Utjecaj sigurnosti i privatnosti na arhitekturu i dizajn web aplikacija

Sakač, Martin

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Martin Sakač

Utjecaj sigurnosti i privatnosti na arhitekturu i dizajn web aplikacija

DIPLOMSKI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Martin Sakač

Matični broj: 49585

Studij: Informacijsko i programsko inženjerstvo

Utjecaj sigurnosti i privatnosti na arhitekturu i dizajn web aplikacija

DIPLOMSKI RAD

Mentor:

doc. dr. sc. Matija Novak

Varaždin, rujan 2023.

Martin Sakač

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Tema ovog rada je utjecaj sigurnosti i privatnosti na arhitekturu i dizajn web aplikacija. Cilj rada je prikazati kako zaštiti osjetljive podatke i osigurati korisničku privatnost i sigurnost u digitalnom okruženju. U teorijskom dijelu, opisane su slabosti, rizici kao i napadi na web aplikacije. Objašnjeni su osobni podaci te važnost očuvanja integriteta i tajnosti takvih podataka.

Praktični dio rada obuhvaća primjenu sigurnosti i privatnosti na stvarnoj web aplikaciji. Web aplikacije razvijena je pomoću tehnologija React JS za aplikaciju klijenta i Spring Boot za aplikaciju poslužitelja. U praktičnom dijelu rada implementirani su neki od ključnih koncepata i sigurnosti i privatnosti.

U radu se također analiziraju najbolje prakse, alati i tehnike koji se koriste za obranu sigurnosnih prijetnji. Definiraju se ciljevi sigurnosti i privatnosti te uzorci sigurnosti i privatnosti koji mogu doprinjeti očuvanju integriteta, povjerljivosti i dostupnosti podataka.

Rad nastoji pružiti uvid u složenost i izazove razvoja sigurnih i privatnih web aplikacija, kao i naglasiti važnost istih u digitalnom dobu gdje se podaci i privatnost korisnika često dovode u pitanje.

Ključne riječi: sigurnost, privatnost, podaci, web aplikacija, autentifikacija, Java, React

Sadržaj

1. Uvod ... 6
2. Sigurnost web aplikacija ... 8
   2.1. Općenito o sigurnosti ... 8
      2.1.1. Odnos IT sigurnosti sa računalnom i informacijskom sigurnosti ... 9
   2.2. Slabosti i rizici loše arhitekture web aplikacije ... 10
      2.2.1. WASC ... 11
      2.2.2. OWASP ... 12
      2.1.1. Slabosti i rizici web aplikacija ... 13
   2.2. Napadi na web aplikacije ... 16
      2.3.1. Napad skriptama (XSS) ... 16
      2.3.2. Ubrizgavanje ... 19
      2.3.3. Napad na autentifikaciju ... 20
      2.3.4. CSRF ... 22
   2.3. Ublažavanje sigurnosnih rizika web aplikacije ... 23
3. Privatnost osobnih podataka ... 25
   3.1. Osobni podaci i njihova obrada ... 25
   3.2. Zaštita osobnih podataka ... 26
   3.3. AZOP i GDPR ... 26
   3.4. Najbolje prakse privatnosti web aplikacija ... 27
4. Utjecaj sigurnosti i privatnosti na dizajn i arhitekturu web aplikacija ... 29
   4.1. Definiranje ciljeva sigurnosti i privatnosti ... 29
   4.2. Odabir odgovarajuće programske arhitekture ... 30
      4.2.1. Monolitna arhitektura ... 30
      4.2.2. Mikroservisna arhitektura ... 31
      4.2.3. Arhitektura bez poslužitelja ... 31
      4.2.4. Servisno orijentirana arhitektura ... 32
   4.3. Primjena uzoraka sigurnosti i privatnosti ... 33
   4.4. Procjena i prilagođavanje arhitekture ... 34
   4.5. Zaštita od napada na web aplikacija ... 35
      4.5.1. Zaštita od XSS napada ... 35
      4.5.2. Zaštita od ubrizgavanja ... 35
      4.5.3. Zaštita autentifikacije ... 36
      4.5.4. Zaštita od CSRF napada ... 37
5. Uzorci dizajna sigurnosti i privatnosti ... 38
   5.1. Uzorci sigurnosti ... 38

