjetljivim sadržajem zahtijeva jaču zaštitu podataka, bilo da se koristi dodatni hardverski metod autentifikacije kao što je kartica ili multifaktorska autentifikacija. Pored svega ovog, arhitektura koja se koristi za implementaciju autentifikacije i autorizacije treba da se lako skalira. Loše iskustvo korisnika prilikom registracije i prijave na sistem može da loše utiče na doživljaj sistema, a pored tog je potrebno voditi računa o zaštiti podataka i poštovanje zakona kao što je GDPR. Implementiranje sopstvenog sistema za pomenuto može biti komplikovano i dovodi do sigurnosnih propusta, pa je preporuka da se koriste specijalizovane aplikacije i standardi, kao što su provajderi identiteta koji rade sa standardizovanim protokolima. [22]

3.3.1 Definicije i događaji u životnom ciklusu identiteta

Termin identitet jeste kolekcija atributa i identifikatora vezanih za specifičnu osobu ili entitet u određenom kontekstu. Osoba može da ima više identiteta, recimo, poslovni i identitet na mreži (*engl. online*). Atributi identiteta osobe mogu da budu ime i godište, a atributi identiteta

mašine mogu da budu IP adresa, lokacija i operativni sistem. Nalog se definiše kao lokalna konstrukcija unutar sistema koja se koristi za sprovođenje operacija u sistemu. Identitet ili referenca na isti se mogu skladištiti unutar naloga. Nalog može da se referencira uz pomoć sopstvenog identifikatora. Sistem za upravljanje identitetom ili provajder identiteta jeste set servisa zaduženih za kreiranje, izmjenu i uklanjanje identiteta i povezanih naloga, kao i proces autentifikacije i autorizacije.
Događaji u životu identiteta su kreiranje identiteta, autorizacija, autentifikacija, primjena politike kontrole pristupa, sesija, SSO, jača autentifikacija, odjava, upravljanje nalogom i uništavanje identiteta.

3.3.2 Kreiranje identiteta

Prvi korak u životu identiteta jeste kreiranje identiteta. Podrazumijeva registraciju korisnika, uvoz informacija o identitetu sa starijeg sistema ili dobavljanje identiteta od eksternog sistema. Zajedničko svim načinima jeste postojanje jedinstvenog identifikatora identiteta i opciono identifikatora naloga, kreiranje naloga i vezivanje identiteta za nalog. Recimo, Alisa u banci može da ima više naloga i identiteta, jedan privatni i jedan poslovni nalog.

3.3.3 Autorizacija naloga

Kada je nalog kreiran, bitno je specifikovati privilegije. Autorizacija može da se posmatra kao dodjeljivanje privilegija, kao što je pomenuto u poglavlju 3.2.1.

3.3.4 Autentifikacija, kontrola pristupa i sesija

Da bi pristupila sadržaju Internet aplikacije, osoba treba da se autentifikuje. Korisnik može da unese kredencijale ili identifikator vezan za nalog sa kojim želi da pristupa sistemu, a podrazumijeva se da je nalog kreiran. Sam proces autentifikacije je pomenut u poglavlju 3.2.1. Alisa ima više naloga u banci, sa različitim korisničkim imenima. Ukoliko želi da pristupi sistemu, treba da u posebnu formu na Web stranici unese lozinku koja pokazuje njeno pravo da koristi nalog i korisničko ime koje određuje sa kojim nalogom pristupa sistemu. Da bi se osiguralo da Alisa može da vrši transakcije samo sa svog naloga, potrebno je da bankovni sistem ima implementiranu kontrolu pristupa, kao što je objašnjeno u poglavlju 3.2.2. Svaki pristup tuđim podacima bi joj bio zabranjen uz poruku na ekranu i sistemsko zapisivanje pokušaja neovlaštenog pristupa. Kao što je pomenuto ranije, sesija omogućava korisniku da izvršava brojne operacije na nalogu od momenta prijave do momenta odjave sa sistema ili do prekida sesije. Sesija vodi računa o tom kada se korisnik autentifikovao i na koji način. Vrijeme koje je dozvoljeno korisniku da provede neaktivan jeste limit sesije ili istek sesije. Pomaže da se spriječi manipulisanje korisničkim nalogom kada vlasnik naloga nije prisutan za ekranom. Na primjer, Alisina bankovna aplikacija nudi kratku sesiju, ali jedan od servisa koje banka pruža nudi dužu sesiju, jer samo objavljuje novosti u banci.
Ukoliko Alisa želi da pristupi servisima banke, može da koristi SSO (poglavlja 3.2.2.5, 4.6), tako da joj je omogućeno da se prijavi samo jednom, a da pristupa i svom nalogu i investicionom blogu. Takva sesija se zove SSO sesija. [22]

3.3.5 Multifaktorska autentifikacija (MFA)

Neke forme autentifikacije, kao što su *korisničko_ime-lozinka* autentifikacija, se smatraju slabim, jer uključuju samo jedan faktor. Jače forme autentifikacije uključuju više faktora, kao što su nešto što korisnik jeste i nešto što korisnik zna. Autentifikacija koja uključuje više faktora se naziva multifaktorska autentifikacija. Nije nužno da se sprovodi odmah prilikom

prijave. Korisnik može da se prijavi na sistem uz pomoć jednog faktora, a prilikom pristupa restriktivnijem dijelu sistema, traži se dodatni faktor autentifikacije. [22]

3.3.6 Odjava i uništavanje identiteta

Odjava najmanje podrazumijeva završetak rada sa aplikacijom i prekid sesije na zahtjev korisnika. Da bi ponovo koristio aplikaciju, korisnik treba da se ponovo autentifikuje. Ukoliko se koristi SSO, odjava podrazumijeva prekid samo nekih ili svih otvorenih sesija, što je na odluci dizajnera sistema. Odjava se razlikuje od implicitnog isteka trajanja sesije zbog toga što istek trajanja sesije može da ostavi sesiju samo u neaktivnom stanju koje se ponovo aktivira autentifikacijom korisnika.
Korisniku treba da je omogućeno da zatvori svoj nalog. To podrazumijeva uništavanje identiteta i korisničkog naloga, tako da ne može ponovo da se koristi. Nalog može da se obriše trajno ili da se deaktivira, tako da se podaci sačuvaju u svrhu revizije. [22] Da bi aplikacija poštovala GDPR, svi korisnički podaci treba da se obrišu kada korisnik to zahtjeva, odnosno da se ispuni pravo korisnika da bude zaboravljen. [23]

3.3.7 Upravljanje nalogom i oporavak

Nekada je potrebno da se nalog ažurira, recimo da se promijene adresa stanovanja ili broj telefona. U kompaniji je potrebno da se promijeni pozicija, zvanje ili privilegije. Upravljanje nalogom se sastoji od procesa koji omogućuju korisnicima i administratorima da vide i ažuriraju atribute identiteta.
U nekim slučajevima je poželjno omogućiti korisniku da povrati pristup nalogu ako je zaboravio lozinku ili da promijeni lozinku na neki siguran način u slučaju zaboravljanja iste. Taj proces se zove oporavak naloga. [22]

4 AUTORIZACIJA I AUTENTIFIKACIJA

Internet pretraživač definiše mehanizam u kom je korisniku omogućeno da se autentifikuje nekoj Web stranici ili HTTP API-ju putem HTTP protokola korištenjem *Authentication* zaglavlja, a takav vid autentifikacije se naziva *HTTP Basic Auth*. Korisničko ime i lozinka se šalju u formi *korisničko_ime:lozinka* koja se dodatno Base64 enkoduje. Ovakav način autentifikacije nije siguran i podložan je napadima. Primjer slanja zahtjeva je prikazan u isječku koda:

```
GET /someresource
Host: www.somehost.com
Authorization: Basic bXl1c2VybmFtZTpteXBhc3N3b3Jkcw==
```

Nešto sigurnija autentifikacija bi bila *HTTP Digest Auth*, koja koristi MD5 heširanje kredencijala da bi se izbjeglo slanje istih u golom tekstu. Iako jeste naprednija od prethodno pomenute, i dalje je podložna napadima presretanja. *Form-Based Auth* omogućuje korištenje HTML forme za slanje kredencijala. Ukoliko se ne koristi HTTPS, korisničko ime i lozinka će biti vidljivi presretaču. Standardni protokoli koji će biti objašnjeni u narednom poglavlju najčešće koriste *Bearer* autentifikaciju. To je vid autentifikacije koji koristi sigurnosne tokene – Bearer tokene, koji se šalju preko *Authorization* zaglavlja. Sigurnosni token jeste string u određenoj formi, koji može biti nasumično generisan ili poseban JWT token:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiO
iIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJS
MeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Kada napadač dođe u posjed ovog tokena, može da pristupi resursima kojima nosilac tokena može da pristupi ili da modifikuje sam token. Pored tog, može da pokuša da iskoristi stari token. Može da pokuša i redirekciju ka drugom serveru korištenjem ukradenog tokena za lažno predstavljanje. Napadač može doći u posjed osjetljivih informacija, ako se iste skladište u tokenu, a token nije enkripotovan. Mjere zaštite tokena bi bile sljedeće: korištenje SSL/TLS enkripcije, ograničenje područja primjene tokena, korištenje strukturisanog tokena kao što je JWT sa kratkim trajanjem. Ukoliko se tokeni čuvaju u bazi, najbolje je da se čuvaju heširani, pri čemu upotreba salta nije obavezna, jer su tokeni prilično nasumični. Takođe, preporučljivo je logovanje i monitoring zahtjeva da bi se uočile nepravilnosti kod izdavanja, korištenja ili povlačenja tokena, naravno, ne sadržeći token u sebi. [24]

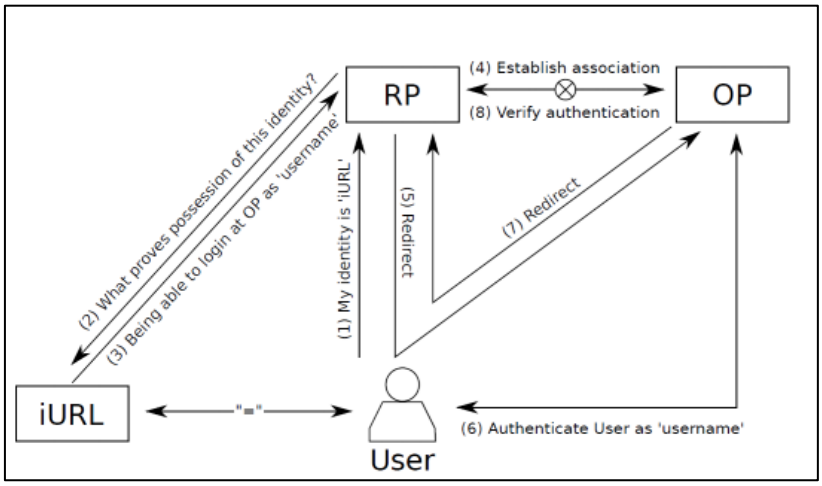
4.1 Standardizovani protokoli za autentifikaciju i/ili autorizaciju

Zašto koristiti standardizovane protokole za autentifikaciju i autorizaciju? Kao otvoreni standardi softverske industrije, ovi protokoli su podvrgnuti javnoj kritici mnogih ljudi, kako početnika tako i eksperta u oblasti sigurnosti. Zbog toga imaju mnogo manje ranjivosti nego nešto što bi sami razvili. Imaju široku upotrebu u svijetu, tako da omogućavaju interoperabilnost između naše aplikacije i provajdera servisa koji podržavaju protokol. Ukoliko aplikacija pristupa korisničkim podacima sa nekog od servisa kao što je Google, potrebno je koristiti standardizovani protokol koji taj servis implementira. Slično, ako se naša aplikacija koristi u sklopu većeg poslovnog sistema, taj veći poslovni sistem može da očekuje da naša aplikacija poštuje neki od standarda. Mnogi od standardizovanih protokola podržavaju SSO, što je olakšica korisnicima. I posljednja prednost bi bila ušteda vremena jer mnogi programski paketi već podržavaju neki od ovih protokola. [22]

4.2 OpenID autentifikacija

OpenID protokol je otvorena specifikacija za autentifikaciju i SSO, kreirana 2005. godine. Rijetko je u upotrebi, ali ima istorijski značaj.
Dokazuje da je korisnik vlasnik identifikatora, bez razmjene kredencijala, odnosno, uz posredstvo treće strane – OpenID provajdera (OP). Decentralizovan je, pri čemu je korisniku omogućeno da odabere provajdera identiteta bez da napušta trenutnu stranicu. Radi nad jednostavnim HTTP(S) protokolom, koristeći se dostupnim funkcionalnostima istog, dakle, ne zahtijeva specifičnu implementaciju Internet pretraživača ili klijentske aplikacije. Glavna uloga OpenID autentifikacije jeste da pruži prenosiv, korisnički-orjentisan digitalni identitet u slobodnoj i decentralizovanoj formi. Protokol (pojednostavljeno) funkcioniše na sljedeći način:

1. Krajnji korisnik inicira proces autentifikacije dajući korisnički isporučiv identifikator aplikaciji koja ga traži tj. pouzdanoj strani RP (*engl. Relying party*) preko Internet pretraživača.
2. Nakon normalizacije identifikatora, RP obavlja pretragu potrebu za obavljanje zahtjeva ka OP.
3. Rezultat pretrage je OP endpoint sa verzijom protokola.
4. RP i OP uspostavljaju asocijaciju – dijeljenu tajnu komunikaciju preko Diffie-Hellman razmjene ključeva. Ovu asocijaciju OP koristi da potpiše sekvencijalne poruke koje RP verifikuje.
5. RP preusmjerava Internet pretraživač krajnjeg korisnika na OP sajt sa OpenID autentifikacionim zahtjevom.
6. OP provjerava da li je korisnik autorizovan da obavi OpenID autentifikaciju i da to želi.
7. Nakon (ne)uspješne autentifikacije, OP preusmjerava pretraživač nazad ka RP sa porukom da je identitet dokazan ili da je autentifikacija neuspješna.
8. RP provjerava povratne informacije OP-a tako što provjerava povratni URL, poruku, provjerava *nonce* i verifikuje potpis sa uspostavljenom asocijacijom ili slanjem direktnog zahtjeva OP-u. [25] Slika 4.1 prikazuje pomenutu komunikaciju.



Slika 4.1 OpenID proces autentifikacije [26]

Iz perspektive korisnika, OpenID funkcioniše na sljedeći način: korisnik želi da se registruje na servis *springnote.com*. Umjesto da popunjava novu registracijsku formu, korisnik pruža datom servisu URL koji reprezentuje njegov identitet: *www.johndoe.com*. Preko ovog URL-a, servis *springnote.com* otkriva kako da verifikuje korisnički identitet. U ovom primjeru, servis je otkrio da vlasnik ovog URL-a treba da se uloguje na *openid.yahoo.com* servis sa korisničkim imenom *jdoe*. Zbog toga se vrši preusmjeravanje korisnika na ekran za prijavu OP-a gdje

korisnik za korisničko ime *jdoe* unosi lozinku. Nakon prijave na OP servisu, OP pita korisnika da li želi da koristi svoj identitet za servis *springnote.com*. Ako korisnik potvrdi, vrši se preusmjeravanje na stranicu *springnote.com*. Dati servis je sada u mogućnosti da kreira identitet za datog korisnika na osnovu identiteta sa drugog servisa. [26]

4.2.1 Sigurnosni problemi

Sigurnosni propusti do kojih može doći sa korištenjem OpenID su prisluškivanje saobraćaja, XSS napad, izmjena korisničke sesije, skeniranje portova, DoS napad u fazi otkrivanja, „Čovjek u sredini" napad (*engl. Man in The Middle*), trovanje asocijacije, recikliranje OpenID-a, pecanje (*engl. Phishing*) i lažiranje oblasti (*engl. realm*).
Upotrebom SSO koncepta, korisnik koristi OP da bi se prijavio ili registrovao na neku drugu aplikaciju. OP čuva korisničke kredencijale. Ukoliko napadač uspješno napadne OP, svi servisi povezani sa tim korisničkim nalogom su kompromitovani i to bez zamjene lozinke. Sa popularnosti OpenID standarda rastu i mjere zaštite, ali je ovo ipak jedna od stvari koje treba imati u vidu prilikom izbora provajdera.
OpenID provajder može da evidentira svaku transakciju sa svakim RP koje korisnik posjeti. Uz pomoć alata za obradu velike količine podataka, napadač može da zloupotrijebi zapise do kojih dođe sa uspješnim napadom.
Korisnik može da se ne odjavi sa OpenID servisa, pri čemu može da pristupi RP servisu bez ponovne forme za prijavu. Napadač može da izvrši napredni XSS napad tako što ugradi skriveni okvir u OpenID prijavnu formu koja se otvara sa RP stranice, bez da je korisnik svjestan napada. Zatim, napadač može da izvodi lažiranje zahtjeva u ime korisnika, tj. CSRF napad. Poželjno je odjaviti se sa OP u slučaju završetka rada.
Budući da korisnik sam unosi URL, može da unese lokalnu adresu RP-a zajedno sa portom, i da vrši inspekciju portova koje kasnije može da zloupotrijebi u daljim napadima.
Napadač takođe može da iskoristi korisnički isporučiv URL tako da sadrži veliku količinu podataka onesposobljujući RP. Naravno, i ovakav napad je lako spriječiti istekom zahtjeva.
„Čovjek u sredini" napad se izvodi zloupotrebom Diffie-Hellman razmjene ključeva. Prisluškujući saobraćaj između RP i OP[15] napadač može da izmjeni podatke u komunikaciji (bez da RP primjeti da potpis nije korektan). Ovim napadom, napadač može da se prijavi na RP kao bilo koji korisnik ili da se prijavi kao bilo koji korisnik sa jednog OP na bilo koji RP.
Jedan od specifičnih OpenID napada jeste trovanje asocijacije. Napadač napravi lažnog OpenID provajdera. Prisluškujući komunikaciju sazna naziv asocijacije između RP i pravog OP kog korisnik koristi. Napadač zatim uspostavlja komunikaciju sa RP, i kao naziv asocijacije šalje ukradeni podatak. Nekada validna asocijacija je sada kompromitovana, jer je RP prepisao MAC ključ sa ključem kog je napadač specifikovao kao i OpenID provajdera za dati naziv asocijacije. Napadač sada može da koristi asocijaciju dok god se stvarni OpenID provajder ponovno ne uključi u proces. Rezultat ovog napada je ili odbijanje komunikacije sa dobrim OP, jer je potpis identifikovan kao nevalidan ili krađa identiteta. Zbog ovih i sličnih napada je bolje koristiti provjerene biblioteke nego svojeručno implementirati protokol. Poželjno je takođe da se za svaki novi RP korisnik mora prijaviti na OP ponovo. Mnoge su metode koje ili smanjuju rizik od nekog napada ili eliminišu, kao što je prethodno pomenuti redukovani SSO, korištenje HTTPS-a, provjera *return_to* polja, standardizovani identifikator kao i lista validnih OP i RP. [26]

[15] Podrazumijeva se da napadač ima pristup ruteru ili DNS serveru da bi prisluškivao saobraćaj između ova dva entiteta.

4.3 OAuth 2.0 autorizacija

OpenID je nastao iz potrebe da se riješi problem autentifikacije na elegantan i standardizovan način. OAuth protokol, s druge strane, je nastao sa ciljem rješavanja drugih problema koji su se javili sa povezivanjem više servisa/aplikacija na Internetu.
Prije OAuth sigurnosnog protokola, postojale su situacije slične situaciji navedenoj u sljedećem primjeru. Korisnik koristi jednu aplikaciju za skladištenje svojih slika i drugih dokumenata različite namjene. Da bi odštampao jedan svoj album, korisnik koristi drugi servis koji treba da pristupi servisu za skladištenje dokumenata. Servisu za štampanje treba dozvola za pristup korisničkim resursima na drugom servisu, i zato traži od korisnika njegove kredencijale koje će da koristi za pristup tom servisu, ili svaki put na zahtjev korisnika ili kad god želi. Neki od bitnih problema kod ovog pristupa, koji su se dešavali sa brojnim aplikacijama i servisima, jeste to što je jednom servisu potreban pristup određenim resursima korisnika na drugom servisu, a dobija korisničke kredencijale koje će da kopira na drugi servis (često u obliku čistog teksta, što je drugi sigurnosni problem) za pristup svim resursima. Dakle, korisnik je dao pravo servisu da ga imitira kod pristupanja drugom servisu. Ukoliko korisnik želi da ograniči servis na isključivo čitanje slika, a ne čuvanje slika, nema način da to uradi. Da bi zabranio pristup datom servisu, potrebna je promjena kredencijala. Ne treba napominjati kakvi problemi mogu nastati ukoliko korisnik iste kredencijale čuva kod brojnih klijenata, i kako je nemoguće postaviti neka ograničenja. Jedno od rješenja za jedan od problema sigurnosti jeste da postoji poseban ključ koji će da uspostavi dozu povjerenja između servisa, što je u praksi skoro ne primjenljivo. Drugi način bi bio da korisnik ima posebnu lozinku koju dijeli sa drugim servisima. Dio problema bi bio riješen, ali pod cijenu tog da korisnik mora da upravlja tim posebnim kredencijalom, što je opet sigurnosno problematično. [24]
OAuth 2.0 Autorizacioni Okvir[16] (*engl. OAuth 2.0 Authorization Framework*) je objavljen 2012. godine sa ciljem da se aplikacija autorizuje za pristup drugim aplikacionim API-jima. [22] Iako se često naziva autorizacionim protokolom, može se posmatrati kao delegacioni protokol, koji omogućuje vlasniku resursa da dozvoli nekom softveru pristup tom istom resursu u njegovo ime, ali bez imitiranja samog vlasnika resursa. Dio privilegija korisnika se delegira datom softveru, ali sam OAuth protokol ne mari i ne sprovodi čin kontrole pristupa, koju obično implementira server sa resursima.
OAuth 2.0, iako često spominje autentifikaciju korisnika, nije autentifikacioni protokol i autentifikacija nije obrađena u okviru OAuth protokola. Ne govori ništa o tome ko je korisnik, niti kako da se potvrdi korisnički identitet. Međutim, omogućuje nadogradnju, što su neki protokoli kao OpenID Connect, HEART i iGov iskoristili. [24]

4.3.1 Terminologija i uloge

OAuth 2.0 definiše 4 uloge bitne za proces autorizacije:

1. Server sa resursima – Server koji nudi API dostupan preko HTTP-a za pristup čuvanim resursima preko tokena, bilo da je resurs dokument, pristup bazi podataka ili izvršavanje metode bez povratne vrijednosti.
2. Vlasnik resursa – Korisnik ili drugi entitet kom pripada vlasništvo nad zaštićenim resursom na serveru sa resursima.
3. Klijent – aplikacija koja treba pristup resursu na serveru sa resursima u ime korisnika ili u sopstveno ime. Prema OAuth 2.0 specifikaciji, može da bude povjerljivi klijent koji se izvršava u sigurnom okruženju i koji može da se autentifikuje autorizacionom serveru sa kredencijalima koje sigurno skladišti, ili javni klijent koji se izvršava ili na

[16] U radu se posebna pažnja obraća na verziju 2.0 OAuth protokola, ukoliko nije drugačije naglašeno upotrebom ili izostavljanjem specifične verzije.

uređaju nekog korisnika ili u Internet pretraživaču i ne može sigurno da skladišti tajne niti da se autentifikuje autorizacionom serveru. Postoje tri vrste profila klijenta: Internet aplikacija koja se izvršava na sigurnom back-end serveru i može da čuva kredencijale od interesa, zatim aplikacija bazirana na Internet pretraživaču kao što je jednostavna JavaScript stranica, ili nativna aplikacija koja jeste javni klijent, instalirana na korisničkom računaru ili mobilnom uređaju.