5.1.1. Single Access Point 39
5.1.2. Secure Logger 41
5.1.3. Secure Adapter 42
5.1.4. Input Validation 44
5.1.5. Session-Based Role-Based Access Control 45
5.2. Uzorci privatnosti 46
5.2.1. Prikaz politike privatnosti 47
5.2.2. Tablica osobnih podataka 47
5.2.3. Nadzorna ploča privatnosti 48
6. Izrada web aplikacije 50
6.1. Opis web aplikacije 50
6.2. ERA model 52
6.3. Arhitektura web aplikacije 54
6.4. Korištene tehnologije 55
6.5. Razvoj web aplikacije 56
6.5.1. Implementacija na strani poslužitelja 56
6.5.1.1. Model 58
6.5.1.2. Kontroler 58
6.5.1.3. Servis 60
6.5.1.4. Maper 61
6.5.1.5. Repozitorij 61
6.5.2. Implementacija na strani klijenta 62
6.6. Sigurnost web aplikacije 64
6.6.1. Alati za kvalitetu i dosljednost programskog koda 64
6.6.2. Autentifikacija 65
6.6.2.1. Registracija 67
6.6.2.2. Ponovno postavljanje lozinke 68
6.6.2.3. Autentifikacija putem OAuth2 68
6.6.3. Autorizacija 70
6.6.4. DTO 72
6.6.5. SQL ubrizgavanje 73
6.6.6. Dnevnik rada 73
6.6.7. Validacija korisničkog unosa 76
6.7. Privatnost web aplikacije 76
6.7.1. Politika privatnosti 77
6.7.2. Brisanje korisničkih podataka 78

6.8. Izazovi i problemi u razvoju aplikacije 79
7. Zaključak 82
Popis literature 83
Popis slika 87
Popis tablica 88
Prilozi 89

1. Uvod

Tehnologija i digitalna komunikacija danas su glavni pokretači društvenih, ekonomskih i kulturnih promjena. Web i aplikacije temeljene na webu postale su neizbježan dio naših svakodnevnih života. Korisnici provode velike količine svojeg vremena koristeći web za komunikaciju, traženje informacija, upravljanje financijama, kupnju proizvoda i usluge te još mnogo toga. Unatoč brojnim prednostima koje web aplikacija nude, sa sobom donose izazove i prijetnje koje mogu ugroziti sigurnost i privatnost krajnjih korisnika. Svjedoci smo sve većeg broja napada na informacijske sustave, krađe i curenja osobnih podataka, neovlaštenog pristupanja sustavima te prouzročavanja značajnih financijskih troškova. Napadi na sigurnost i privatnost sustava i informacija postali su ozbiljna prijetnja, kako organizacijama, tako i pojedincima.

U ovom radu istraženi su i praktično prikazani ključni aspekti sigurnosti i privatnosti u web aplikacijama te smjernice i najbolje prakse koje mogu pomoći organizacijama da postignu visoke razine zaštite podataka te time osiguraju povjerenje i integritet korisnika. Prikazane su različite metode zaštite, alati i tehnike koji se koriste za zaštitu od napada i sigurnosnih prijetnji, a u cilju osiguranja integriteta, povjerljivosti i dostupnosti podataka u web okruženju.

U prvom i drugom poglavlju, detaljno će biti razrađeni pojmovi sigurnosti web aplikacije te privatnost osobnih podataka. Prikazani će biti slabosti i rizici loše arhitekture web aplikacije te probleme koje takva arhitektura sa sobom nosi. Obrađeni su neki od čestih napada na web aplikacije te dane smjernice sa kojima se može ublažiti sigurnosni rizik web aplikacija. Poglavlje privatnosti osobnih podataka definirat će što su to osobni podaci, kakve vrste postoje te na koji način se obrađuju. Zatim će biti navedene najbolje prakse privatnosti i metode zaštite osobnih podataka te regulatorna tijela koja nadgledaju provođenje zakona o zaštiti istih.

Utjecaj sigurnosti i privatnosti na dizajn i arhitekturu igra veliku ulogu tokom izgradnje web aplikacije te je važno od samog početa uzeti iste u obzir. U četvrtom poglavlju definirani su utjecaji navedenih faktora na arhitekturu web aplikacije. Navedene su vrste arhitekture te njihov utjecaj na sigurnost web aplikacije. Isto ako prikazani su mehanizmi i načini implemetacije koji osiguravaju zaštitu od čestih napada na web aplikacije.

U petom poglavlju objašnjen je termin uzoraka dizajna, uzoraka sigurnosti te uzoraka privatnosti. Postoji velik broj uzoraka koji se može primijeniti na sigurnost ili privatnost web aplikacije. Nekoliko uzoraka sigurnosti i privatnost je detaljnije razređeno te njhova implementacija, posljedice i sudionici.

Kako bi se prikazali praktični aspekti sigurnosti i privatnosti, u šestom poglavlju temeljito je opisana web aplikacija, njezini podatkovni modeli i arhitektura. Prikazane su sigurnosne metode i tehnike koje su implementirane kako bi se zaštitili korisnici i njihovi podaci. Također su navedeni uzorci privatnosti koji su osmišljeni kako bi se osigurala integritet i povjerljivost korisničkih podataka. Tijekom razvoja aplikacije suočavate se s brojnim izazovima i problemima, a neki od njih su također detaljno razmotreni.

Konačno, zaključak donosi sažetak ključnih spoznaja i naglašava važnost sigurnosti i privatnosti u web aplikacijama, ističući njihovu važnu ulogu u očuvanju povjerenja korisnika i integriteta podataka.

2. Sigurnost web aplikacija

Sigurnost je jedan od ključnih faktora i prioriteta web aplikacije koji tokom cijelog životnog ciklusa igra važnu ulogu. U ranim danima weba, većina ljudi nije imala pristup internetu, što je rezultiralo manjim naglaskom na pitanje sigurnosti. Međutim, s masovnim usvajanjem weba, sigurnost je postala važni aspekta weba. Danas je od ključne važnosti rješavati probleme sigurnosti kako bismo osigurali povjerenje u online okruženje te zaštitili sustave, korisnike i njihove podatke od neovlaštenog pristupa ili zloupotrebe. Veliki broj poslovanja događa se putem različitih aplikacija i servisa na webu. Ljudi danas više vremena provode listajući kroz razne proizvode i usluge na ekranu nego u trgovinama, sve više poslova provodi se na računalima koristeći milijune različitih web aplikacija. Većina korisnika nije svjesna napora koji se ulažu kako bi se osigurala zaštita njihovih podataka i računa na webu. Svakodnevno se šire vijesti o povredi i iskorištavanju podataka, krađi identiteta, bankovnih kartica i novca putem weba. Svaki podatak koji je predan nekoj web stranici dovodi se u rizik od krađe, zloupotrebe ili neke druge maliciozne radnje. Kako bi se spriječili takvi neželjeni incidenti, proizvođači web aplikacija kontinuirano implementiraju sigurnosne procedure na svim razinama svoje aplikacije. Sigurnost je danas postala najvažniji faktor na webu te se zbog toga mnoge organizacije odlučuju na razvoj sigurnih projekata već od samog početka. Razvoj web aplikacije s sigurnom arhitekturom, korištenjem najboljih praksi za pisanje koda te uvođenjem vanjskih sustava koji su prošli rigorozne sigurnosne testove, u značajnoj mjeri ublažuje rizik od sigurnosnih propusta i proboja napadača.

U ovom poglavlju, detaljnije će se razraditi pojam sigurnosti i IT (eng. Information Technology) sigurnosti te usporediti IT sigurnost s računalnom i informacijskom sigurnošću. Loša arhitektura web aplikacije sa sobom donosi slabosti i rizike. Navedene će biti organizacije koje se profesionalno bave sigurnošću web aplikacije te slabosti i rizici na koje one stavljaju pozornost. Zatim će se detaljnije objasniti neke vrste napada na web aplikacije te kako se može ublažiti sigurnosni rizik web aplikacija.