4. Autorizacioni server – HTTP server[17] koji autorizuje aplikacije za pristup resursnom serveru i kom resursni server vjeruje. Autentifikuje vlasnika resursa i klijenta i pruža mehanizam koji dozvoljava klijentu pristup resursima na resursnom serveru. [22]

Pored navedenih uloga, u OAuth 2.0 protokolu se pominju termini kao što su pristupni token (*engl. access token*), opseg, token za obnavljanje i autorizaciona dozvola (*engl. authorization grant*).

1. Token za pristup jeste artefakt koji autorizacioni server izdaje klijentu da bi prenio prava pristupa koja je vlasnik resursa odlučio da delegira. Ne mora da ima strukturu, iako se danas često koristi strukturisani token, kao što je JWT. Server sa resursima zna kako da provjeri validnost tokena. Ovakav pristup omogućava klijentu jednostavnu implementaciju protokola.
2. Opseg je reprezentacija skupa privilegija koje token ima nad zaštićenim resursom. Predstavljaju se listom stringova odvojenih praznim karakterom, a sam format opsega nije definisan protokolom i njegovo značenje određuje zaštićeni resurs, odnosno, resursni server. Primjeri opsega u prethodnom primjeru sa štampanjem slika bi bili sljedeći: *read-photo, read-metadata, upload-photo* opsezi.
3. Token za osvježavanje je koncept sličan tokenu za pristup, jer klijent ne mari za sadržaj tokena. Razlika je u tome da se nikada ne šalje zaštićenom resursu, nego se zbog kratkog trajanja pristupnog tokena, koristi da se izda novi pristupni token bez učešća korisnika. U verziji 1.0 nisu postojali, pa je trajanje pristupnog tokena bilo često podešeno na zabrinjavajuće dug period koji povećava površinu napada. Sada je token za osvježavanje taj koji ima duži životni vijek, ali bez prava pristupa resursnom serveru.
4. Autorizaciona dozvola jeste dozvola koju OAuth daje klijentu kako bi pristupio zaštićenom resursu koristeći OAuth protokol i postoji više vrsta dozvola. [24]

4.3.2 Vrste autorizacione dozvole

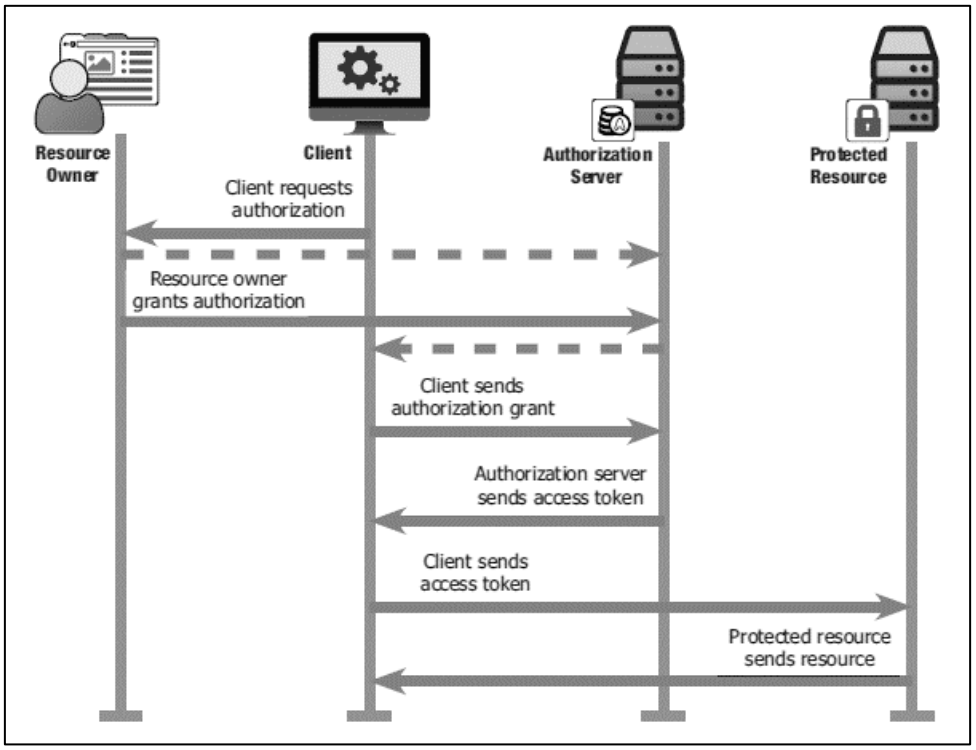
Postoji nekoliko načina da se implementira autorizaciona dozvola definisanih OAuth 2.0 protokolom. Najpouzdanija i sigurnija jeste Autorizaciona dozvola sa autorizacionim kodom i biće najdetaljnije opisana.

4.3.2.1 Autorizaciona dozvola sa autorizacionim kodom

Koristi dva zahtjeva od klijenta ka autorizacionom serveru kako bi se dobio pristupni token. U prvom zahtjevu, Internet pretraživač se preusmjerava na autorizacioni endpoint autorizacionog servera sa HTTP POST zahtjevom da se autorizuje klijent kako bi pristupio zaštićenom resursu vlasnika resursa. Da bi se smanjio rizik od CSRF napada, svaki poziv sadrži parametar zvani *state*, kojim klijent prati odnos između zahtjeva i odgovora. Trebao bi da sadrži vrijednost koja povezuje zahtjev sa korisničkom sesijom. Kada primi odgovor, klijent može da provjeri da li je vrijednost parametra *state* ista kao prilikom slanja zahtjeva (isti Internet pretraživač). Uz pomoć redirekcije, autorizacionom serveru se daje kontrola da komunicira sa korisnikom i da dobije dozvolu za izvršavanje autorizacije. Ukoliko korisnik ne odobri radnju, proces se završava. Kada dobije dozvolu korisnika, autorizacioni server preusmjerava

[17] OAuth 2.0 je predviđen da radi preko HTTP protokola, ali je moguće mapiranje na druge protokole. [24]

pretraživač uz pomoć *redirect_uri* parametra nazad na klijenta sa autorizacionim kodom – *code* parametar (i opciono *state* parametrom) u odgovoru.
Klijent (klijentska aplikacija) koristi taj autorizacioni kod pri slanju drugog HTTP POST zahtjeva na endpoint autorizacionog servera za dobijanje pristupnog tokena i opciono tokena za obnovu. Autorizacioni server će u slučaju da je zahtjev validan, poslati HTTP odgovor sa statusnim kodom 200 i JSON objektom koji sadrži pristupni token, tip tokena (najčešće Bearer) i opciono još dva elementa, a to su *expires_in* (koji govori o trajanju pristupnog tokena u sekundama) i *refresh_token* (token za obnovu). Slanje ovog zahtjeva se izvršava u pozadini i korisnik nije uključen u proces (*engl. backchannel request*). Slika 4.2 prikazuje objašnjenu komunikaciju. Za javne klijente se preporučuje korištenje dodatne zaštite uz pomoć PKCE. [22]



Slika 4.2 - Autorizaciona dozvola sa kodom [24]

PKCE jeste mehanizam kojim se garantuje da je klijent koji koristi autorizacioni kod za dobijanje pristupnog tokena isti klijent koji je dobio autorizacioni kod. Navedeno se postiže na sljedeći način: klijent kreira slučajni string koji se zove verifikator koda i koji je dovoljno dugačak da se teško pogodi. Klijent koristeći posebnu metodu transformiše ovaj verifikator koda u vrijednost *code_challenge*. Prilikom slanja prvog zahtjeva autorizacionom serveru, klijent šalje i metodu koju je koristio da kreira *code_challenge*. Prilikom slanja drugog zahtjeva, klijent šalje verifikator koda, a autorizacioni server na osnovu informacija iz prethodnog zahtjeva provjerava da li je sačuvani *code_challenge* jednak onom koji se dobije iz verifikatora koda iz zahtjeva za token. PKCE specifikacija navodi dvije metode za transformaciju: "plain" i "S256", pri čemu je kod prve izlaz jednak ulazu, a kod druge se izlaz dobija uz pomoć SHA256 heširanja. Druga metoda je sigurnija.
OAuth 2.0 definiše parametre zahtjeva i odgovora, međutim, opisivanje i pominjanje svih bi prelazilo okvire rada. Tabela 4.1 sadrži neke od čestih parametara (koji nisu već pomenuti). [22]

Tabela 4.1 - Parametri HTTP zahtjeva i odgovora definisani OAuth 2.0 protokolom

Naziv parametra	Značenje
client_id	Identifikator za klijenta registrovanog kod autorizacionog servera
grant_type	Naziv autorizacione dozvole koja se koristi: authorization_code, password, client_credentials
access_token	Pristupni token
token_type	Tip pristupnog tokena
resource	Identifikator za API zaštićenog resursa registrovan na autorizacionom serveru
code_challenge	PKCE kod za provjeru
code_challenge_method	Metoda koja se koristi za transformaciju verifikatora koda u PKCE kod za provjeru
code_verifier	Verifikator koda

4.3.2.2 Podrazumijevana autorizaciona dozvola

Upotrebom ovog tipa autorizacione dozvole, klijent dobija pristupni token u jednom zahtjevu. Budući da postoje klijenti kao JavaScript aplikacije koje se izvršavaju u potpunosti u okruženju Internet pretraživača, između njih više nema tajni. Samim tim, nema svrhe u slanju autorizacionog koda. Dakle, ovaj tip OAuth dozvole koristi samo prednji kanal (*engl. front channel*) za komunikaciju sa autorizacionim serverom. Pristupni token se vraća kao heš fragment URL-a za preusmjeravanje kao što je prikazano u fragmentu koda ispod teksta.

```
GET /callback#access_token=987tghjkiu6trfghjuytrghj&token_type=Bearer
```

Ovdje se takođe ne koristi token za osvježavanje, zbog samog domena primjene klijenta koji živi dugo koliko sesija na Internet pretraživaču. Zbog sigurnosnih problema, kada god je moguće, poželjno je izbjeći ovaj tip dozvole. [24]

4.3.2.3 "Resource Owner Password Credentials" autorizaciona dozvola

Ovaj tip dozvole podržava situacije kada je klijentu dozvoljeno da traži kredencijale od vlasnika resursa i te kredencijale šalje autorizacionom serveru preko pozadinske komunikacije kako bi dobio pristupni token. Već je spomenuto ranije da je to loša praksa, ali je opisano u OAuth standardu i koristi se samo ako ne postoji drugi način. Mimo svih pomenutih ranjivosti ovakvog pristupa, uvijek je lakše upravljati kratkoživućim tokenom nego korisničkim kredencijalima. Što se samog slanja zahtjeva tiče, šalju se i korisnički kredencijali i šalju se identifikator aplikacije i dodijeljena mu tajna (*client_id* i *client_secret*) kao HTTP Basic autentifikacija. Najčešće se koristi tokom migracije podataka o korisnicima sa jednog na drugi repozitorijum, pri čemu se strogo savjetuje brisanje kredencijala sa klijenta nakon završetka migracije. Ovaj tip dozvole je baziran na anti-obrascu "traži ključeve". [22] [24]

4.3.2.4 "Client credentials" autorizaciona dozvola

Ovaj tip autorizaciona dozvole se koristi kada klijent poziva API za pristup resursima koje posjeduje. U jednom zahtjevu se dobavlja pristupni token, jer nema interakcije sa korisnikom. Ovdje se šalju klijentski identifikator i klijentska tajna jednako kao u prethodnom slučaju. [22]

4.3.3 Tokeni

OAuth protokol se bazira na tokenima. Tokeni su srž svih važnih transakcija. Klijenti dobavljaju tokene sa autorizacionog servera kako bi pristupili zaštićenom resursu na resursnom serveru, koji prima date tokene i provjerava ih pozivom prema autorizacionom serveru, pozivom direktno u bazu ili služeći se strukturom tokena, ako je takav u upotrebi. Da bi klijent dobio novi token za istu sesiju bez da ometa korisnika, on šalje pozadinski zahtjev ka autorizacionom serveru i tom prilikom koristi drugi token, a to je token za osvježavanje. Sam OAuth protokol ne definiše šta čini sadržaj tokena, kao ni format, što je već rečeno. Klijentu je dakle nepoznat sadržaj tokena, čime se postiže daleko šira upotreba nego kod protokola koji definišu striktan format tokena, kao što su SAML ili Kerberos. [24]

4.3.3.1 Pristupni tokeni

Pristupni tokeni su kredencijali koji se koriste da se pristupi zaštićenom resursu u ime korisnika. Predstavlja sloj apstrakcije mijenjajući na taj način upotrebu uobičajenih kredencijala (korisničkog imena i lozinke). Obično je u formi stringa čiji sadržaj i forma klijentu koji ga dobija nisu važni. Predstavlja specifične opsege i trajanje pristupa koje je vlasnik resursa dozvolio. Može takođe da uključuje identifikator za dobijanje zaštićenih informacija ili da bude samodovoljan (sadržavajući podatke i potpis). Može da ima različite formate, strukturu i metode upotrebe zavisno od sigurnosnih mjera resursnog servera.
Klijent ne treba da koristi token ukoliko mu nije poznat tip tokena. OAuth 2.0 specifikacija definiše dva tipa tokena: Bearer i MAC token. [27]

4.3.3.2 Bearer i MAC token

Svaka stranka u posjedu Bearer tokena, odnosno stranka koja jeste nosilac *(engl. Bearer)* tokena, može da koristi dati token za pristup povezanom resursu bez pokazivanja kriptografskog ključa. Obavezno je korištenje TLS-a.
Iako token može da se šalje kao dio forme ili URL upita[18], najčešće se šalje kao dio *Autorization* zaglavlja sa ključnom riječi "Bearer"[19], kao u primjeru koda ispod: [28]

```
Authorization: Bearer <token>
```

MAC token je kombinacija pristupnog tokena i MAC (*engl. Message Authentication Code*) ključa koji se koristi za potpisivanje određenih komponenti HTTP zahtjeva. Primjer zahtjeva koji uključuje korištenje Authorization zaglavlja i ključne riječi MAC je dat u isječku koda ispod:

[18] Ova dva načina slanja imaju svoje sigurnosne nedostatke, dok upotreba HTTP zaglavlja nudi veću fleksibilnost. Svejedno, veće biblioteke za rad sa ovim tokenima podržavaju sve tri metode slanja.
[19] Nije važno da li se koriste velika ili mala slova.

```
GET <resource> HTTP/1.1
HOST: <host>
Authorization: MAC id=”h480djs93hd8”,
                   nonce=”274312:dj83hs9s”,
                   mac=”kDZvddkndxvhGRXZhvuDjEWhGae=”
```

MAC šema zahtijeva razmjenu dijeljenog ključa između servera i klijenta, što se obično obavlja prilikom klijentske registracije. Koristi se uglavnom kada servisi ne mogu da koriste TLS. [29]

4.3.3.3 JWT token

Da bi se provjerila validnost tokena, potrebno je da resursni server kontaktira ili dijeljenu bazu ili autorizacioni server. Kako bi se spriječio takav vid komunikacije (dodatni API poziv, dijeljena baza), kreirani su strukturisani tokeni koji nose dovoljno informacija da bi se provjere izvršile sa minimalnim brojem poziva ka serveru.
JWT tokeni su tokeni u formi JSON objekta u određenom formatu: *zaglavlje.podaci.potpis*. Takav format omogućuje jednostavno prenošenje informacija koje treba da se pošalju sa tokenom. Ukoliko se ne koristi potpisivanje, format izgleda kao u primjeru ispod:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikp
vaG4gRG9lIiwiYWRtaW4iOnRydWV9.
```

Sektori tokena su odvojeni tačkom. Svaka vrijednost između tačaka jeste Base64 enkodovan JSON objekat. Budući da se token nalazi u HTTP zaglavlju, dijelu HTTP upita, forme ili tijela, upotreba Base64 enkodovanja je bila logičan izbor, kako bi se spriječilo komplikovano maskiranje karaktera, izbjegle potencijalne slučajne greške i postigla uniformnost.
Ukoliko se dekoduje prvi dio *eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0* i isparsira, dobija se sljedeći JSON objekat koji opisuje ostatak tokena:

```
{
  "typ": "JWT",
  "alg": "none"
}
```

Tip tokena, odnosno *typ*, govori o kom tokenu je riječ, jer druge vrste tokena mogu da koriste sličnu strukturu kao JWT. Algoritam, odnosno *alg*, indicira algoritam korišten za potpisivanje tokena. Drugi dio tokena se obrađuje na isti način kao i prvi, i dobija se sljedeći JSON objekat koji predstavlja podatke o vlasniku resursa:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Riječ je o proizvoljnom odabiru podataka (dok god je riječ o validnim JSON podacima), ali su obično koriste informacije iz skupa JWT tvrdnji. Tabela 4.2 navodi neke od njih.

Tabela 4.2 - Neke od JWT tvrdnji

Naziv tvrdnje	Opis tvrdnje
iss	Izdavač tokena
exp	Vrijeme isteka tokena u sekundama od UNIX epohe.
iat	Vrijeme izdavanja tokena u sekundama od UNIX epohe.
jti	Jedinstveni identifikator tokena.
aud	Indikator strana kojima je token namijenjen.
sub	Subjekt tokena, najčešće indikator vlasnika resursa.

Budući da klijentu sadržaj tokena nije važan, klijent može da koristi token dok god ne prestane da radi, ili autorizacioni server može da pošalje dodatnu vrijednost *expires_in* kako bi naznačio trajanje tokena.
Ovakvo prenošenje JWT tokena nije sigurno, jer je izmjena nepotpisanog tokena jednostavna i bilo ko može da manipuliše ovakvim tokenom. JOSE (*engl. JSON Object Signing and Encryption*) skup specifikacija o načinima zaštite JWT tokena. Token može samo da se potpiše ili da se u potpunosti enkriptuje. Primjer potpisivanja će biti dat upotrebom jednostavnog HMAC sa SHA-256 algoritma.[20] Zaglavlje je dato u primjeru ispod:

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

HS256 koristi jedan dijeljeni ključ (simetrično heširanje) koji treba da je dostupan i autorizacionom serveru koji vrši potpisivanje i resursnom serveru koji vrši validaciju potpisa. Potpis se kreira upotrebom pomenutog algoritma nad enkodovanim zaglavljem, podacima i dijeljenim ključem, kao u primjeru ispod:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Na taj način se verifikuje da se poruka (token) nije izmijenila tokom slanja, a ukoliko se koristi asimetrični algoritam sa privatnim ključem, moguće je verifikovati i da je token kreirao stvarni autorizaciona server.
Upotreba JWT tokena u odnosu na ostale vrste tokena jeste u lakoći prenosa i jednostavnosti parsiranja u odnosu na XML bazirane tokene ili tokene koji mogu da se potpišu samo upotrebom simetričnog algoritma kao što je slučaj sa SWT tokenima. [24] [27]

[20] Upotreba asimetričnih algoritama, kao što je RSA, je sigurnija. Potpisivanje se obavlja sa privatnim ključem dostupnim samo autorizacionom serveru, dok se javni ključ nalazi i na resursnom i na autorizacionom serveru. Kako bi se spriječilo kopiranje ključeva, autorizacioni server može da objavi svoje javne ključeve, što je praksa kod OpenID Connect protokola. [24]

4.3.3.4 Osvježavanje tokena

OAuth 2.0 pruža mehanizam dobavljanja novog pristupnog tokena bez interakcije vlasnika resursa uz pomoć tokena za osvježavanje. Zahvaljujući ovom mehanizmu, sam OAuth 2.0 protokol je sigurniji za korištenje od prethodne verzije, jer se može skratiti vrijeme trajanja pristupnog tokena. Token za osvježavanje može da traje i nekoliko dana, i kao i pristupni token, treba sigurno da se skladišti. Da bi se dobio novi pristupni token, klijent šalje pozadinski zahtjev za dobijanje novog pristupnog tokena zajedno sa tokenom za osvježavanje.
Kada klijent ima token za osvježavanje, da bi dobio novi pristupni token, šalje HTTP POST zahtjev sa sljedećim parametrima:

```
grant_type=refresh_token
refresh_token=<vrijednost tokena za osvježavanje>
scope=<opciono>
```

Ukoliko je token za osvježavanje ispravan, kreira se novi pristupni token i vraća klijentu. [22] Da bi se povećala sigurnost upotrebe tokena za osvježavanje, preporučuje se korištenje rotacije tokena za osvježavanje. Ovaj mehanizam rotacije spriječava korištenje tokena za osvježavanje u slučaju "čovjek u sredini" napada. [27] Najbolje vrijeme za osvježavanje tokena jeste onda kada je pristupni token istekao, a javila se potreba za njegovom upotrebom. OAuth 2.0 specifikacija ne definiše kada se kreira token za osvježavanje, tako da to zavisi od implementacije autorizacionog servera koji može da vrati token za osvježavanje prilikom prvog vraćanja pristupnog tokena, ili može da zahtijeva slanje zasebnog zahtjeva.
Statički tokeni za osvježavanje se ne preporučuju kod javnih klijenata, jer ih takvi klijenti ne mogu sigurno skladištiti. Ova funkcionalnost je važna za mnoge scenarije, ali ne za sve. Recimo, u slučaju "*Client Credentials*" autorizacije nije potrebno da se koristi token za osvježavanje, jer aplikacija programski može da zahtijeva novi pristupni token. [22]

4.3.3.5 Povlačenje pristupnog tokena

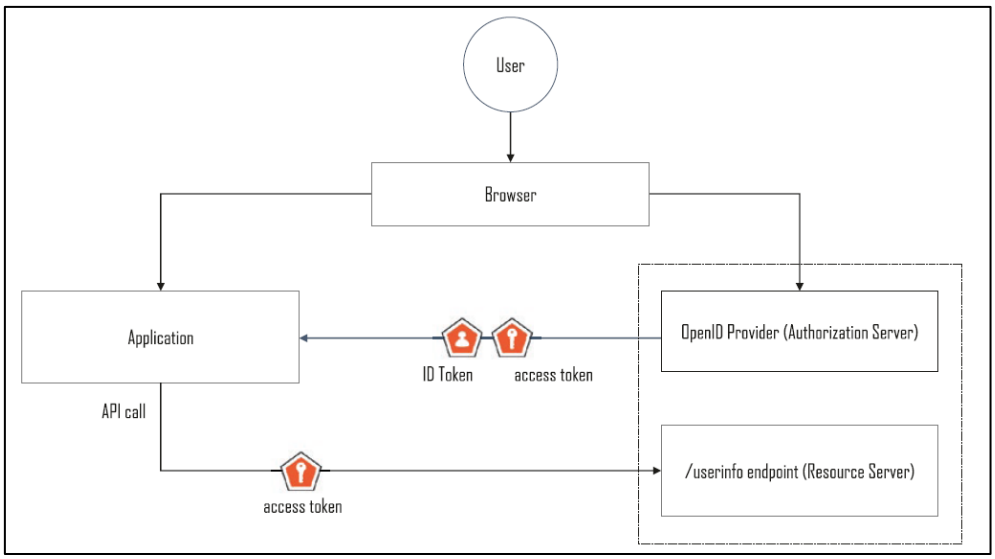
Token ima svoj vijek trajanja, međutim, moguće je povući token iz upotrebe i proglasiti ga nevalidnim što je često kod odjave sa sistema. OAuth definiše specifikaciju za povlačenje. Klijent šalje HTTP POST zahtjev na endpoint za povlačenje tokena, pri čemu je token koji se povlači enkodovan u formi POST zahtjeva. Ukoliko autorizacioni server pronađe token, briše ga iz baze ili označava povučenim i vraća OK. Ukoliko ne pronađe token ili klijent nije autorizovan za povlačenje, ne radi ništa i vraća statusni kod OK, kako bi izbjegao davanje informacija o tokenima. Klijent odbacuje token na svojoj strani i to je to. Ukoliko se vrši povlačenje tokena za osvježavanje, potrebno je povući i vezani pristupni token. [24]

4.4 OpenID Connect (OIDC) autentifikacija i autorizacija

U prethodnom poglavlju je rečeno da OAuth 2.0 protokol nije autentifikacioni protokol, ali nudi mogućnosti nadogradnje. OpenID Connect jeste interoperabilni autentifikacioni protokol koji je baziran na OAuth 2.0 razvojnom okviru. Pojednostavljuje način verifikacije korisničkog identiteta uz pomoć autorizacionog servera u duhu REST stila. [30]
Kada korisnik pristupi aplikaciji, vrši se preusmjeravanje Internet pretraživača na autorizacioni server (u kontekstu OIDC specifikacije zvan OpenID provajder) koji implementira OIDC protokol. OpenID provajder vrši interakciju sa korisnikom u cilju autentifikacije. Nakon autentifikacije, Internet pretraživač se usmjerava natrag na aplikaciju. Sada sama aplikacija može da traži tvrdnje o korisniku u formatu sigurnosnog tokena zvanog ID Token ili može da zahtijeva OAuth 2.0 pristupni token koji može da iskoristi za dobavljanje tvrdnji o korisniku. OIDC definiše sljedeće uloge:

1. Krajnji korisnik – subjekat koji se autentifikuje.[21]
2. OpenID provajder (OP) ili provajder identiteta (IDP) – OAuth 2.0 autorizacioni server koji implementira OIDC protokol (samim tim i OAuth 2.0) i može da autentifikuje korisnika i da vrati informacije o događaju autentifikacije ili tvrdnje o korisniku aplikaciji.
3. Pouzdana strana (RP) – OAuth 2.0 klijent koji delegira autentifikaciju korisnika OP i zahtijeva tvrdnje o korisniku.
4. ID Token – predstavlja izlaz procesa autentifikacije i sadrži najmanje identifikator korisnika i informaciju kako i kada je korisnik autentifikovan. Koristi JWT format, sa tvrdnjama koje su ranije definisane. Pored tog, može da sadrži dodatne tvrdnje kao što su: *given_name*, *email*, *email_verified*, *picture* i druge OP definisane tvrdnje. Zavisno koji tok OIDC se koristi, zavise i tvrdnje koje se koriste.

OIDC definiše javne i povjerljive klijentske aplikacije, kao i aplikacije instalirane na korisničkom uređaju. Slika 4.3 prikazuje komunikaciju u OIDC protokolu.



Slika 4.3 - Komunikacija kod OpenID Connect protokola [22]

OIDC definiše iste endpoint-e kao i OAuth 2.0, sa dodatkom endpoint-a za dobavljanje korisničkih tvrdnji (osiguran pristupnim tokenom). OIDC definiše tri toka: autorizaciona dozvola sa autorizacionim kodom, podrazumijevana dozvola i hibridna dozvola. [22]

[21] U radu se umjesto termina krajnji korisnik koristi termin korisnik, a umjesto pouzdana strana aplikacija.

4.4.1 Autorizaciona dozvola sa autorizacionim kodom

Slično kao kod OAuth 2.0, postoje dva endpoint-a i posrednički autorizacioni kod. Klijentska aplikacija preusmjerava Internet pretraživač na OP endpoint za dobijanje autorizacionog koda. OP autentifikuje korisnika i vraća Internet pretraživač na klijentsku aplikaciju. Dalje, klijent šalje novi zahtjev sa tim kodom na drugi endpoint u cilju dobijanja ID tokena, pristupnog tokena i opciono, tokena za osvježavanje. Sigurniji pristup bi bio korištenje PKCE za javne klijente, što je već objašnjeno u opisu OAuth 2.0 protokola. Primjer zahtjeva:

```
GET /authorize?
response_type=code
& client_id=<klijentski identifikator>
& state=<vrijednost stanja>
& nonce=<nonce vrijednost>
& scope=<opseg>
& redirect_uri=<povratni URI>
& code_challenge=<code challenge>
& code_challenge_method=<metoda code challenge> HTTP/1.1
HOST: oprovajder.com
```

Većina parametara zahtjeva je poznata iz OAuth 2.0 protokola. Tip OIDC toka koji se koristi se specifikuje sa *response_type* parametrom i ovaj parametar je obavezan. Sa druge strane, parametar *response_mode* je opcion i služi za zahtijevanje načina slanja odgovora klijentskoj aplikaciji. Podrazumijevana vrijednost jeste vraćanje tokena kao parametara upita ili heširanog fragmenta povratnog URI-ja. Još jedan opcioni parametar jeste *nonce*, koji treba da se koristi kada se zahtijeva ID token. Kada aplikacija pošalje zahtjev, treba da specifikuje *nonce* kao jedinstvenu, nemoguću za pogoditi vrijednost koja je vezana za sesiju korisnika – generiše se slučajna vrijednost i čuva u sesiji korisnika. Kada aplikacija primi ID token, odgovor treba da sadrži neizmjenjenu nonce vrijednost. Da bi se specifikovala svrha korištenja OIDC i tražile određene tvrdnje o korisniku, koristi se parametar *scope*, na primjer openid%profil%email. Obavezno sadrži "openid" vrijednost.
OpenID provajder vraća odgovor sa autorizacionim kodom koji se može iskoristiti samo jednom i neizmjenjenim parametrom stanja. Prilikom slanja zahtjeva za token, šalje se i klijentska tajna (HTTP Basic autentifikacijom). Takođe se šalje i verifikator koda. OP vraća rezultat u JSON formatu, kao u primjeru ispod:

```
{
  "id_token": <ID token>,
  "access_token": <pristupni token>,
  "refresh_token": <token za osvježavanje>,
  "token_type": "Bearer"
  "expires_in": <trajanje tokena>
}
```

Prije nego se pouzda u tvrdnje u ID tokenu, klijentska aplikacija treba da provjeri token prema OP vodiču i validacionim koracima JWT specifikacije. Aplikacija može da dobavi detalje i tvrdnje o korisniku ili iz ID tokena ili pozivajući odgovarajući OP endpoint sa pristupnim tokenom. [22]

4.4.2 Implicitni tok

Implicitni tok OIDC protokola je sličan istoimenom toku OAuth 2.0 protokola. Za aplikacije kojima je potrebna samo autentifikacija i ID token, a ne pristupni token, ovaj tok može biti

odgovarajući. Korisnik pristupi aplikaciji i biva preusmjeren ka OP sa autentifikacionim zahtjevom. OP komunicira sa korisnikom radi autentifikacije i pružanja tvrdnji o korisniku datom klijentu. Nakon što se korisnik prijavi na sistem i pruži odobrenje za pristup informacijama, OP kreira ili ažurira sesiju za korisnika. Internet pretraživač korisnika se sada preusmjerava, sa ID tokenom u odgovoru, nazad ka klijentskoj aplikaciji. Dozvoljene vrijednosti za *response_type* parametar su "*id_token*" i "*id_token token*" (odgovor sadrži i ID token i pristupni token, što nije preporučljivo). [22]

4.4.3 Hibridni tok

Predstavlja kombinaciju implicitnog i toka sa autorizacionim kodom. Dizajniran je za aplikacije koje imaju siguran i front-end i back-end i JavaScript kod koji se izvršava u Internet pretraživaču. Dobavljanje ID tokena i autorizacionog koda se vrši pomoću implicitnog toka, dok se dobavljanje pristupnog tokena (i opciono tokena za osvježavanje) vrši uz pomoć toka sa autorizacionim kodom i pozadinskog kanala. Dakle, nakon što klijentska aplikacija dobije ID token i autorizacioni kod, provjerava ID token i ako je validan, šalje pozadinski zahtjev OP zarad dobijanja ostalih tokena. Parametar response_type u ovom slučaju može imati sljedeće vrijednosti: "*code id_token*", "*code token*" i "*code id_token token*", a posljednja dva nisu preporučljiva zbog vraćanja pristupnog tokena kao dio URL-a. Ovaj tok nije toliko korišten zbog komplikovane implementacije. [22]

4.4.4 Endpoint za dobavljanje tvrdnji o korisniku

Neka se ovaj endpoint zove *KorisnikInfo* endpoint. Aplikacija može da kontaktira ovaj endpoint OpenID provajdera da bi dobila tvrdnje o korisniku uz pomoć tokena. Prilikom slanja zahtjeva za dobijanje pristupnog tokena, aplikacija je poslala *scope* parametar da utvrdi željene tvrdnje o korisniku. OP autentifikuje korisnika i dobija njegov pristanak za tražene tvrdnje, i izdaje pristupni token sa autorizovanim opsegom za date tvrdnje aplikaciji. Aplikacija koristi pristupni token da dobije tvrdnje od *KorisnikInfo* endpoint-a. Odgovor koji aplikacija dobija pozivajući ovaj endpoint je u JSON formatu i sadrži informacije o korisniku. Ovaj endpoint je naročito koristan kada korisničkih tvrdnji ima mnogo da bi sve bile sadržane u ID tokenu. [22]

4.5 SAML 2.0

SAML 2.0 (*engl. Security Assertion Markup Language*) je standard koji se koristi za rješavanje problema SSO sa unakrsnim domenima, nudi federativni identitet (*engl. identity federation*) i standardizovan način dijeljenja informacija. Iako su OAuth 2.0 i OpenID Connect noviji protokoli, u praksi su česti poslovni sistemi koji upotrebljavaju SAML. SAML je kreiran kao XML-baziran okvir za opisivanje i razmjenu sigurnih informacija između on-line poslovnih partnera. Omogućava aplikacijama da delegiraju autentifikaciju korisnika eksternom entitetu zvanom provajder identiteta i nudi SSO.
Takođe, uz pomoć SAML, aplikacija i provajder identiteta mogu da koriste mehanizam kojim koriste dijeljeni identifikator korisnika u svrhu razmjene informacija, što je poznao kao federativni identitet. [22] [31]

4.5.1 Terminologija

SAML specifikacija definiše sljedeće uloge:

- Subjekat – Entitet koji je vlasnik informacija koje će da se razmjenjuju. Može biti ili osoba ili drugi sistem.

- SAML tvrdnja (*engl. SAML Assertion*) – Poruka u XML formatu koja sadrži povjerljive informacije o subjektu.
- SAML profil – Specifikacija koja definiše kako se SAML poruke koriste u poslovnom sistemu.
- Provajder identiteta (IP) – Uloga definisana za SAML profil. To je server koji izdaje SAML tvrdnje o subjektu u kontekstu SSO kod unakrsnih domena.
- Provajder servisa – Uloga definisana za SAML profil. Provajder servisa delegira autentifikaciju provajderu identiteta i vjeruje SAML tvrdnji koju dati provajder vrati.
- Veza povjerenja – Dogovor između SAML provajdera servisa i provajdera identiteta prema kom provajder servisa vjeruje SAML tvrdnjama koje provajder identiteta vraća.
- SAML vezivanje protokola – Opis kako se SAML poruke mapiraju na standardne komunikacione protokole kao što je HTTP. [22] [31]

4.5.2 Opis funkcionisanja

Subjekat, u ovom slučaju korisnik, želi da koristi neku Internet aplikaciju. U ovom slučaju, ta aplikacija predstavlja provajder servisa. Samim tim, data aplikacija delegira proces autentifikacije povezanom SAML IP, koji se nalazi na drugom domenu. IP autentifikuje korisnika zahtijevajući od njega prijavu sa kredencijalima i vraća sigurnosni token poznat kao SAML tvrdnja u XML formatu. Ta tvrdnja pruža informacije o korisniku i o autentifikacionom događaju. Preduslov za ovakvu komunikaciju jeste povezivanje provajdera identiteta i provajdera servisa pri čemu moraju da razmijene meta podatke kao što su URL, sertifikati i slično, kako bi se uspostavila veza povjerenja.
SSO najčešće inicira provajder servisa. Korisnik posjeti aplikaciju, koja ga preusmjeri na provajder identiteta sa SAML autentifikacionim zahtjevom. IP komunicira sa korisnikom da bi ga autentifikovao. Nakon što korisnik unese kredencijale i IP ih provjeri, šalje Internet pretraživač korisnika nazad ka aplikaciji sa SAML odgovorom koji sadrži SAML tvrdnju. Sam odgovor se šalje na URL provajdera servisa koji je ranije registrovan. Aplikacija može da obradi originalni korisnički zahtjev nakon što obradi SAML odgovor i provjeri privilegije. U slučaju da korisnik sada pristupi drugoj aplikaciji koja koristi isti provajder identiteta, ta aplikacija vrši preusmjeravanje korisničkog Internet pretraživača ka datom IP. IP prepoznaje da korisnik već ima aktivnu sesiju i samo preusmjeri Internet pretraživač nazad ka aplikaciji sa SAML tvrdnjom. Dakle, preskočeni su koraci interakcije sa korisnikom. U slučaju da druga aplikacija zahtijeva restriktivnije privilegije ili ponovnu prijavu, interakcija sa korisnikom će biti izvršena.
SAML definiše i SSO koji inicira provajder identiteta. Često ga koriste organizacije sa poslovnim aplikacijama kojima se pristupa preko korporativnog portala. Kada korisnik pristupi portalu, autentifikuje se preko provajdera identiteta sa SAML autentifikacionim zahtjevom. Nakon toga, biva preusmjeren sa SAML odgovorom na portal koji sada nudi listu aplikacija kojima može da pristupi, zavisno od privilegija. Kada korisnik pokuša da pristupi nekoj od ponuđenih aplikacija, Internet pretraživač usmjerava korisnika ka provajderu identiteta sa URL-om željene aplikacije, a IP preusmjerava pretraživač ka registrovanom URL-u date aplikacije zajedno sa novim SAML odgovorom. [22]

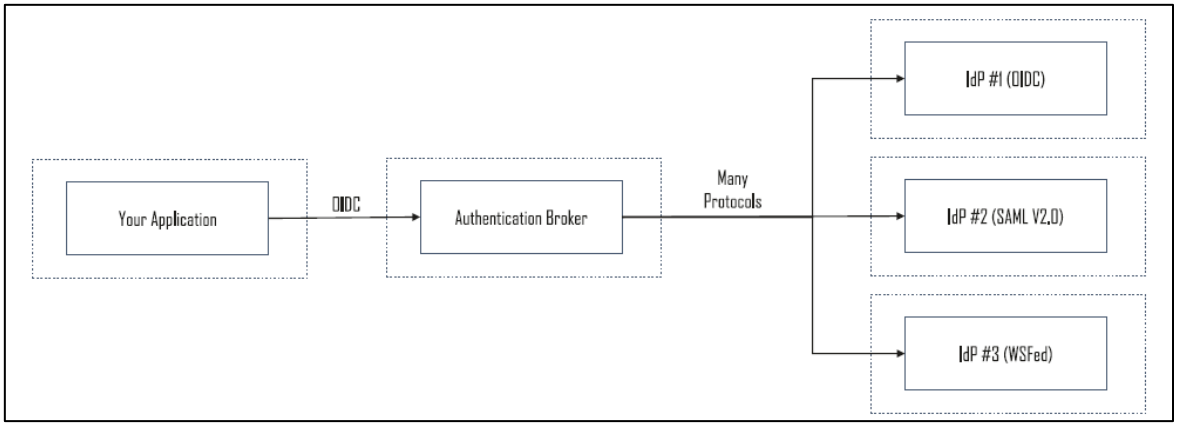
4.5.3 Federativni identitet

Provajder servisa i provajder identiteta uspostave federativni identitet tako što se dogovore oko jedinstvenog identifikatora korisnika, što omogućuje provajderu servisa da provajderu identiteta delegira autentifikaciju i dobije SAML odgovor sa tvrdnjama za korisnika kome pripada dati identifikator identiteta. Administratori provajdera identiteta ga konfigurišu tako da

svaki provajder servisa dobije tvrdnje sa identifikatorom identiteta koji odgovara datom provajderu servisa. Dakle, svaki provajder servisa će imati svoj link sa provajderom identiteta, a to je najčešće email korisnika. [31]

4.5.4 Autentifikacioni broker

Sistemi mogu da koriste autentifikacione brokere da bi podržali višestruke autentifikacione protokole i mehanizme. Ukoliko se projektuje aplikacija koja koristi OIDC kao protokol, organizacija može da zahtijeva da se podrži SAML za neke poslovne partnere. Kako je SAML kompleksan protokol, nije preporučljivo da se implementira od početka. Umjesto toga, autentifikacioni broker dozvoljava aplikaciji da implementira novi protokol kao što je OIDC, ali da se osloni na njega za komunikaciju preko drugih protokola raznih provajdera identiteta. Slika 4.4 prikazuje ulogu autentifikacionog brokera.



Slika 4.4 - Autentifikacioni broker kao posrednik u komunikaciji [22]

4.5.5 Konfiguracija i primjer SAML zahtjeva/odgovora

Da bi se SAML konfigurisao, obično treba da se konfigurišu sljedeće stvari:
SSO URL je URL provajdera identiteta. Sertifikat provajdera identiteta se koristi da bi se provjerili potpisi SAML odgovora. Takođe, koristi se ako je potrebno enkriptovati SAML zahtjeve. U tom slučaju se mogu koristiti različiti sertifikati. Potpisivanje/enkripcija zahtjeva treba da bude postavljeno ukoliko se planira digitalno potpisivanje/enkripcija zahtjeva. ASC URL jeste URL provajdera servisa na koji će se primiti SAML odgovori. Sertifikati provajdera servisa se koriste za validiranje potpisa SAML zahtjeva. Takođe, koriste se ako je potrebno enkriptovati SAML odgovore. Kao i u slučaju SAML zahtjeva, ukoliko se planira digitalno potpisivanje/enkripcija SAML odgovora, potrebno ga je podesiti.
SAML zahtjev može da značajno da varira, jer su mnogi elementi opcioni. Primjer jednog SAML zahtjeva koji ne koristi potpisivanje:

```
<samlp:AuthnRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
Destination="IDP URL"
ID="ID"
IssueInstant="TIME ISSUED"
ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
Version="2.0"
ProviderName="SERVICE PROVIDER"
AssertionConsumerServiceURL="ACS URL">
   <saml:Issuer>SERVICE PROVIDER</saml:Issuer>
</samlp:AuthnRequest>
```

SAML odgovor je XML poruka sa više komponenti, a sadržaj odgovora zavisi od prirode zahtjeva, konfiguracije IP i informacija koje su tražene. Primjer jednog SAML odgovora:

```
<samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
ID="ID"
InResponseTo="ID OF CORRESPONDING REQUEST"
Version="2.0"
IssueInstant="TIME ISSUED"
Destination="ACS URL of Service Provider">
   <saml:Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
ISSUER OF RESPONSE - IDENTITY PROVIDER
   </saml:Issuer>
   <samlp:Status>
      <samlp:StatusCode Value="SAML 2.0:Success"/>
   </samlp:Status>
</sampl:Response>
```

4.6 Single-Sign On (SSO)

SSO je, kao što je već rečeno, mogućnost korisnika da se jednom prijavi i da ostvari pravo pristupa na više sistema bez ponovne prijave, dok god se sesija ne okonča. Najčešće se koristi sa OIDC ili SAML 2.0 protokolom i implementira se na strani provajdera identiteta. Pojavljuje se u više scenarija, bilo da je riječ o komercijalnim aplikacijama koje nude prijavu preko Google Sign-In[22], bilo da je riječ o internim poslovnim sistemima koje koriste interni provajder identiteta. Pored delegacije prijave na sistem, aplikacije mogu da se oslone na provajder identiteta da implementira stranice za prijavu, validaciju kredencijala, skladištenje istih, kao i druge funkcionalnosti, kao što je oporavak naloga. Za poslovni sistem, SSO nudi jedno mjesto na kom se implementiraju polise autorizacije, oporavak naloga, odjava, dodavanje i brisanje naloga. O drugim prednostima je bilo riječi ranije. Međutim, SSO ima sigurnosni nedostatak koji se bazira upravo na tom centralizovanom pristupu. Naime, zahvaljujući SSO, postoji samo jedna tačka pada i jedna tačka za napad. Da bi se smanjio rizik, potrebno je implementirati visoku dostupnost i najbolje prakse za unapređenje sigurnosti sistema. Još jedan nedostatak je što provajder identiteta može da prati aktivnost korisnika na raznim stranicama i aplikacijama, što narušava pitanje privatnosti. Zbog toga aplikacija treba da bude oprezna pri izboru provajdera identiteta. Prateći kako rade prethodno pomenuti protokoli, SSO funkcioniše na sljedeći način: korisnik posjeti *aplikaciju 1* koja zahtijeva prijavu i biva preusmjeren na stranicu provajdera identiteta. Provajder identiteta autentifikuje korisnika kroz interakciju sa istim, kreira sesiju i kolačić za Internet pretraživač sa informacijama o toj sesiji. Korisnički pretraživač preusmjerava ka *aplikaciji 1* sa sigurnosnim tokenom, a ta aplikacija može da održava svoju lokalnu sesiju za datog korisnika. Ukoliko korisnik sa istim Internet pretraživačem posjeti *aplikaciju 2*, data aplikacija detektuje da se korisnik nije prijavio i preusmjerava ga ka provajderu identiteta. Provajder identiteta dobija zahtjev zajedno sa prethodno kreiranim kolačićem, i nakon što se uvjeri da postoji kreirana sesija, vraća pretraživač ka *aplikaciji 2* sa novim sigurnosnim tokenom. *Aplikacija 2* može da vodi lokalnu sesiju. Ukoliko je potreban striktniji pristup, provajder identiteta može da zahtijeva interakciju korisnika.

Prilikom konfiguracije SSO, bitno je voditi računa o atributima sesije, kao što je trajanje sesije, snaga autentifikacionih mehanizama, brendiranje stranice za prijavu i mehanizmi za terminiranje sesije.

[22] https://developers.google.com/identity/

4.6.1 Trajanje SSO sesije

Trajanje SSO sesije se specifikuje u terminima maksimalnog trajanja i isteka zbog neaktivnosti. Ako aplikacija koja koristi OIDC zahtijeva od korisnika više prijava nego što zahtjeva sesija provajdera identiteta, onda se parametrom "*max_age*" u autentifikacionom zahtjevu može specifikovati maksimalno dozvoljeno trajanje sesije u sekundama. Aplikacija može da omogući korisniku da ostane aktivan duže nego što traje sesija provajdera identiteta koristeći duže vrijeme isteka sesije.

4.6.2 Više provajdera identiteta

SSO može da koristi autentifikacioni broker za konfiguraciju više provajdera identiteta, a broker treba da se konfiguriše tako da osigura da se korisnik prijavljuje na aplikaciju korištenjem odgovarajućeg provajdera identiteta. Recimo da kompanija koristi jedan provajder identiteta za poslovne partnere, a drugi za zaposlene, broker treba da osigura da partneri ne mogu da pristupe aplikacijama koje su namijenjene samo zaposlenim.

4.6.3 Brendiranje stranice za prijavu

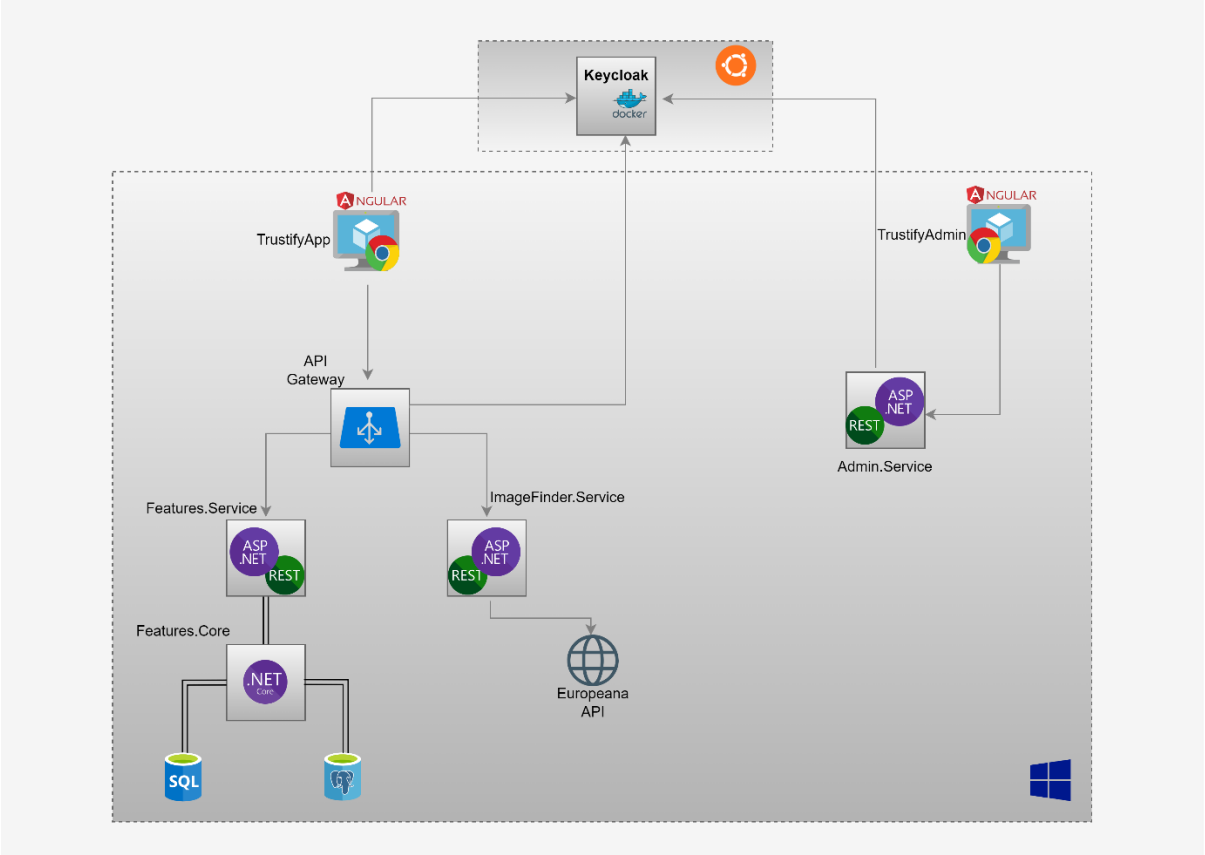
Vrlo je važno korisniku naglasiti gdje se to prijavljuje kada koristi SSO. Ako se prijavljuje na korporativni provajder identiteta, takva stranica za prijavu treba da sadrži brend date korporacije, u vidu dizajna ili logoa. Tako je korisnicima lakše da prate sesije otvorene kod više različitih provajdera identiteta i lakše im je da okončaju pravilnu sesiju. Takođe, ukoliko aplikacija vodi računa o lokalnoj sesiji, i korisnik se odjavi samo sa lokalne sesije, treba da bude svjestan tog da je sesija kod provajdera identiteta još uvijek otvorena. [22]

5 PRAKTIČNI DIO

U praktičnom dijelu rada je implementirano više aplikacija, kako bi demonstracija autentifikacije i autorizacije, kao i same arhitekture sistema, bila na zadovoljavajućem nivou. Kao što je pomenuto u prethodnim poglavljima, brojni su načini autentifikovanja i autorizovanja korisnika. U praktičnom dijelu je fokus stavljen na upotrebu kontrole pristupa zasnovane na ulogama. Što se samog sistema tiče, u pitanju su dvije Internet aplikacije, razvijene upotrebom .NET tehnologija na back-end strani i Angular razvojnog okvira na front-end strani. U ostatku rada biće opisane najbitnije stavke iz praktičnog dijela rada. Demonstrirana je upotreba kolačića, bearer tokena, mikroservisa i API mrežnog prolaza (*engl. gateway*), troslojne arhitekture, dva provajdera baze podataka, Docker-a, Keycloak provajdera identiteta i Keycloak REST API-ja, OpenID Connect protokola, brendiranja aplikacije, SSO i kreiranja jedinstvenog međusloja (*engl. middlware*) za autorizaciju.

5.1 Arhitektura sistema i korišteni alati

Kao što je već rečeno, sistem je sačinjen od nekoliko aplikacija. Slika 5.1 prikazuje način organizacije sistema. Kao što se vidi sa slike, sistem je podijeljen na dvije mašine. Na Linux operativnom sistemu je instaliran Docker. Instanca Keycloak servera je podignuta kao Docker kontejner, a kako bi se na siguran način uspostavila komunikacija sa drugim servisima, podešen je i HTTPS. Na Windows operativnom sistemu je instaliran .NET 8, NodeJS, Angular 18, PostgreSQL i SQL Server 22. Tu je podignuto više aplikacija koje čine Trustify sistem.



Slika 5.1 - Način organizacije sistema

Mašine se nalaze u različitim mrežama, sa IP adresama: 192.168.100.6 i 192.168.56.101.

Prva aplikacija predstavlja korisničku poslovnu aplikaciju TrustifyApp koja omogućuje rad sa sadržajem, kao što su slike, tekstualni sadržaj i članci. Front-end date aplikacije (Trustify.Features) je implementiran korištenjem Angular razvojnog okvira, a back-end se nalazi iza Ocelot API mrežnog prolaza sa kojim Trustify.Features aplikacija komunicira putem HTTPS protokola. Iza Ocelot API mrežnog prolaza se nalaze dva mikroservisa: Trustify.Backend.Features.FeaturesService (radi jednostavnosti biće korišten naziv FeaturesService) i Trustify.Backend.Features.ImageGeneratorService (radi jednostavnosti biće korišten naziv ImageGeneratorService). ImageGeneratorService je jednostavan mikroservis koji omogućuje korisniku da na osnovu zadate riječi pronađe/generiše sliku pozivanjem Europeana servisa[23]. FeaturesService je mikroservis koji delegira pozive kontrolera ka Features.Core biblioteci (implementiranoj korištenjem .NET 8 razvojnog okvira). Data Features.Core biblioteka (Trustify.Backend.Features.Core) se povezuje na odabranu bazu podataka korištenjem Entity Framework paketa funkcionalnosti. Izbor baze podataka zavisi od potreba klijenta za brzinom ili povoljnijim riješenjem, pa se tako nude dva provajdera baze podataka, a to su Microsoft SQL Server i PostgreSQL. Cilj ovih aplikacije jeste simulacija autorizacije i autentifikacije, gdje naglasak nije na samim funkcionalnostima, već na autorizaciji i autentifikaciji korisnika, kao i mogućnostima Angular razvojnog okvira za prikaz i upravljanje različitim UI komponentama. Angular aplikacija je razvijena kao samostalna (*engl. standalone*) aplikacija.

Druga aplikacija naziva TrustifyAdmin predstavlja administratorsku aplikaciju za upravljanje Keycloak administratorskim funkcionalnostima unutar jedne oblasti (*engl. realm*). Postojeći Keycloak administratorski panel je naklonjen programerima i obučenim administratorima, dok za menadžere informacionih sistema nije intuitivan, te im je lako napraviti grešku. Zbog toga je implementacija sopstvenog rješenja isplativa, kako zbog jednostavnosti upotrebe, tako i zbog ograničavanja permisija samih menadžera (kroz TrustifyAdmin aplikaciju korisnik ne može da doda ili obriše klijenta, ali može da registruje novog ili obriše postojećeg korisnika).

Dvije aplikacije prikazuju različit način rada sa Keycloak provajderom identiteta i različite načine autorizacije, što će biti objašnjeno u ostatku rada.

Što se tiče ostalih korištenih alata, za simulaciju primanja email pošte u svrhu promjene lozinke ili potvrđivanja email adrese, korišten je MailHog servis. Dizajn je rađen uz pomoć Angular Material biblioteke i CSS-a. Sistem se čuva i verzioniše u okviru jednog repozitorija na GitHub-u pod nazivom Trustify[24].

5.2 Podešavanje Keycloak provajdera identiteta

U septembru 2014. godine pušten je u upotrebu open-source alat za upravljanje identitetom i SSO, poznat kao Keycloak. [32] Zbog svoje pristupačnosti, skoro 5000 poznatih kompanija ga danas koristi, među kojima su CloudNative Inc. i Cybertech. U radu je opisana upotreba Keycloak-a i mogućnosti prilagođavanja sopstvenim potrebama. Podešavanje Keycloak servera je odrađeno na način opisan u ostatku poglavlja.

5.2.1 Podešavanje docker kontejnera

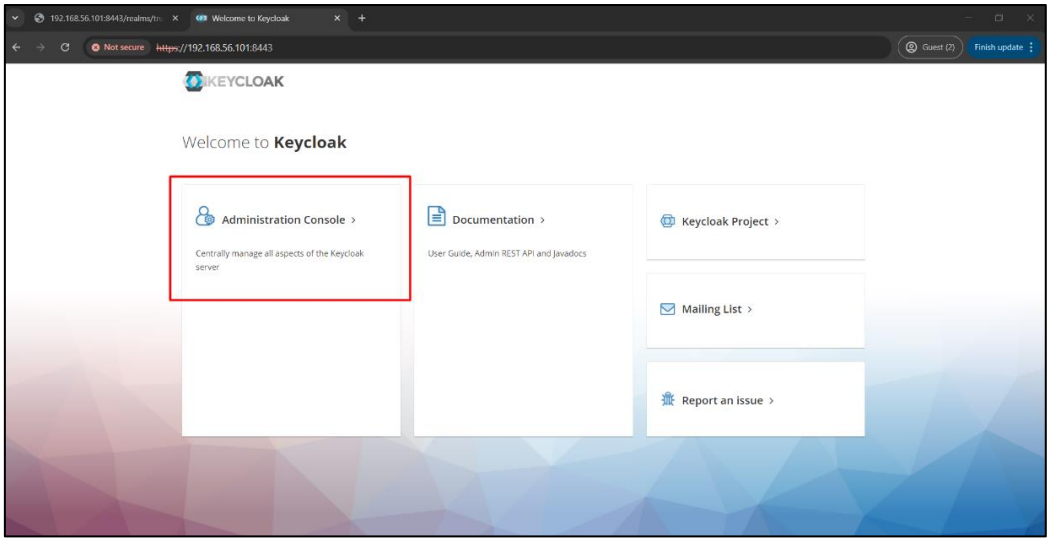
Na Linux Ubuntu 24.04. virtuelnoj mašini je instaliran Docker softver [25], koji omogućava instalaciju i podešavanje Keycloak kontejnera, u verziji 23.0.6. Da bi se na najlakši način

[23] Servis se može pronaći na sajtu: https://www.europeana.eu/en, kada se izvrši prijava na dati sistem.
[24] https://github.com/chimarry
[25] Docker je alat koji omogućava jednostavno kreiranje i objavljivanje aplikacija kao kontejnera u izolovanom okruženju. Više na linku: https://www.docker.com/

podesio Keycloak server, korišten je *docker-compose* fajl, čiji je sadržaj prikazan u isječku koda ispod. Kao što se vidi iz navedenog podešavanja, ova verzija Keycloak servera koristi PostgreSQL kao bazu podataka i samopotpisane sertifikate u svrhu korištenja HTTPS protokola na portu 8443. Da bi se pristupilo administratorskoj konzoli, uneseni su i kredencijali za podrazumijevanog administratora master oblasti (prve oblasti) Keycloak provajdera identiteta).

```
version: '3.7'

services:
  postgres:
    image: postgres:16.2
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      POSTGRES_DB: keycloak_db
      POSTGRES_USER: keycloak_db_user
      POSTGRES_PASSWORD: keycloak_db_user_password
    networks:
      - keycloak_network

  keycloak:
    image: quay.io/keycloak/keycloak:23.0.6
    command: start
    container_name: keycloak_trustify
    environment:
      KC_HOSTNAME: 192.168.56.101
      KC_HOSTNAME_PORT: 8443
      KC_HOSTNAME_STRICT_BACKCHANNEL: 'false'
      KC_HTTP_ENABLED: 'true'
      KC_HOSTNAME_STRICT_HTTPS: 'true'
      KC_HEALTH_ENABLED: 'true'
      KEYCLOAK_ADMIN: admin
      KEYCLOAK_ADMIN_PASSWORD: admin
      KC_DB: postgres
      KC_DB_URL: jdbc:postgresql://postgres/keycloak_db
      KC_DB_USERNAME: keycloak_db_user
      KC_DB_PASSWORD: keycloak_db_user_password
      KC_HTTPS_CERTIFICATE_KEY_FILE: /etc/x509/https/server.key
      KC_HTTPS_CERTIFICATE_FILE: /etc/x509/https/server.crt
    ports:
      - 8080:8080
      - 8443:8443
    volumes:
      - ./https:/etc/x509/https
    restart: always
    depends_on:
      - postgres
    networks:
      - keycloak_network