2.1. Općenito o sigurnosti

Kada govorimo o sigurnosti, jasno je da se radi o sveprisutnom pojmu i jednom od ključnih aspekata naših života. Sigurnost je neizostavan pojam naše svakodnevice te se može primijeniti na skoro svaki objekt i radnju koja nas okružuje. Sve što gradimo, koristimo, jedemo i pijemo, želimo da bude sigurno za naše fizičko i psihičko zdravlje, za sve oko nas pa i za okoliš. Isto tako, svaki čovjek teži nekoj sigurnosti, od onih najosnovnijih, odnosno najvažnijih sigurnosti, kao što su prehrambena i zdravstvena sigurnost, pa do ekonomske, političke i osobne sigurnosti.

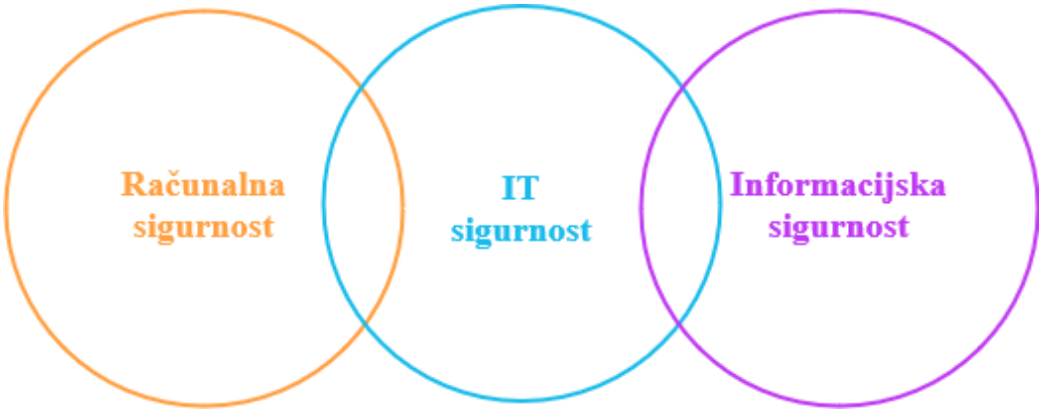
Kada pokušamo definirati sigurnost, dolazimo do problema da to baš i nije jednostavno sročiti u jednu sveobuhvatnu definiciju. Sigurnost je višedimenzionalna i vrlo raznolika u različitim praksama, te se zbog toga ne može svrstati pod samo jednu definiciju, jer ona ovisi o kontekstu u kojem se sigurnost primjenjuje. Mogli bismo reći da sigurnost predstavlja stabilno i predvidljivo okruženje u kojem pojedinac ili grupa može slijediti svoje ciljeve bez ometanja i straha. Sama definicija sigurnosti ima različito značenje za različite ljude s obzirom na vrijeme, mjesto i kontekst. [1]

Fokus ovog rada će biti na sigurnosti web aplikacija, odnosno aplikacija koje su javno dostupne na internetu. Sigurnost web aplikacija je jedan od aspekata informacijske sigurnosti (u nastavku IT sigurnost), koji se često naziva i kibernetička sigurnost. IT sigurnost obuhvaća skup strategija zaštite od neovlaštenog pristupa digitalnim imovinama, kao što su računala, mreže i informacije dostupne na njima. Osim sigurnosti aplikacija, u IT sigurnost spadaju i sljedeći aspekti sigurnosti [2], [3]:

- Sigurnost mreža (eng. Network Security)
- Sigurnost u oblaku (eng. Cloud Security),
- Sigurnost krajnjih točaka (eng. Endpoint Security),
- Mobilna sigurnost
- IoT sigurnost (eng. Internet of Things security)

2.1.1. Odnos IT sigurnosti sa računalnom i informacijskom sigurnosti

Kada govorimo o IT sigurnosti, ključno je shvatiti da je ona vrlo blisko povezana s informacijskom sigurnošću i računalnom sigurnošću, ali postoje ključne razlike između njih. U nastavku je prikazan odnos između navedenih sigurnosti, te su dalje objašnjene razlike IT sigurnosti u odnosu na računalnu i informacijsku sigurnost.