volumes:
  postgres_data:
    driver: local

networks:
  keycloak_network:
    driver: bridge
```

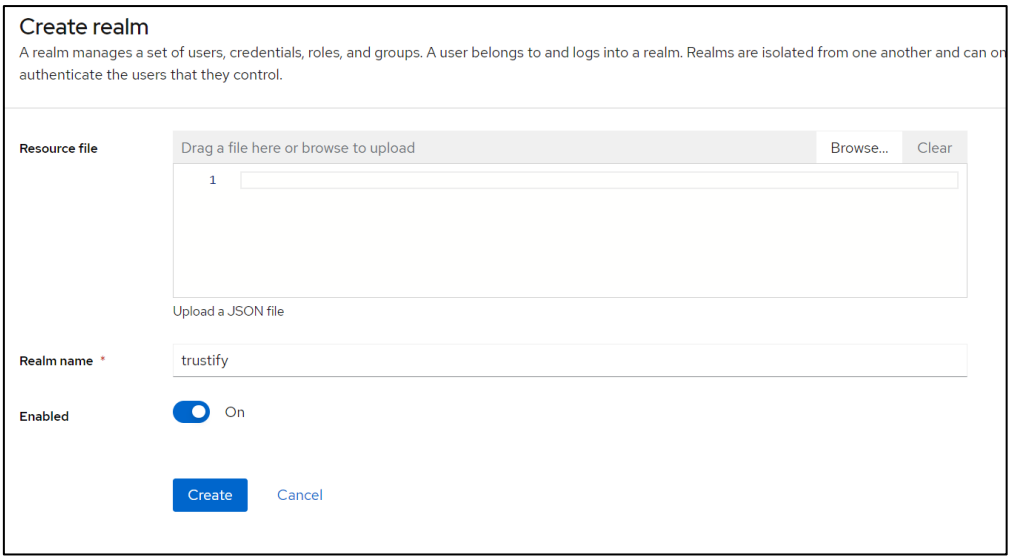
Kada se pokrene Keycloak sa komandom *sudo docker-compose up*, tada preko adrese https://192.168.56.101:8443 možemo da pristupimo datom serveru. Slika 5.2 prikazuje prozor koji se otvara.
Dalje podešavanje se vrši preko administratorske konzole na koju se vrši prijava sa kredencijalima iz *docker-compose.yml* fajla.



Slika 5.2 - Keycloak početna strana

5.2.2 *Podešavanje Keycloak-a kroz dostupni administratorski panel*

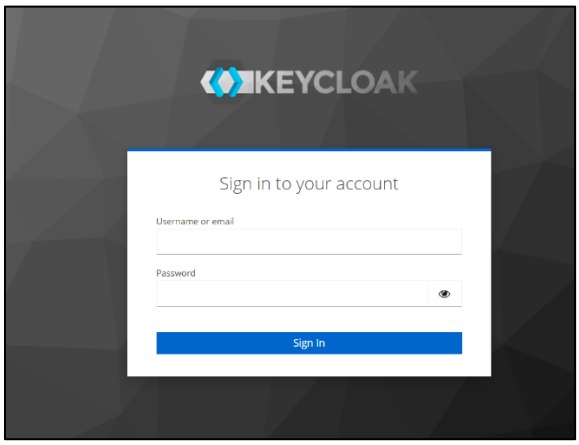
Keycloak sadrži podrazumijevanu master oblast. Trustify sistem koristi svoju oblast, tako ograničavajući pristup aplikaciji. Slika 5.3 prikazuje kreiranje oblasti.



Slika 5.3 - Kreiranje oblasti

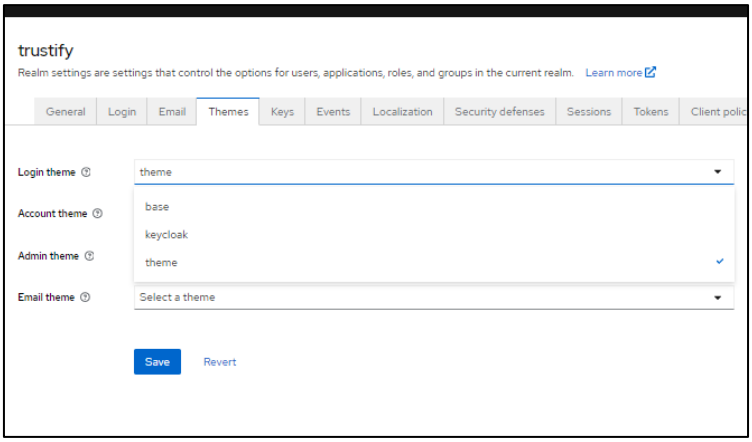
Da bi se dobila konfiguracija te oblasti, koristi se URL, a prilikom podešavanja autentifikacije na back-end strani, upravo će ugrađeni mehanizam za OpenID Connect da koristi obezbijeđeni URL: https://192.168.56.101:8443/realms/trustify/.well-known/openid-configuration.

Moguće je kroz administratorski panel promijeniti podešavanja oblasti. Kao što je rečeno u poglavlju 4.6.3, jedan od načina zaštite podataka korisnika i same aplikacije jeste brendiranje iste, dodavanje logo znaka, naziva ili autorskih prava. Takođe, kao što je objašnjeno u poglavlju 4.4, autentifikacija sa provajderom identiteta podrazumijeva preusmjeravanje zahtjeva za autentifikacijom ili autorizacijom ka provajderu identiteta, otvaranje stranice za prijavu datog IP, i uspješan ili neuspješan povratak na aplikaciju. Keycloak omogućava, u svrhu brendiranja, promjenu izgleda stranice za prijavu, koja podrazumijevano izgleda kao na slici 5.4.



Slika 5.4 - Podrazumijevana tema Keycloak-a

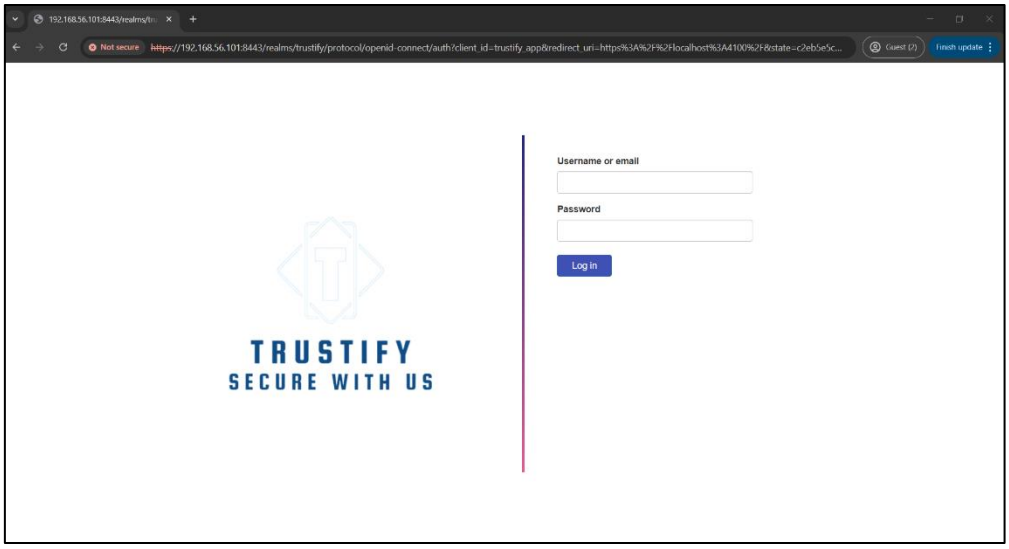
Da bi se promijenila tema za prijavu, potrebno je kreirati temu[26], kreirati odgovarajući folder unutar Keycloak docker kontejnera[27] i u taj folder kopirati temu sa Linux OS. Nakon tog je tema vidljiva kroz administratorski panel, i tema oblasti se mijenja izborom željene teme iz padajućeg menija, kao što je prikazano na slici 5.5.



Slika 5.5 - Podešavanje teme

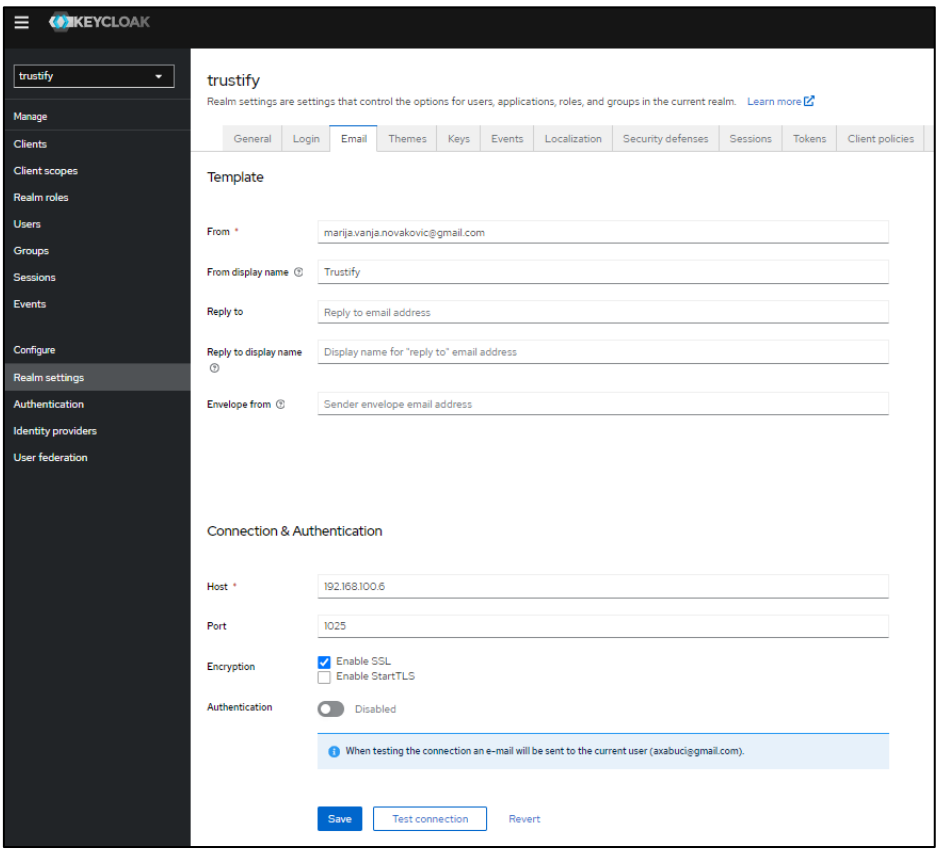
Kada se pristupa “trustify” oblasti, otvara se stranica za prijavu kao na slici 5.6.

[26] Keycloak koristi odgovarajući jezik kojim se mijenja tema, pa je potrebno prilagoditi HTML, CSS i JS kod datom obrascu koji Keycloak koristi.
[27] U folderu opt, zavisno od podešavanja kontejnera, može da se pronađe folder themes. Potrebno je kreirati folder koji nosi naziv teme i u tom folderu kreirati folder login.



Slika 5.6 - Trustify stranica za prijavu

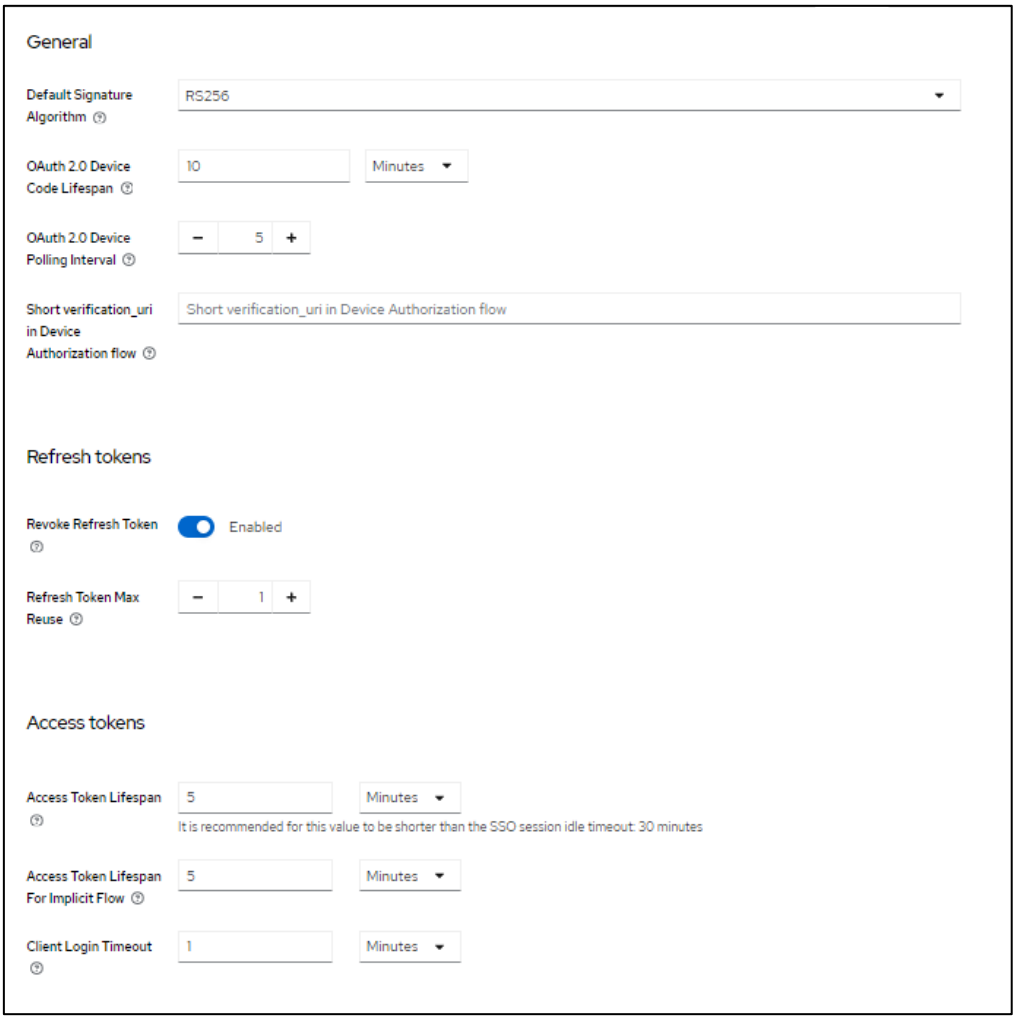
Vrlo često se u praksi javlja potreba za verifikacijom email adrese korisnika (koja se često koristi za autentifikaciju) ili promjena lozinke (kada se korisnik ne registruje sam). Takav vid dodatne sigurnosti je omogućen podešavanjem Keycloak mejl servera. U svrhu demonstriranja slanja mejlova, verifikacije prijave i promjene lozinke, iskorišten je MailHog SMTP server, [33] pokrenut na Windows mašini i registrovan na Keycloak-u na način prikazan na slici 5.7.



Slika 5.7 - Podešavanje konekcije na SMTP server

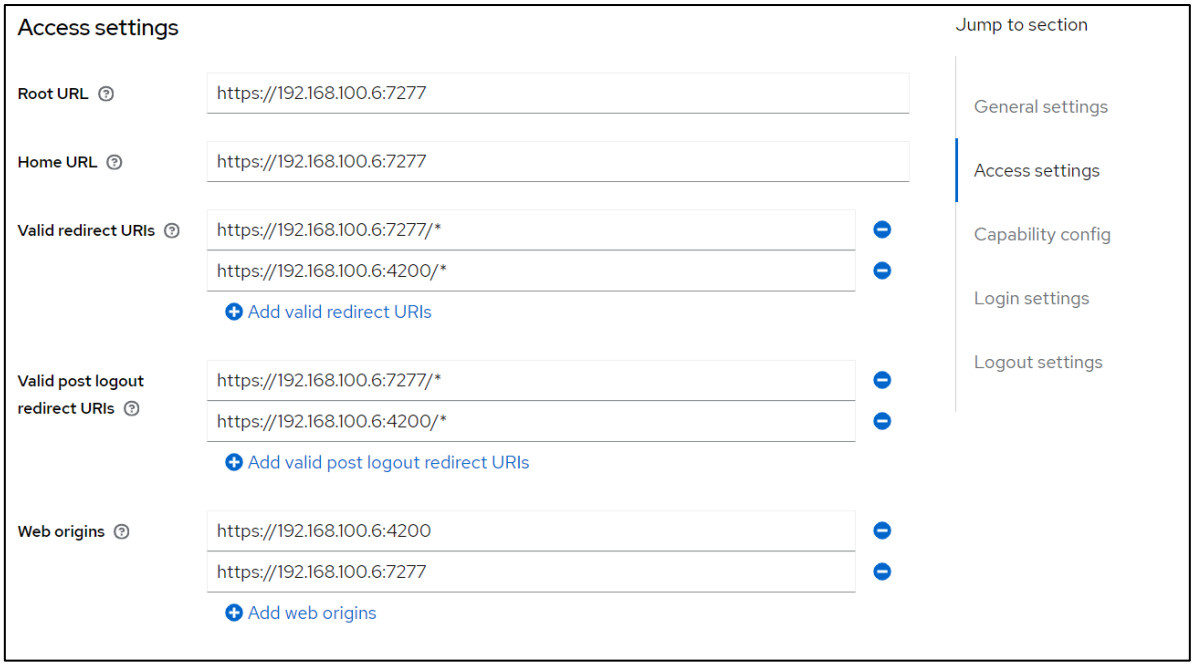
Kao što se vidi sa slike, potrebno je unijeti adresu kojom se pristupa SMTP servisu, port i mejl adresu sa koje se šalju poruke korisnicima. Kao što je rečeno, prilikom rada sa pristupnim

tokenima, radi bolje sigurnosti, potrebno je podesiti trajanje istog. Keycloak omogućava podešavanje karakteristika sesije oblasti izborom opcije Tokeni iz navigacionog menija. U Trustify aplikaciji, trajanje tokena je podešeno na 5 min, omogućena je rotacija tokena za obnovu (čime se postiže sprečavanje zloupotrebe tokena). Slika 5.8 prikazuje moguću konfiguraciju.



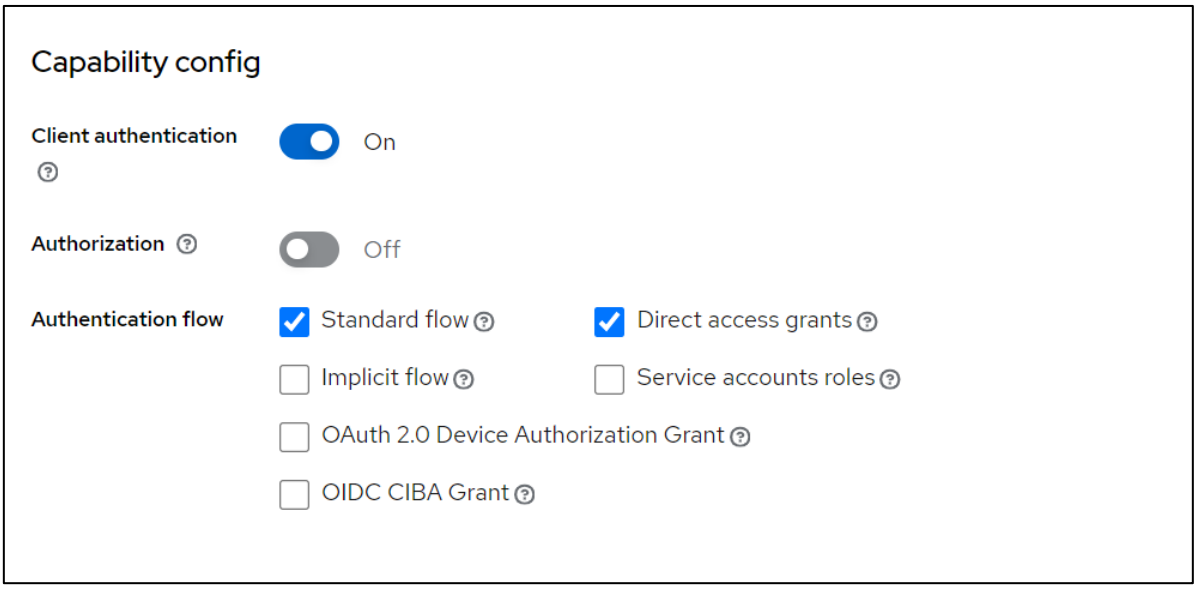
Slika 5.8 - Podešavanje tokena

Nakon što je kreirana oblast, potrebno je registrovati klijente u okviru te oblasti. Trustify koristi autentifikaciju sa identifikatorom klijenta (*client_id*) i sa tajnom lozinkom (*client_secret*). Automatski je omogućen SSO sa jednog klijenta na drugi korisnicima iste oblasti, ukoliko naravno postoje odgovarajuće permisije. Nakon što se klijent kreira, dobija se panel sa podešavanjima datog klijenta, koji uključuje pristupnu konfiguraciju kao što su kredencijali i način autentifikacije, dozvoljeni povratni URL, dozvoljeni izvorni URL zahtjeva ka Keycloak serveru (prikazano na slikama 5.9 i 5.10).



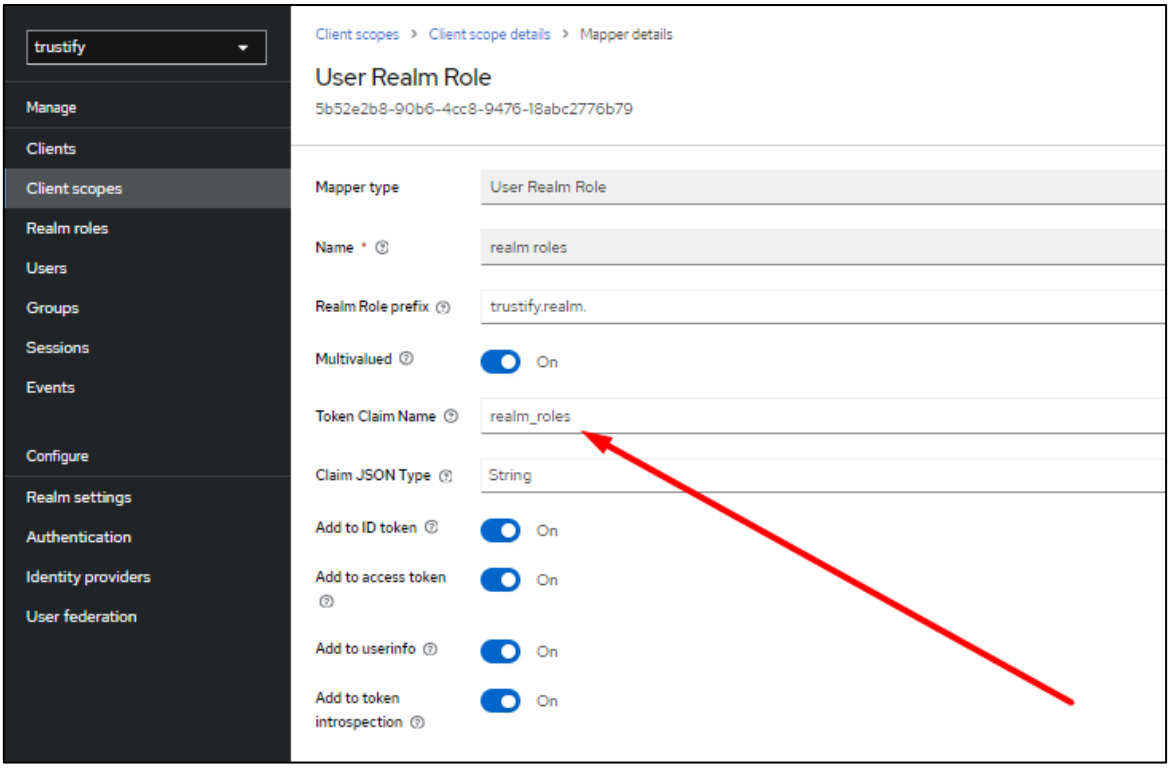
Slika 5.9 - Podešavanje URL-ova

U skladu sa pomenutim OIDC protokolom, ponuđeno je vise metoda autentifikacije (prikazano na slici 5.10).



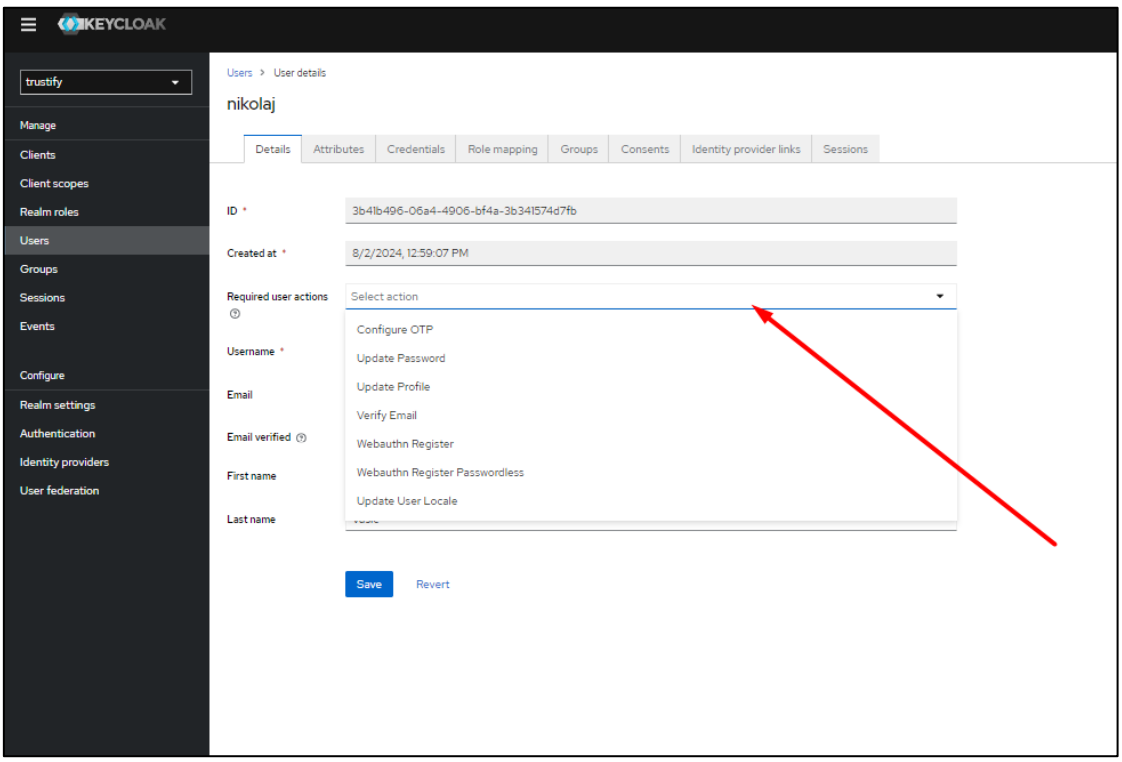
Slika 5.10 - Izbor autorizacione dozvole

Keycloak administratorski panel omogućuje odabir kontrole pristupa, a Trustify aplikacija koristi kontrolu pristupa zasnovanu na ulogama. Svaki klijent sadrži listu scope-ova, među kojima je moguće dodati scope uloge, podesiti oblik uloge unutar tokena ili korisničkih informacija, omogućiti ili onemogućiti vidljivost pojedinih podataka korisnika unutar tokena, endpoint-a za dobavljanje korisničkih podataka i slično. Uloga scope-a je ranije objašnjena. Slika 5.11 prikazuje podešavanje mapiranja uloga oblasti u odgovarajući oblik prilikom slanja preko mreže. Moguće je odabrati ugrađene mapere, pa onda prilagoditi naziv, prefiks koji će se dodati prilikom prijave korisnika, format kojim će se slati preko mreže i u kojim tokenima će se nalaziti. Slično je kod dodavanja drugih mapiranja.



Slika 5.11 - Podešavanje uloga oblasti

Kada se dodaje novi korisnik, pored unosa podataka, moguće je unijeti i akcije koje se zahtijevaju od korisnika (isto je omogućeno kroz TrustifyAdmin aplikaciju). Slika 5.12 prikazuje akcije koje podržava Keycloak.



Slika 5.12 - Zahtjevane korisničke akcije

5.3 TrustifyAdmin aplikacija

Suština TrustifyAdmin aplikacije jeste prilagođavanje upravljanja opcijama provajdera identiteta potrebama informacionog sistema i kadra koji sa tim sistemom radi. Aplikacija je sačinjena od back-end dijela koji pozive delegira Keycloak API-ju, kao i front-end dijela.

5.3.1 Back-end aplikacija

Back-end aplikacije je implementiran upotrebom ASP.NET Web API razvojnog okvira, nad platformom .NET 8, koji podržava C# programski jezik u verziji 12. Budući da se koristi razvojni okvir za implementiranje RESTful arhitekture, potrebno je pratiti tok koji taj okvir nudi. Trustify.Backend.Admin aplikacija ima podešeno logovanje zahtjeva i akcija, uz pomoć Serilog NuGet paketa. Važnost vođenja zapisa o izuzecima koji se dese, ali i o akcijama koje korisnik sprovodi nad sistemom je ranije rečena. Za strukturisanje podataka i slanje preko mreže i kroz sistem se koriste DTO objekti. Za mapiranje objekata koristi se AutoMapper. Time se postiže razdvajanje odgovornosti među klasama. Takođe, koristi se ugrađeni mehanizam razrješavanja zavisnosti (*engl. Dependency Injection*). Na osnovu podešavanja se za odgovarajući interfejs pronalazi implementacija i kreira se objekat, a takođe je vrlo jednostavno promijeniti implementaciju bez da se mijenjaju brojne klase. Primjer Program.cs konfiguracije iz Trustify.Backend.Admin aplikacije je prikazan na slici 5.13.

```
builder.Services.ConfigureAuthorization(builder.Configuration);
builder.Services.ConfigureAuthentication(builder.Configuration);
builder.Services.Configure<FrontendConfig>(builder.Configuration.GetSection("Frontend"));
builder.Services.Configure<KeycloakOptions>(builder.Configuration.GetSection("KeycloakOptions"));

builder.Services.AddScoped<IRoleService, KeycloakRoleService>();
builder.Services.AddScoped<IGroupService, KeycloakGroupService>();
builder.Services.AddScoped<IUserService, KeycloakUserService>();
builder.Services.AddScoped<IClientService, KeycloakClientService>();
builder.Services.AddScoped<IHttpService, HttpService>();
builder.Services.AddScoped<IExceptionHandler, ExceptionHandler>();
builder.Services.AddSingleton(AutoMapperConfig.CreateMapper());
```

Slika 5.13 - Konfiguracija u Program.cs fajlu

Podešen je HTTP klijent tako da ne zahtijeva provjerenost sertifikata (sertifikati su samopotpisani). Polisa bazirana na kolačićima je podešena tako da sadrži kolačić koji se prenosi preko HTTPS protokola, koji ne može da se modifikuje upotrebom JavaScript-a i koji ne zahtijeva pristanak korisnika za kolačić koji nije nužan. Navedeno je prikazano na slici 5.14.

```
FlurlHttp.Clients
        .WithDefaults(builder => builder.ConfigureInnerHandler(x =>
                                {
                                    x.ServerCertificateCustomValidationCallback = (message, cert, chain, errors) => true;
                                })
                    );
builder.Services.Configure<CookiePolicyOptions>(options =>
{
    options.CheckConsentNeeded = context => false;
    options.HttpOnly = HttpOnlyPolicy.None;
    options.Secure = CookieSecurePolicy.Always;
});
```

Slika 5.14 - Podešavanje polise kolačića

Podešavanje autorizacije i autentifikacije je u odvojenim statičkim klasa, implementirano u okviru metode proširivanja (*engl. Extension methods*). Autentifikacija je podešena na način prikazan na slici 5.15.

```
services.AddAuthentication(options =>
{
    options.DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = OpenIdConnectDefaults.AuthenticationScheme;
})
.AddCookie(options =>
{
    options.Cookie.Name = "TRF_COOKIE";
    options.Cookie.HttpOnly = true;
    options.Cookie.SecurePolicy = CookieSecurePolicy.Always;
    options.LoginPath = "/api/v1.0/auth/login";
    options.AccessDeniedPath = "/api/v1.0/auth/access-denied";
    options.ExpireTimeSpan = TimeSpan.FromMinutes(2);
    options.Events = new CookieAuthenticationEvents...;
})
.AddOpenIdConnect(options =>
{
    options.Authority = keycloakOptions.Authority;
    options.ClientId = keycloakOptions.ClientId;
    options.ClientSecret = keycloakOptions.ClientSecret;
    options.ResponseType = "code";
    options.Backchannel = new HttpClient(new HttpClientHandler()...)...;
    options.Backchannel.DefaultRequestHeaders.Add("Acces-Control-Allow-Origin",
        frontendConfig.BaseUrl);
    options.BackchannelHttpHandler = new HttpClientHandler()...;
    options.SaveTokens = true;
    options.Scope.Clear();
    options.Scope.Add("openid");
    options.SkipUnrecognizedRequests = true;
    options.RequireHttpsMetadata = true;
    options.TokenValidationParameters = new TokenValidationParameters
    {
        RoleClaimType = ClaimTypes.Role,
        RequireExpirationTime = true,
        ValidateLifetime = true,
    };
    options.Events = new OpenIdConnectEvents
```

Slika 5.15 - Podešavanje autentifikacije

Kada se konfiguriše autentifikacija, potrebno je razriješiti šeme koje će da koristi međusloj , pa se koristi metoda *AddAuthentification*. Korištene su ASP.NET šeme, a kolačić je podešen tako da ima poseban naziv, da bude siguran, da ističe nakon dvije minute i da se ne može mijenjati kroz JavaScript. OIDC protokol je konfigurisan preko metode *AddOpenIdConnect,* pri čemu se atribut *Authority* odnosi na provajdera identiteta, u ovom slučaju je to URL do konfiguracije Keycloak servera, atributi *ClientId* i *ClientSecret* odnose na kredencijale *trustify_admin* klijenta (date aplikacije *Trustify.Admin*) koji se koriste prilikom autentifikovanja aplikacije na Keycloak server i atribut *scope* je podešen na openid. Zahtijeva se upotreba HTTPS-a, preskaču se zahtjevi koji su nepoznati i podešava se validacija tokena. Na slici nije prikazano podešavanje dobavljanja tokena za obnovu, koje se obavlja slanjem posebnog zahtjeva Keycloak serveru, pri čemu se u kodu promijene vrijednosti pristupnog tokena i tokena za obnovu. Upotreba kolačića za čuvanje informacija o identitetu nije toliko popularna, i u sljedećoj aplikaciji TrustifyApp biće prikazana upotreba Bearer tokena pri svakom zahtjevu. Kada korisnik nije autentifikovan, vraća se statusni kod 401.

Autorizacija podrazumijeva da je korisnik autentifikovan (u slučaju da nije vraća se 401), pa podatke potrebne za autorizaciju može da dobavi iz stvorenog identiteta korisnika. Podešava se na sličan način kao i autentifikacije, a to je preko metode *AddAuthorization,* koja je prisutna u ASP.NET kako bi razriješila registrovane rukovatelje autorizacije (*engl. Authorization Handlers),* i/ili mapirala odgovarajuće polise sa načinima validacije. Iako je Keycloak provajder identiteta, potrebno je uz pomoć informacija obezbijediti sigurnost API-ja. Primjer

kako je autorizacija implementirana na nivou back-end-a TrustifyAdmin panela je prikazan na slici 5.16, gdje su dodane tri polise. Prva zahtijeva autentifikovanog korisnika (demonstracije radi, ukoliko nije kreiran identitet, desiće se greška. U datoj aplikaciji je to podešeno na nivou autentifikacije, pa nije bilo potrebe da se dodaje ta polisa), druga zahtijeva ulogu administratora, a treća ulogu SuperAdministratora. Obje uloge su uloge oblasti (*engl. realm roles*), jer su nosioci ovih uloga menadžeri svih klijenata. Koristi se ugrađena metoda *RequireClaim*, koja prihvata ime i vrijednost validne tvrdnje kao argumente.

```
1 reference
public static void ConfigureAuthorization(this IServiceCollection services, IConfiguration configuration)
{
    services.AddAuthorizationBuilder()
        .AddPolicy(TrustifyPolicy.Authenticated, p => p.RequireAuthenticatedUser())
        .AddPolicy(TrustifyPolicy.All, p => p.RequireClaim(RealmRole.TokenClaimName, RealmRole.Administrator))
        .AddPolicy(TrustifyPolicy.Restricted, p => p.RequireClaim(RealmRole.TokenClaimName, RealmRole.SuperAdministrator));
}
```

Slika 5.16 - Podešavanje autorizacije

Nije dovoljno dodati autorizaciju i autentifikaciju, potrebno je naglasiti aplikaciji da koristi međuslojeve, pri čemu je redoslijed istih bitan[28]. Navedeno je prikazano na slici 5.17.

```
app.UseHttpsRedirection();

app.UseMiddleware<ErrorHandlingMiddleware>();

app.UseCookiePolicy();

app.UseCors("CorsPolicy");


app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Slika 5.17 - Middleware-i

Kada se desi zahtjev za kreiranjem korisnika, uloge, grupe, dobavljanjem klijenata i slični zahtjevi koji se tiču Keycloak servera, aplikacija šalje zahtjev Keycloak serveru. Primjer slanja zahtjeva je prikazan na slici 5.18. *ResultMessage* je implementirana generička klasa koja sadrži rezultat, kod greške i detalje o istoj, kako bi povratni tip HTTPS zahtjeva bio konzistentan. *Flurl* je biblioteka koja olakšava rad sa .NET HTTP klijentom. *ExceptionHandler* jeste klasa koja obrađuje izuzetke različitog tipa i zavisnosti od izuzetka, formira *ResultMessage* objekat. Naravno, moguće je koristiti *when*[29] da bi se specifikovalo kada sve da se uhvati izuzetak u *catch* bloku. Takođe je moguće proslijediti izuzetak i obraditi ga u globalnom međusloju za obradu grešaka i izuzetaka. Aplikacija posjeduje globalnog rukovatelja kako se ne bi desilo da se neki izuzetak ne obradi, ali je implementirana tako da se ne posmatra sve kao izuzetak, već se vraća odgovarajući indikator o izvršenju operacije (npr. *OperationStatus.Exists*).

[28] Middleware-i su organizovani tako da se pri izlasku iz jednog middleware-a i obrade zahtjeva prelazi u drugi middleware.
[29] https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/when

```
/// <summary>
/// Adds group of users to the realm.
/// </summary>
/// <param name="group">Groups' data</param>
/// <param name="token">Access token of the logged in user</param>
/// <returns></returns>
2 references
public async Task<ResultMessage<bool>> AddGroup(GroupDTO group, string token)
{
    try
    {
        IFlurlResponse? response = await httpService.GetAdminBaseUrl(token)
                                                    .AppendPathSegment("/groups")
                                                    .WithOAuthBearerToken(token)
                                                    .PostJsonAsync(group);

        return new ResultMessage<bool>(true);

    }
    catch (Exception ex)
    {
        (OperationStatus status, string message) = exceptionHandler.HandleException(ex);
        return new ResultMessage<bool>(status, message);
    }
}
```

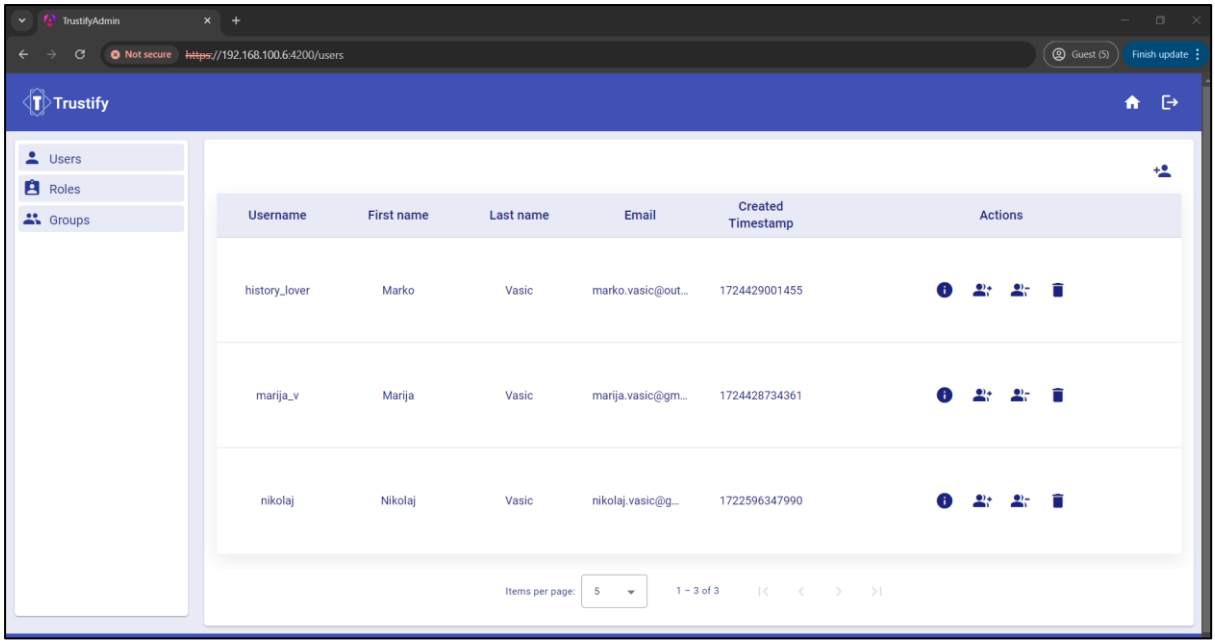
Slika 5.18 - Dodavanje grupe u trustify oblast

5.3.2 Front-end aplikacija

Trustify.Admin je naziv front-end aplikacije za administratorski panel Keycloak servera. Implementiran je korištenjem Angular razvojnog okvira, koji za razvoj aplikacija koristi Typescript programski jezik. Izvršava se na NodeJS 20.11.1. Kako fokus nije na izgledu aplikacije, korištene su teme Angular Material modula. Podešen je konfiguracioni fajl *proxy.config.json* za slanje zahtjeva serveru i podešen je HTTPS sa samopotpisanim sertifikatima.Važno je naglasiti da konfiguracija odgovara samo simulaciji *proxy* servera po pitanju delegiranja zahtjeva. Da bi se olakšalo implementiranje Angular koda koji poziva REST API Trustify.Admin aplikacije, korišten je alat za generisanje koda iz OpenAPI specifikacije. Korištene su sljedeće komande:

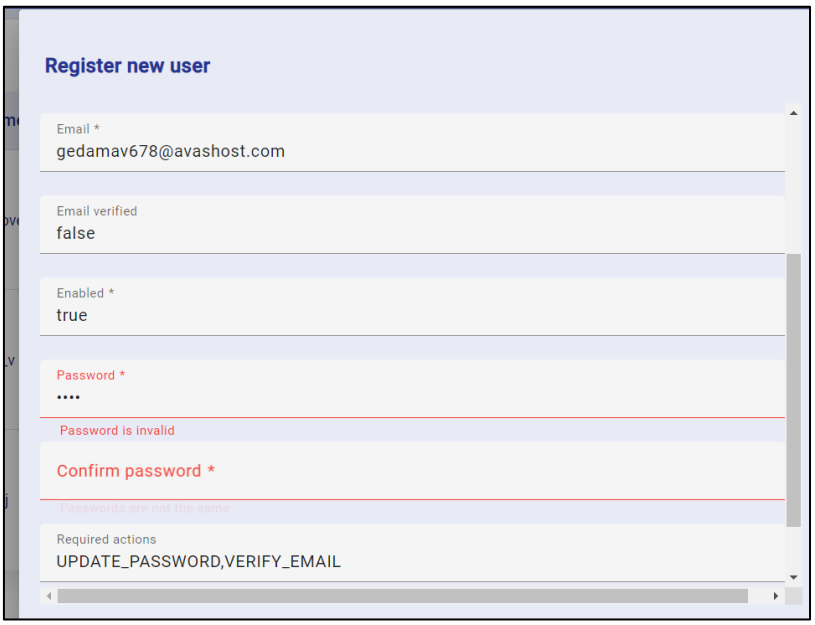
```
node_modules/.bin/api-spec-converter --from=openapi_3 --to=swagger_2 --
syntax=json --order=alpha openapi.json > swagger.json

node_modules/.bin/ng-swagger-gen -i swagger.json -o ./src/app/api
```

U "*api*" folderu aplikacije je generisan kod sa servisima i modelima sa back-end-a. Kada se aplikacija pokrene na portu 4200, poziva se back-end aplikacije. Sva autentifikacije i autorizacija se vrše na back-end-u. Trustify.Admin aplikacija ima podešen HTTP presretač zahtjeva (*engl. Interceptor*) koji obradi grešku ako je do nje došlo, a servis *DisplayMessageService* prikaže korisniku poruku sa rezultatom kao *pop-up*. Kako korisnik nije autentifikovan (kolačići nisu prisutni ili nisu validni), OpenID Connect podešeni protokol poziva Keycloak server na adresi https://192.168.56.101:8443. Otvara se forma za prijavu i nakon što se korisnik uspješno prijavi, otvara se stranica za upravljanje korisnicima prikazana na slici 5.19. U radu neće biti opisane druge funkcionalnosti, jer bi to prešlo obim rada. Operacije koje je moguće izvesti nad korisnicima su prikaz informacija, dodavanje u grupu, uklanjanje iz grupe i brisanje korisnika. Moguće je dodati novog korisnika. Važno je napomenuti da novi korisnik neće moći da koristi funkcionalosti TrustifyAdmin panela jer nema uloge oblasti i nije ih moguće dodijeliti kroz TrustifyAdmin aplikaciju. Kada bi ih imao, ne bi mogao da poziva administratorski API Keycloak IP, ukoliko nema podešene odgovarajuće uloge *realm_management* klijenta.



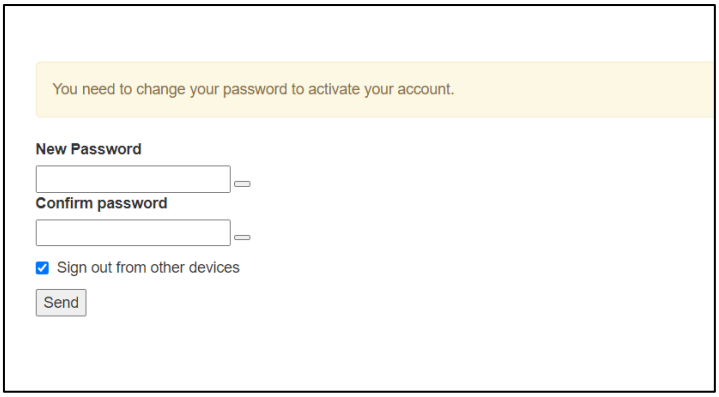
Slika 5.19 - Sekcija sa korisnicima

Prilikom dodavanja korisnika, kao i na svim drugim formama u Trustify sistemu, vrši se validacija unosa. U primjeru na slici unesena je prekratka lozinka koja uz to nije potvrđena. Unesen je i privremena email adresa.

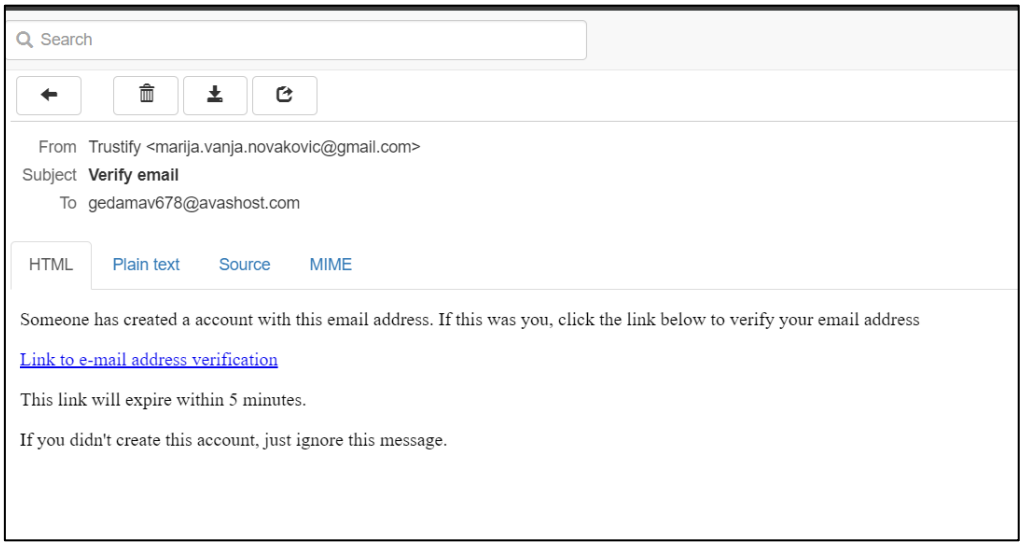


Slika 5.20 - Dodavanje korisnika

Kada se korisnik kreira, moguće ga je dodati u grupe. Grupa podrazumijeva skup uloga, tako da se korisniku ne moraju dodavati pojedinačno sve uloge. Ukoliko se doda nova uloga u grupu, ili se izbaci neka od postojećih uloga, automatski će se to odraziti na sve pripadnike te grupe, tako da to olakšava upravljanje ulogama. Prilikom prijave na sistem novokreiranog korisnika, i unosa privremenih kredencijala, Keycloak zahtijeva promjenu lozinke, što je prikazano na slici 5.21. Nakon uspješne promjene lozinke, korisnik treba da verifikuje email adresu, a to može da uradi preko email poruke, klikom na link (Slika 5.22).



Slika 5.21 - Promjena lozinke



Slika 5.22 - Verifikacija e-mail adrese

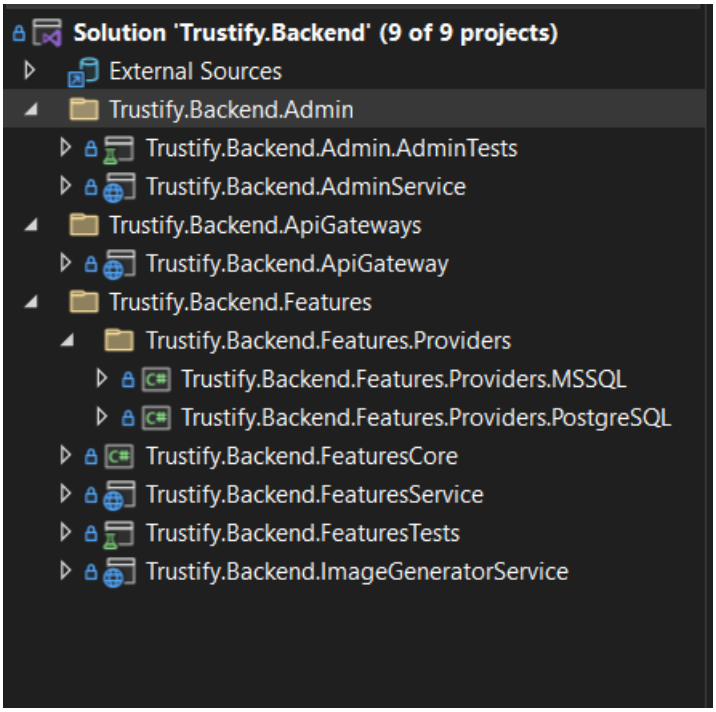
Klik na „Kuća“ dugmić će da otvori TrustifyApp aplikaciju, a budući da je podržan SSO, korisnik će moći da izbjegne ponovnu prijavu. Iako neće biti prikazano dodavanje uloga i grupa, biće objašnjeno kako funkcionišu. Kada se dodaje grupa, moguće je odabrati uloge koje pripadaju selektovanom klijentu (nije moguće dodati uloge oblasti u grupu, radi sigurnosnih ograničenja aplikacije). Prilikom dodavanja uloge unose se naziv uloge i opis. Korisniku nije moguće dodijeliti pojedinačnu ulogu, moguće je samo dodijeliti pripadnost određenoj grupi. Odjava se vrši klikom na dugmić za odjavljivanje u gornjem desnom uglu aplikacije, i pri tom se korisnik odjavljuje i sa TrustifyApp aplikacije.

5.4 TrustifyApp aplikacija

TrustifyApp aplikacija služi najviše da demonstrira upotrebu različitih uloga na jednostavnom sistemu upravljanja sadržajem. Tema je generička, budući da skoro svaka poslovna Internet aplikacija ima potrebu za slikama, tabelama i tekstualnim sadržajem kao osnovnim komponentama, a vrlo često su takvi podaci strukturisani u nekoj relacionoj bazi podataka.
Što se implementacije TrustifyApp aplikacije tiče, postoje back-end i front-end aplikacije. Back-end aplikacije posjeduje neke sličnosti sa Trustify.Backend.AdminService aplikacijom, jer ipak koriste isti razvojni okvir ASP.NET. U ostatku poglavlja biće opisane stvari koje nisu opisane u prethodnom poglavlju.

5.4.1 *Back-end aplikacije*

Da bi se pokazala upotreba mikroservisa, aplikacija ima dvije funkcionalnosti: upravljanje sadržajem i pretraga Evropskih muzeja i galerija za određenim slikama. Može se reći da su te dvije cjeline odvojene, s obzirom da jedna poziva eksterni servis, a druga ima svoju bazu podataka. Ovakva podjela je urađena i da bi se demonstriralo podešavanje autentifikacije i autorizacije na API mrežnom prolazu koji je skoro nezaobilazan prilikom korištenja mikroservisne arhitekture. Struktura je prikazana na Slika 5.23.