Slika 1: Odnos između računalne, informacijske i IT sigurnosti (Izvor: vlastita izrada, prema: [4])

Informacijska sigurnost se može definirati kao očuvanje povjerljivosti, integriteta i dostupnosti informacija, bez obzira na medij ili njihovu povezanost. IT sigurnost je podskup informacijske sigurnosti, pri čemu je cilj i dalje očuvanje povjerljivosti, integriteta i dostupnosti informacija, ali u kibernetičkom prostoru koji može obuhvaćati internet, digitalne mreže ili povezane uređaje. Osim toga, IT sigurnost se ne bavi samo zaštitom informacija, već i informacijskom infrastrukturom koja je dostupna putem kibernetičkog prostora, te služi kao komunikacijski kanal za prijenos samih informacija. Povijesno gledano, informacijska sigurnost postoji već preko stotinu godina, dok je IT sigurnost relativno novi pojam koji je uveden prije nekoliko desetljeća s razvojem informacijskih tehnologija. Ukratko, IT sigurnost je usmjerena na zaštitu informacija fizičkih subjekata koji su povezani putem interneta i same informacijske infrastrukture, dok informacijska sigurnost ne naglašava povezanost i medije. [4]

Računalna sigurnost usmjerena je na osiguranje pojedinačnih računalnih sustava, softvera i podataka koji se obrađuju i pohranjuju unutar tih sustava. Njen fokus je zaštita računalnih sustava i komponenti od neovlaštenih pristupa, zlonamjernog softvera, računalnih prijetnji i fizičke krađe sustava. Međutim, računalna sigurnost ne uzima u obzir šire okruženje izvan tih sustava. S druge strane, IT sigurnost ima širi opseg jer obuhvaća zaštitu ne samo sustava, već i informacija, mreža na internetu i digitalnih mreža. Naglašava zaštitu od prijetnji i rizika koji su specifični za kibernetički prostor, kao što su kibernetički napadi, povrede podataka i neovlašteni pristup kibernetičkom prostoru. Dakle, dok je računalna sigurnost usmjerena na zaštitu računalnih sustava, softvera i podataka, IT sigurnost ima širi fokus koji obuhvaća zaštitu informacija, mreža i kibernetičkog prostora. [4]

2.2. Slabosti i rizici loše arhitekture web aplikacije

Arhitektura web aplikacije je jedan od najvažnijih aspekata razvoja web aplikacije prije nego što se uopće krene s izradom aplikacije. Arhitekturu web aplikacije je vrlo lako zanemariti i staviti sa strane te odmah krenuti s razvojem aplikacije. U većini slučajeva, razmišljanje o samoj arhitekturi može se činiti kao gubitak dragocjenog vremena koje bi se moglo utrošiti na razvoj i testiranje web aplikacije. No s vremenom, razvoj aplikacije postaje sve kompleksniji. Klijent može zaželjeti dodatnu funkcionalnost koja nije bila predviđena u početnom dogovoru, ili se može pojaviti greška koju će biti veoma teško pronaći. Tada postaje očito da nedostatak pažnje prema arhitekturi web aplikacije predstavlja ozbiljan problem.

Aplikacije s lošom arhitekturom postaju vrlo teške za održavanje, te u nekim slučajevima čak i najmanje promjene mogu dovesti do propadanja cjelokupne funkcionalnosti. U takvim situacijama, dodavanje novih funkcionalnosti postaje gotovo nemoguće. Također, važno je istaknuti probleme s skaliranjem. Ako je arhitektura web aplikacije loše dizajnirana, skaliranje

aplikacije na različite lokacije ili platforme postaje izuzetno teško, bez potrebe za dupliciranjem cijele aplikacije ili pisanjem nove u drugom jeziku. Osim navedenih problema, najveći rizik loše arhitekture je sigurnost web aplikacije. [5, 6]