Slika 5.23 - Struktura back-end sistema

5.4.1.1 *Trustify.Backend.FeaturesService aplikacija*

Ova aplikacija nudi RESTful API za upravljanje sadržajem TrustifyApp aplikacije. FeaturesService predstavlja ASP.NET Web API aplikaciju koja prima zahtjeve od API mrežnog prolaza, i delegira ih sloju biznis logike FeaturesCore (.NET biblioteka). Podešeno je logovanje korisničkih aktivnosti i izuzetaka, a za mapiranje između slojeva se koristi AutoMapper. Da bi se olakšao rad sa bazom podataka, koristi se ORM alat EntityFramework. Budući da se koristi Entity Framework, nije potrebna upotreba repozitorija. Kao što je ranije rečeno, teži se upotrebi interfejsa i apstrakcija, pa se koristi DI pristup. Entity Framework danas ima brojne mogućnosti, i nudi različite pristupe prilikom modelovanja baze (generisanje šeme iz koda ili generisanje koda iz šeme). Upotreba ORM alata omogućava lakši prelazak na drugu bazu podataka, pa je implementirana i promjena baze podataka sa SQL na PostgreSQL i obratno. Da bi se to omogućilo, koriste se migracije. Za svaki provajder baze podataka je potrebno dodati odvojen projekat u koji će biti dodatne migracije, a koji će da se registruje prilikom registracije baze u Program.cs klasi glavnom projekta. EF migracije takođe omogućuju praćenje verzije baze (besplatno), a podržano je i generisanje SQL ili PostgreSQL skripte. Koja baza će biti korištena se određuje na osnovu podešavanja konekcionog stringa iz *appsettings.json* fajla koji ujedno predstavlja osnovni konfiguracioni fajl ASP.NET Web API aplikacije. Poželjno je konekcioni string za bazu čuvati u nekom sigurnom okruženju. Da bi se povećala sigurnost, aplikacija se u produkciji ne može konektovati na bazu pod

administratorskim privilegijama, već je kreiran novi korisnik. Registrovanje više provajdera u DI je prikazano na slici 5.24.

```
/// Configure database context with appropriate database driver (in this case Sql Server .NET driver is used).
/// It is neccessary that connection string is also available in the configuration.
/// /// </summary>
/// <param name="serviceCollection"></param>
/// <param name="configuration">Configuration object that contains information about database connection string.</param>
1 reference
public static void ConfigureDb(this IServiceCollection serviceCollection, IConfiguration configuration)
{
    string connectionString = configuration.GetConnectionString(ApiSettingsKeys.DatabaseConnectionStringKey)
        ?? throw new ArgumentNullException(nameof(configuration));
    string databaseProvider = configuration.GetSection(ApiSettingsKeys.ConnectionStringsKey).GetValue<string>(ApiSettingsKeys.DatabaseProviderKey

    serviceCollection.AddDbContext<TrustifyDbContext>(optionsAction: options =>
    {
        switch ((DatabaseProvider)Enum.Parse(typeof(DatabaseProvider), databaseProvider))
        {
            case DatabaseProvider.SqlServer:
                options.UseSqlServer(connectionString,
                                     x => x.MigrationsAssembly("Trustify.Backend.Features.Providers.MSSQL"))
                .ReplaceService<IMigrationsSqlGenerator, SqlServerMigrationsGenerator>(); break;
            case DatabaseProvider.PostgreSql:
                options.UseNpgsql(connectionString,
                                     x => x.MigrationsAssembly("Trustify.Backend.Features.Providers.PostgreSQL"))
                        .ReplaceService<IMigrationsSqlGenerator, PostgreSqlMigrationsGenerator>(); break;
            default: throw new Exception("Not supported");
        }
    }
    , contextLifetime: ServiceLifetime.Scoped
    , optionsLifetime: ServiceLifetime.Scoped);
}
```

Slika 5.24 - Provajderi baze podataka sa DI

FeaturesService aplikacija koristi "lagane" kontrolere, kojima je osnova uloga da zaprime zahtjev koji je prije tog prošao kroz lanac međuslojeva, da validiraju podatke i da proslijede zahtjev back-end Core projektu koji sadrži biznis logiku i mijenja i čita unose iz baze. Obično ne sadrže više od 3-4 linije koda. Iako linije koda nisu validna metrika, čitaocu mogu da dočaraju tu lakoću kontrolera. Modeli koji se zaprime na akciji kontrolera prosljeđuju se kao DTO objekti Core servisima[30], a ovi se mapiraju u/iz EF entitetskih klasa. Entitetske klase EF moraju da ispunjavaju određene uslove kako bi predstavljale tabele iz baze podataka i kako bi pratile nametnutu strukturu biblioteke. Izbjegnuto je korištenje anotacija, jer se koristi EF FluentAPI, ali klase i dalje koriste navigacione atribute i svaka izmjena objekta entitetske klase se prati i potencijalno sačuva u bazi, pa je upotreba DTO klasa sigurnija. Baza je sačinjena od tabela: ImageContents, TextualContents, Sections, Roles i veznih tabela RoleSection (*..*), SectionTextualContents (*..*) I ImageContentSections (*..*). Tabela Roles je dodana da bi se prikazala mogućnost zabrane pristupa dijelovima sadržaja, a ne samo endpoint-ima API-ja.
U radu neće biti objašnjeno korištenje EF, kao ni prikazani dijelovi back-end koda ovog projekta.

5.4.1.2 Trustify.Backend.ImageGeneratorService aplikacija

Ovaj jednostavni mikroservis nudi jedan endpoint za pronalazak neke slike u bazi sistema koji vodi zapise o Evropskim muzejima i galerijama (Europeana). Da bi se koristio dati servis, potrebno je kreirati nalog i zatražiti API ključ kojim se pristupa API-ju aplikacije (takođe primjer najjednostavnije autentifikacije jedne aplikacije drugoj).

[30] Termin servis u domenu Core projekta se ne odnosi na Web servis, već na menadžersku klasu koja upravlja odgovarajućom logikom: *ImageContentService*, *TextContentService* i sl.

5.4.1.3 Trustify.Backend.ApiGateway aplikacija

U svrhu implementacije API Mrežnog prolaza za aplikaciju TrustifyApp, korišten je Ocelot [34]. Kreiran je ASP.NET Web API projekat, bez kontrolera. Dodan je *ocelot.json* fajl kao konfiguracioni fajl Ocelot-a. U Program.cs fajlu su izvršena dodavanja potrebnih međuslojeva i konfiguracija. Za autentifikaciju i autorizaciju su iskorištene ugrađene šeme za rad sa Bearer JWT tokenima. Podešena je konekcija ka Keycloak serveru (Slika 5.25).

```
builder.Services
    .AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, o =>
    {
        o.MetadataAddress = "https://192.168.56.101:8443/realms/trustify/.well-known/openid-configuration";
        o.RequireHttpsMetadata = true;
        o.Authority = "https://192.168.56.101:8443/realms/trustify";
        o.Audience = "account";
        o.Backchannel = new HttpClient(new HttpClientHandler()
        {
            ServerCertificateCustomValidationCallback = (a, b, c, d) => true
        });
        o.SaveToken = true;
    });
```

Slika 5.25 - JWT Bearer podešavanje šeme

Da bi se implementirala kontrola pristupa bazirana na ulogama, takva da korisnik može da ima više uloga, bilo je potrebno implementirati sopstveni međusloj za autorizaciju koji se dodaje u Ocelot konfiguraciju (Slika 5.26).

```
var configuration = new OcelotPipelineConfiguration
{
    AuthorizationMiddleware = OcelotAuthorizationMiddleware.Authorize()
};

await app.UseOcelot(configuration);

await app.RunAsync();
```

Slika 5.26 - Ocelot međusloj tok

Jednostavna implementacija međusloja koji funkcioniše na prepoznavanju minimalne potrebne uloge korisnika za pristup resursu je prikazana na slici 5.27.

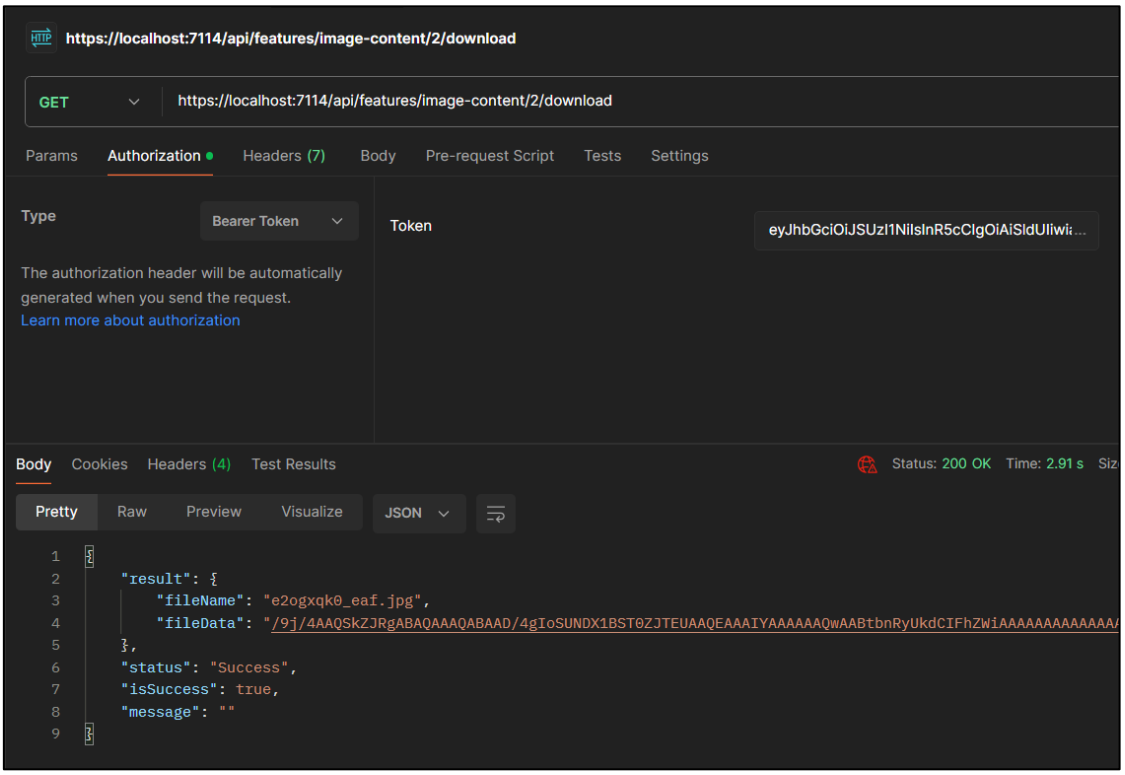
```
1 reference
public static Func<HttpContext, Func<Task>, Task> Authorize()
{
    return async (httpContext, next) =>
    {
        bool isAuthorized = false;
        var downstreamRoute = httpContext.Items.DownstreamRoute();
        downstreamRoute.RouteClaimsRequirement.TryGetValue(ClaimsRequirement, out string? roles);

        // Get roles
        string[] realRoles = roles.ToStringArray();
        Root? item = JsonConvert.DeserializeObject<Root>(httpContext.User.Claims.First(x => x.Type == ClaimType).Value)
              ?? new Root(null);
        if (item != null && item.TrustifyApp != null)
        {
            string[] claimRoles = [.. item.TrustifyApp.Roles];
            isAuthorized = claimRoles.Intersect(realRoles).Any();
        }
        if (!isAuthorized)
        {
            httpContext.Items.UpsertErrors((List<Error>)([
                new UnauthorizedError("Forbidden access")
            ]));
            return;
        }
        await next.Invoke();
    };
}
```

Slika 5.27 - Ocelot međusloj

U ocelot.json konfiguraciju se za svaki endpoint doda *RouteClaimsRequirement*, a vrijednost je JSON objekat sa *roles* atributom i nizom dozvoljenih uloga, kao što je prikazano u isječku *ocelot.json* fajla.

```
{
  "Routes": [
    {
      "DownstreamPathTemplate": "/image-finder?{everything}",
      "DownstreamScheme": "https",
      "DownstreamHostAndPorts": [
        {
          "Host": "192.168.100.6",
          "Port": 7016
        }
      ],
      "UpstreamPathTemplate": "/api/image-finder?{everything}",
      "UpstreamHttpMethod": [ "Get" ],
      "AuthenticationOptions": {
        "AuthenticationProviderKey": "Bearer"
      },
      "RouteClaimsRequirement": {
        "roles": "[premium_user,content_administrator]"
      }
    },
```

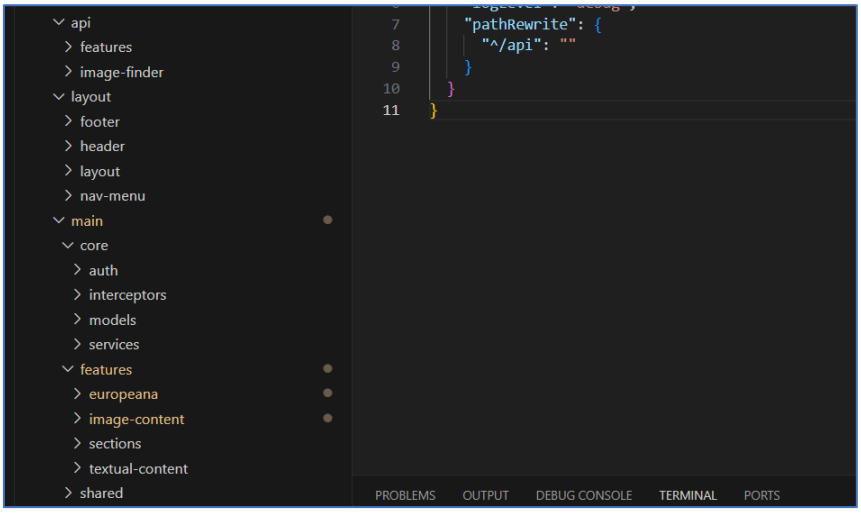
Primjer rada API mrežnog prolaza je prikazan upotrebom Postmana na slici 5.28.



Slika 5.28 - Postman zahtjev ka API Gateway sa JWT tokenom

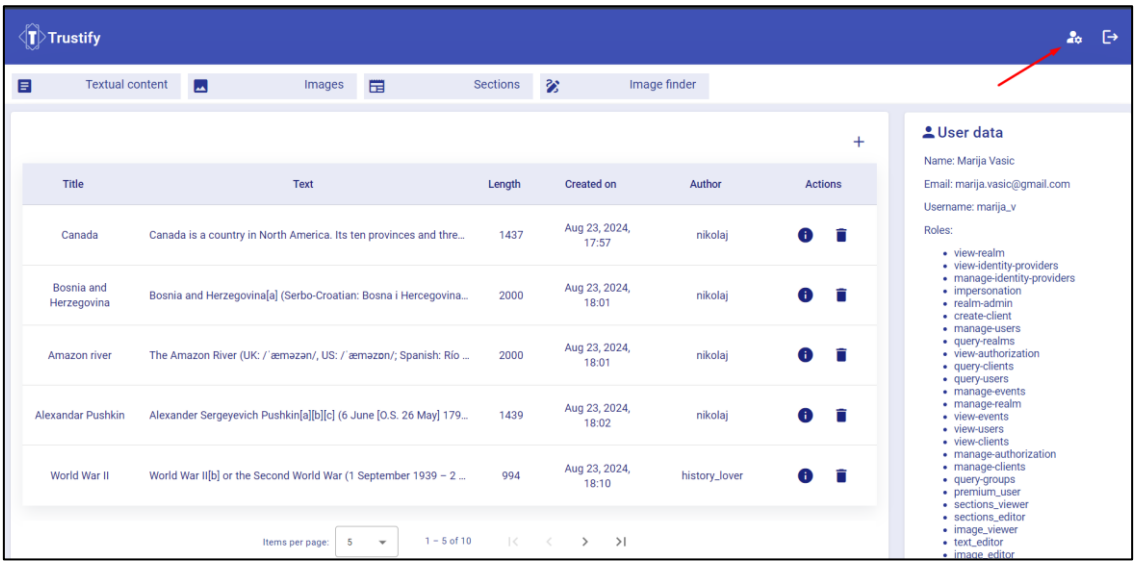
5.4.2 *Front-end aplikacije*

Front-end TrustifyApp aplikacije (Trustify.Frontent) je rađen poštujući iste principe kao i Trustify.Admin aplikacija, pri čemu je korišteno generisanje koda sa back-end-a, korišten je Angular Material i Google ikonice. Kod je organizovan na način prikazan na slici 5.29. Za razliku od Trustify.Admin aplikacije gdje je autentifikacija i autorizacija rađena pozivima prema back-end serveru, Trustify.Frontend aplikacija komunicira direktno sa IP preko *keycloak-angular* paketa[31]. Konfigurisan je i presretač zahtjeva koji u svaki HTTP poziv prema back-end serveru dodaje Bearer token unutar *Authorization* zaglavlja. Kada zahtjev stigne na mrežni prolaz, Ocelot će da provjeri validnost tokena uz pomoć konfigurisane JWT šeme.



Slika 5.29 - Organizacija koda

Kada se pokrene aplikacija, otvara se početna stranica kao na slici 5.30. Prikazana je tabela



Slika 5.30 - Tekstualni sadržaj aplikacije

sa tekstualnim sadržajem. U desnom uglu (crvena strelica) postoji dugme za prelazak u administratorsku aplikaciju. Kada se otvori ekran, korisnicima je prikazano samo ono za šta

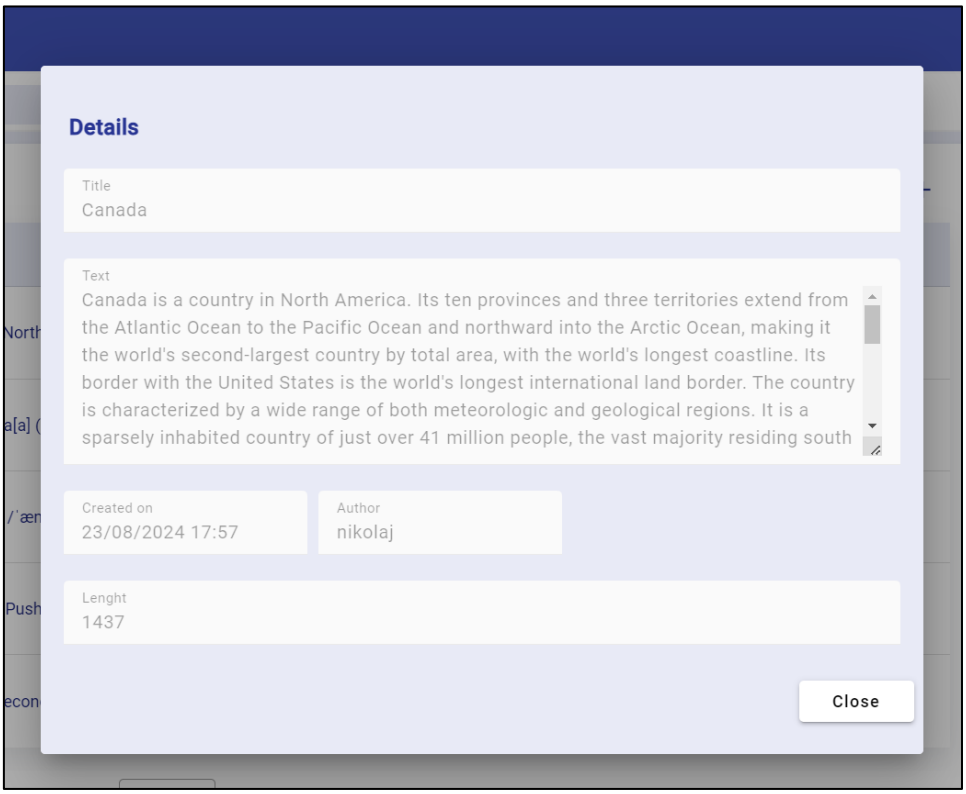
[31] https://www.npmjs.com/package/keycloak-angular

imaju dozvoljene privilegije. Trenutno prijavljeni korisnik ima permisije uloge *premium_user*, odnosno pripada grupi "*premium_users*", koja ima sve privilegije. Tabela 5.1 prikazuje funkcionalnosti i minimalnu potrebnu ulogu za datu funkcionalnost.

Tabela 5.1 - Prikaz funkcionalnosti i uloga

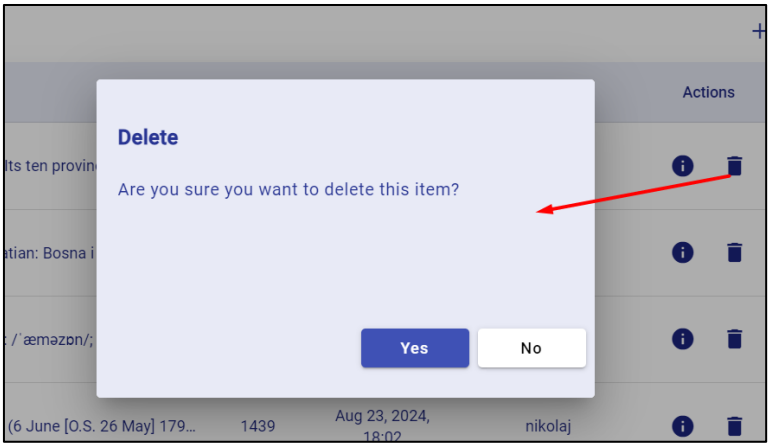
Naziv funkcionalnosti	Podfunkcionalnost	Potrebna uloga	Opis uloge
Tekstualni sadržaj	Pregled	text_viewer	Može da vidi samo tekstualni sadržaj.
	Detalji o selektovanom tekstu	text_viewer	
	Brisanje selektovanog teksta	text_editor	Može da vidi i mijenja samo tekstualni sadržaj.
	Dodavanje novog tekstualnog sadržaja	text_editor	
Sadržaj sa slikama	Pregled	image_viewer	Može da vidi samo slike.
	Brisanje selektovane slike	image_editor	Može da vidi i mijenja samo sadržaj sa slikama.
	Dodavanje nove slike	image_editor	
Sekcije	Prikaz dozvoljenih sekcija	sections_viewer	Može da vidi tekstualni sadržaj, sadržaj sa slikama i sekcije.
	Detalji o sekciji	sections_viewer	
	Brisanje selektovane sekcije	sections_editor	Može da vidi i mijenja tekstualni sadržaj, sadržaj sa slikama i sekcije.
	Dodavanje nove sekcije	sections_editor	
Pretraga slika Europeana servisa	Pronalazak slike po zadanoj riječi	premium_user	Ima sve permisije u aplikaciji.
	Dodavanje slike u sistem	premium_user, content_administrator	Ima sve permisije u aplikaciji. Uloga premium_user se plaća, dok je uloga content_administrator namijenjena korisnicima koji održavaju sajt.

Jedna od funkcionalnosti upravljanja tekstualnim sadržajem jeste prikaz detalja tog sadržaja, što prikazuje slika 5.31.



Slika 5.31 - Detalji selektovanog tekstualnog sadržaja

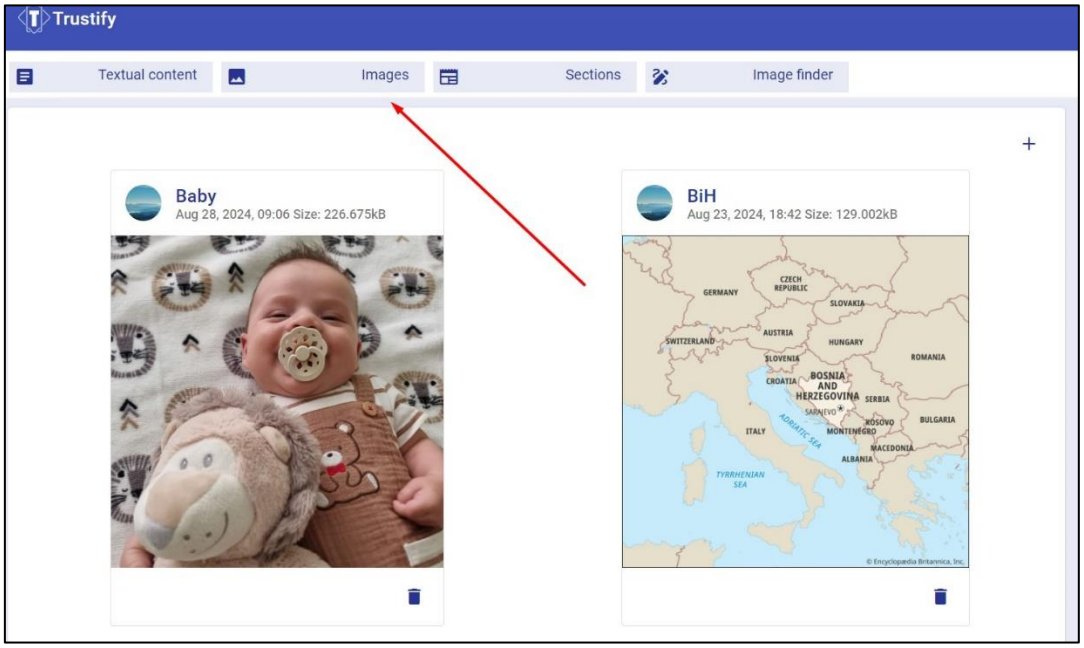
Pored toga moguće je obrisati selektovani sadrža i prije svakog brisanja bilo kog zapisa aplikacije, korisniku se otvara forma za potvrdu ili odustajanje od brisanja (Slika 5.32). Pored stvarnog brisanja korisničkih privatnih podataka, svi podaci se stvarno brišu iz baze kada korisnik to zatraži[32].



Slika 5.32 - Brisanje sadržaja

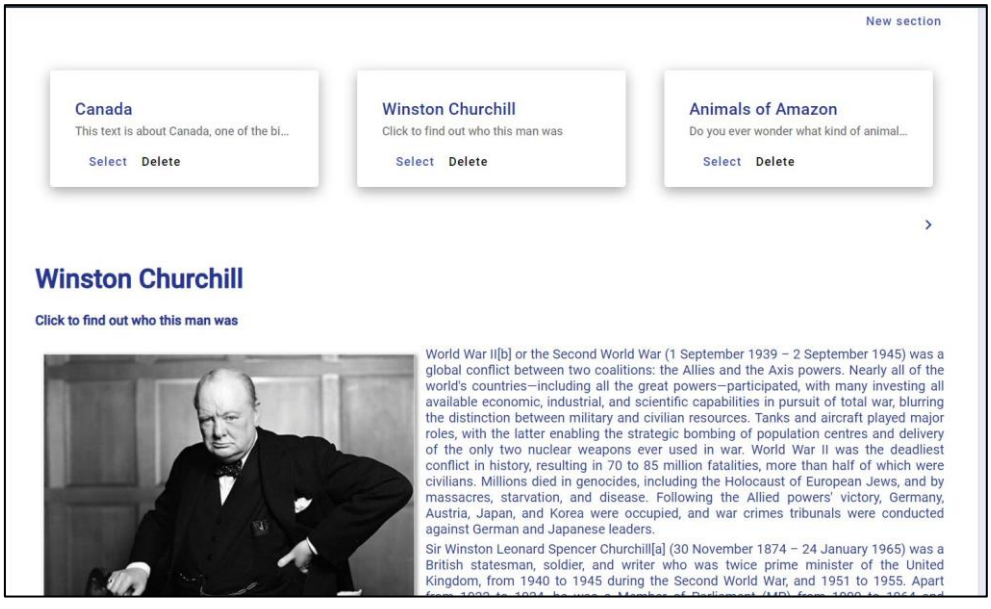
[32] https://gdpr.eu/right-to-be forgotten/#:~:text=In%20Article%2017%2C%20the%20GDPR,originally%20collected%20or%20processed%20it.

Što se tiče prikaza sadržaja sa slikama, kada korisnik klikne na dugme "*Images*" učitava se sadržaj prikazan na slici. Slike je moguće listati, dodati novu sliku i obrisati postojeću (Slika 5.33).



Slika 5.33 - Prikaz sadržaja sa slikama

Cilj sekcija jeste da prikaže vezu više naprema više u bazi podataka, koja se susreće u poslovnim aplikacija. Jedna sekcija ima više slika, pri čemu jedna slika može da pripada više sekcija. Jedna sekcija može da ima više tekstualnog sadržaja, i jedan tekstualni sadržaj može da ima više slika. Takođe, nekada nije poželjno da korisnik dobije poruku o zabranjenom pristupu, već je potrebno i da se prikažu samo one komponente aplikacije kojim korisnik može da pristupi. Korisnik sa ulogom „*sections_viewer*" može da pristupi prikazu sekcija, ali mu se prikazuju samo sekcije koje su dozvoljene njegovoj ulozi prilikom kreiranja sekcije.

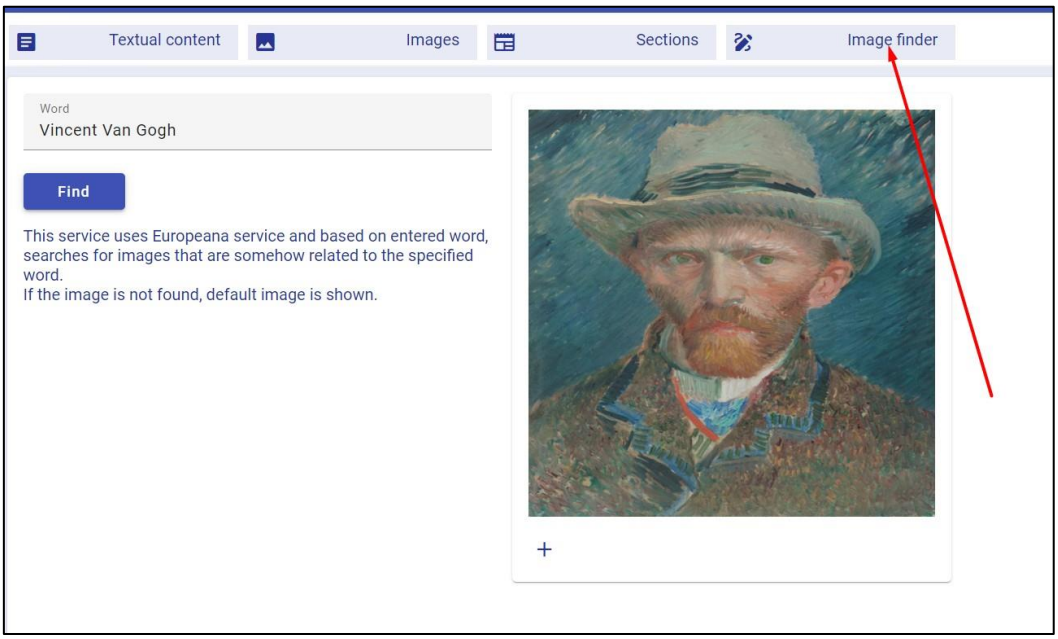


Slika 5.34 – Sekcije

Slika 5.34 prikazuje dio aplikacije sa sekcijama. Na slici je odabrana sekcija Winston Churchill[33]. Klikom na dugme za dodavanje sekcije, otvara se prozor sa formom za dodavanje kao na slici 5.35. Moguće je odabrati neki od 5 proizvoljno prikazanih tekstova i slika, ili dodavanje dodatnih. Takođe je moguće selektovati uloge koje mogu da vide datu sekciju.

Slika 5.35 - Dodavanje sekcije

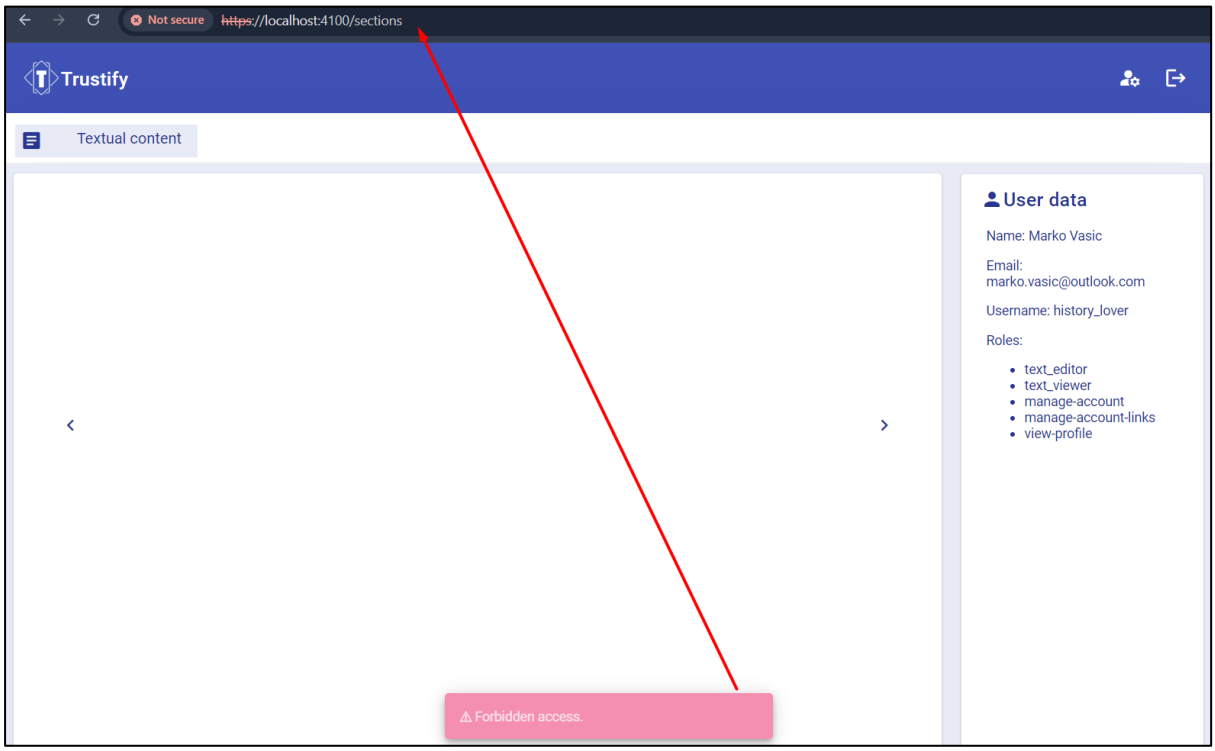
Pored manipulisanja slikama, tekstom i sekcijama, moguća je upotreba pretraživača (generatora) slika klikom na dugme "*Image finder*". Ukoliko u polje za pretragu unesemo tekst „Vincent Van Gogh", dobićemo prvu sliku Europeana baze koja je vezana za datu sintaksu (Slika 5.36).



Slika 5.36 - Pretraga slika iz Europeana baze

[33] Sadržaj je preuzet sa Wikipedije. [36]

Moguće je datu sliku, klikom na dugme "+" dodati kao sadržaj u aplikaciju.
Da bi se korisnik informisao o rezultatu akcije, prikazuje mu se pop-up sa odabranom porukom. Primjer poruke u slučaju zabranjenog pristupa je prikazan na slici 5.37. Iako možemo da sakrijemo komponente koje nisu dozvoljene prijavljenom korisniku, ne možemo da ga zaustavimo u pokušaju direktnog pristupa komponenti ili API-ju preko URL-a. Na slici se vidi da korisnik *history_lover* nema nijednu ulogu koja bi mu dozvolila pristup sekcijama i prilikom pristupanja sekcijama, dobija grešku *Forbidden access* (server je vratio statusni kod 403).



Slika 5.37 - Zabranjen pristup sekcijama

6 Zaključak

Danas je teško zamisliti uspješnu organizaciju koja na neki način ne koristi informacioni sistem da bi unaprijedila način rada, bilo da je riječ o školama, bilo da je riječ o industrijama. Informacioni sistem se razvija imajući u vidu sociološku i tehnološku komponentu, tako da pri razvoju istog ne treba zanemariti buduće korisnike. I hardver i softver se moraju održavati, pa u skladu sa potrebama i mogućnostima, arhitekta sistema treba da konstruiše takvu strukturu aplikacije da sistem opstane što je duže moguće. Kada govorimo o Internet aplikacijama, govorimo o arhitekturama kao što su klijent-server, MVC, višeslojna i servisno-orijentisana arhitektura. Veći sistemi sa više korisnika često zahtijevaju skalabilnost na više uređaja, pa je i mikroservisna arhitektura doživjela svoj procvat.

Prilikom izbora arhitekture, potrebno je odrediti i sigurnosne kontrole i mjere. Sistem treba da se nadzire i osigura na načine koje bi, u idealnom slučaju, odredio stručnjak za oblast sigurnosti na samom početku. Osnovni i najjeftiniji vid nadzora nad aplikacijom jeste vođenje zapisa o aktivnostima unutar aplikacije. Rečeno je da je vođenje zapisa najjeftiniji način nadzora, međutim često nam skuplje mjere koje preduzmemo na početku osiguravaju uštedu novca u kasnijim fazama upotrebe softvera.

Svaki sistem je dizajniran s ciljem da ga koriste ili ljudi ili aplikacije. U domenu poslovnih procesa, govorimo o oba slučaja. Zbog toga se uvodi termin identitet. Kada se korisnik ili sistem registruje, stvara se nalog. Sistemi sa nalozima najčešće podrazumijevaju registraciju, prijavu na sistem, odjavu sa sistema i brisanje naloga. Takođe, rijetki su poslovni procesi gdje bilo ko može da pristupi bilo čemu. Zbog toga se uvodi kontrola pristupa koja može biti zasnovana na atributima, ulogama i pravilima. Naravno, kakav vid kontrole pristupa će da se koristi zavisi od domena primjene aplikacije, pa se tako MAC kontrola pristupa najčešće koristi u vojne svrhe, dok ćemo u komercijalnim aplikacijama najčešće sresti kontrolu pristupa zasnovanu na ulogama. U praktičnom dijelu rada je upravo ta kontrola pristupa opisana. Vođenje sesije može da se reguliše na razne načine, a jedan od njih jeste upotreba Bearer tokena. Format koji omogućuje validaciju bez dodatnog poziva serveru jeste JWT format tokena. Za razliku od standardnog vođenja sesije na back-end-u, Bearer token se čuva na klijentskoj strani, pa je podložniji napadima. Zbog toga je bitno da se podesi kratko vrijeme trajanja tokena, a da se sesija produži upotrebom tokena za obnovu, koji ne nosi važne korisničke informacije.

Kao što je najbolja klasa koja ima jednu odgovornost, tako je i posebno specijalizovan sistem razvijen od strane ljudi posvećenih tom softveru često bolji od sopstvene implementacije nekih rješenja. Jedno od takvih rješenja koje nije preporučljivo da inženjer sam razvija jeste vezan za autentifikaciju i autorizaciju. Postoje specifikacije, definisane i prihvaćene od kruga stručnjaka iz oblasti sigurnosti, koje pomažu u zaštiti aplikacije od neautentifikovanog ili od neautorizovanog pristupa, ili čak od oboje, kao što je OIDC protokol. OIDC protokol je nadgradnja uspješnog OAuth 2.0 protokola. Provajderi identiteta, kao ključne komponente OIDC protokola, se danas uglavnom nude kao gotova rješenja. Keycloak je besplatan alat, opisan i podešen u ovom radu, koji omogućuje inženjerima da kreiraju oblasti, klijente, nove korisnike i uloge ili atribute, kroz interfejs, ne ulazeći u pozadinu kako je odabrani protokol implementiran. Time se smanjuje mogućnost propusta i ubrzava vrijeme razvoja aplikacije. Kako su menadžeri ti koji upravljaju korisnicima i ulogama, Keycloak interfejs može da dovede do propusta koje bi željeli da spriječimo, kao što je brisanje oblasti. U radu su opisane smjernice kojih treba da se pridržavamo prilikom kreiranja korisničkog interfejsa, pa je prilikom implementacije sopstvenog Keycloak administratorskog panela poželjno pratiti te smjernice. Prikladne tehnologije za razvoj takvog panela su Angular i .NET, koji nudi mogućnost implementiranja sigurnosti uz pomoć međuslojeva i šema i koji nudi jednostavan

razvoj RESTful API-ja. Danas postoje pouzdane biblioteke koje olakšavaju razvoj softvera u .NET i praćenje dobrih praksi i principa, kao što su AutoMapper, Microsoft Dependency Injection i Serilog. Mikroservisi mogu da se nalaze iza API mrežnog prolaza, a u .NET je podržan Ocelot API mrežni prolaz. U jednostavnoj JSON konfiguraciji je podešeno rutiranje ka mikroservisima, i podešena su pravila načina autentifikovanja korisnika i verifikovanja permisija. Angular takođe nudi biblioteku koja olakšava rad sa Keycloak provajderom identiteta, ukoliko se čitava kontrola pristupa ne odvija na back-end-u. Uz pomoć datih tehnologija su implementirana dva rješenja kao demonstracija principa iz oblasti arhitekture softvera i sigurnosti.

Današnja najkorištenija tehnologija u oblasti računarstva, Internet, je razvijena zbog vojnih potreba. Dvije strane pokušavaju da nadmudre jedna drugu prilikom slanja povjerljivih informacija. Danas, kako se Internet koristi u komercijalne svrhe i kako je postao oruđe razvoja poslovanja, ne treba zaboraviti da se podaci i dalje šalju i da postoje zlonamjerne strane koje pokušavaju da nađu način da se domognu koristi ili nanesu štetu. Sigurnost je razvijena i treba je koristiti. Ljudi ulažu povjerenje u informacione sisteme koje upotrebljavaju i njihova privatnost treba da je većeg prioriteta od brzog i jeftinog razvoja. Čak i uz jednostavno podržavanje HTTPS protokola, već smo na korak ka svijesti o značaju skrivanja informacija.

Softver treba da se djelimično posmatra kao priča, svaka metoda treba da ima namjenu, svaka funkcionalnost svoju svrhu. Međutim, priči dođe kraj, a softver treba da ima samo početak, treba da mijenja oblik informacija i obrade informacija tako da imaju značaj u poslovnom procesu dok god taj poslovni proces ima smisla. Pravilna implementacija provjerenih arhitektura daje čvrst temelj takvom sistemu. Jedna od priča koja neće izumrijeti i koja nema kraj, pored dobrog softvera, jeste priča o sigurnosti na Internetu.

LITERATURA

[1] R. T. Watson, Information Systems, 2007.

[2] L. Carter, „10 Biggest Cyber Attacks in History,“ [Na mreži]. Available: https://clearinsurance.com.au/10-biggest-cyber-attacks-in-history/.

[3] S. Newman, Building Microservices, 2021.

[4] G. Dhillon, Information Security Management: Global Challenges in the New Millennium, 2007.

[5] R. C. Martin, Clean Code, 2008.

[6] M. Fowler, Patterns of Enterprise Application Architecture, 2011.

[7] R. K. P. C. L. Brass, Software Architecture in Practice, 2013.

[8] B. B. K. S. E. R. E. Freeman, Head First Design Patterns, 2004.

[9] R. C. Martin, Clean Architecture: A Craftman’s Guide to Software Structure and Design, 2017.

[10] B. Golden, „4 ways to exploit microservice architecture for better app sec,“ [Na mreži]. Available: https://techbeacon.com/app-dev-testing/4-ways-exploit-microservices-architecture-better-app-sec. [Poslednji pristup 1 11 2023].

[11] W. Williams, Security for Service Oriented Architectures, 2014.

[12] T. Erl, SOA Principles of Software Design, 2007.

[13] C. Phan, Service Oriented Architecture (SOA) - Security Challenges and Mitigation Strategies, MILCOM 2007 - IEEE Military Communications Conference, 2007.

[14] W. Stallings, Network Security Essentials: Applications and Standards, 2010.

[15] M. P. Dafydd Stuttard, Web Application Hacker's Handbook: Finding and Exploiting Security Flaws.

[16] R. K. R. C. David Ferraiolo, Role-Based Access Control, 2nd ur., 2007.

[17] D. F. R. K. A. S. Vincent C. Hu, Attribute Based Access Control, National Institute of Standards and Technology, 2014.

[18] N. Shavit, „RBAC vs ABAC: pros, cons, and example policies,“ [Na mreži]. Available: https://www.aserto.com/blog/rbac-vs-abac-authorization-models. [Poslednji pristup 16 11 2023].

[19] „When to use a rule based access control,“ [Na mreži]. Available: https://www.avigilon.com/blog/access-control-models#when-to-use-a-rule-based-access-control-model. [Poslednji pristup 26 11 2023].

[20] T. M. Jonathan LeBlanc, Identity and Data Security for Web Development: Best Practices, 2016.

[21] N. P. Axel Buecker, Enterprise Single Sign-On Design Guide, 2012.

[22] A. H. Yvonne Wilson, Solving Identity Management in Modern Applications: Demystifying OAuth 2.0, OpenID Connect and SAML 2.0, 2019.

[23] „GDPR,“ 2016. [Na mreži]. Available: https://gdpr-info.eu/art-2-gdpr/. [Poslednji pristup 28 11 2023].

[24] A. S. Justin Richer, OAuth 2 In Action, 2017.

[25] O. Foundation, „OpenID Authentication 2.0 - Final,“ 5 12 2007. [Na mreži]. Available: https://openid.net/specs/openid-authentication-2_0.html. [Poslednji pristup 28 1 2024].

[26] M. O. Bart Delft, *A Security Analysis of OpenID,* Oslo, 2010.

[27] D. Hardt, „The OAuth 2.0 Authorization Framework,“ Internet Engineering Task Force (IETF), 10 2012. [Na mreži]. Available: https://datatracker.ietf.org/doc/html/rfc6749. [Poslednji pristup 1 2 2024].

[28] D. H. M. Jones, „The OAuth 2.0 Authorization Framework: Bearer Token Usage,“ 10 2012. [Na mreži]. Available: https://datatracker.ietf.org/doc/html/rfc6750. [Poslednji pristup 1 2 2024].

[29] E. E. Hammer-Lahav, „HTTP Authentication: MAC Access,“ 2 2012. [Na mreži]. Available: https://www.ietf.org/archive/id/draft-ietf-oauth-v2-http-mac-00.html#rfc.section.1. [Poslednji pristup 12 2 2024].

[30] „How OpenID Connect Works,“ [Na mreži]. Available: https://openid.net/developers/how-connect-works/. [Poslednji pristup 20 2 2024].

[31] „Security Assertion Markup Language (SAML) V2.0 Technical Overview,“ OASIS Security Services Technical Committee, 25 3 2008. [Na mreži]. Available: https://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html. [Poslednji pristup 22 2 2024].

[32] JBoss, „Keycloak,“ [Na mreži]. Available: https://www.keycloak.org/. [Poslednji pristup 26 8 2024].

[33] I. Kent, „MainHog,“ [Na mreži]. Available: https://github.com/mailhog/MailHog. [Poslednji pristup 27 8 2024].

[34] T. O. team, „Ocelot,“ [Na mreži]. Available: https://ocelot.readthedocs.io/en/latest/introduction/gettingstarted.html. [Poslednji pristup 27 8 2024].

[35] J. B. M. J. N. Sakimura, „OpenID Connect Core 1.0,“ 15 12 2023. [Na mreži]. Available: https://openid.net/specs/openid-connect-core-1_0.html. [Poslednji pristup 20 2 2024].

[36] „Wikipedia,“ [Na mreži]. Available: https://www.wikipedia.org/. [Poslednji pristup 28 8 2024].