Sigurnost web aplikacije i arhitektura web aplikacije idu ruku pod ruku. Loša arhitektura web aplikacije može uzrokovati kolaps cjelokupnih sigurnosnih sustava koje web aplikacija koristi. Slabosti koje često dovode do sigurnosnih problema i propusta u web aplikacijama uglavnom su nedostaci ili slabosti u cijelom sustavu i vanjskim sustavima koje ta aplikacija koristi. Primjeri takvih slabosti uključuju lošu konfiguraciju postavki web poslužitelja, neispravno obrađivanje unesenih podataka u HTML (eng. Hypertext Markup Language) elementima (npr. nedostatak saniranja sadržaja) te programski propusti unutar samog koda aplikacije.Takve slabosti u arhitekturi mogu dovesti do ozbiljnih rizika za samu web aplikaciju i krajnje korisnike. Rizik se pojavljuje kada hakeri pokušaju probiti sigurnosne sustave web aplikacije. Cilj kriminalnih hakera (tzv. Black Hat Hackeri) može biti dobivanje neovlaštenog pristupa sustavu ili web aplikaciji, s krajnjim ciljem oštećenja same infrastrukture sustava ili web aplikacije te krađe korisničkih podataka. Rizici s kojima se takve osobe pokušavaju probiti kroz sigurnosne sustave web aplikacije uključuju: [7]

- Iskorištavanje loše konfiguriranih i nesigurnih poslužitelja;
- Iskorištavanje ranjivosti u sustavu i samom aplikacijskom kodu;
- Dobivanjem privatnih podataka korisnika putem socijalnog inženjeringa.

Arhitektura web aplikacija ima direktan utjecaj na sigurnost web aplikacije. Bez dobre arhitekture web aplikacija, a samim tim i sigurnosti, pojavljuje se mnogo rizika koji se mogu iskoristiti za oštećivanje različitih dijelova aplikacijskog okruženja. Razne web aplikacije svakodnevno su žrtve različitih napada, kojima napadači žele pronaći propuste i iskoristiti ih u svoju korist. Zbog toga su se razvile razne organizacije koje sakupljaju podatke od stručnjaka kako bi sastavile popise ranjivosti web aplikacija ili sigurnosne standarde za poboljšanje sigurnosti aplikacija na webu. Neke od takvih organizacija su Otvoreni svjetski projekt sigurnosti web aplikacija (eng. OWASP - Open Web Application Security Project) i konzorcij za sigurnost web aplikacija (eng. WASC - Web Application Security Consortium). Ove organizacije igraju važnu ulogu u podizanju svijesti o sigurnosti web aplikacija te pružaju smjernice i preporuke za poboljšanje sigurnosti. Njihovi resursi i smjernice mogu biti korisni za razvoj i održavanje sigurnih web aplikacija. U nastavku će biti više riječi o sigurnosnim organizacijama.

2.2.1. WASC

Konzorcij za sigurnost web aplikacija odnosno WASC je neprofitna organizacija sastavljena od međunarodnih grupi stručnjaka iz industrije koji proizvode standarde sigurnosti otvorenog

koda i najbolje prakse za izradu web aplikacija. WASC je organizirao industrijske projekte te je objavljuvao tehničke informacije, članke te sigurnosne smjernice koje se i danas koriste za izgradnju aplikacije u cilju rješavanja sigurnosti web aplikacija. [8] Iako je WASC igrao veliku ulogu u prošlosti u promicanje najboljih praksi za sigurnost web aplikacija, njihova aktivnost se u posljednjim godinama uveliko smanjila te više ne objavljuju novije relevantne članke i smjernice koje bi dodatno osigurale sigurnost razvoja web aplikacija sa novim tehnologijama koje se iz dana u dan mijenjaju.

Konzorcij za sigurnost web aplikacija, poznat i kao WASC, je neprofitna organizacija koja okuplja međunarodne grupe stručnjaka iz industrije. Njihova svrha je stvaranje sigurnosnih standarda otvorenog koda i najboljih praksi za razvoj web aplikacija. WASC je organizirao industrijske projekte te je objavljivao tehničke informacije, članke i sigurnosne smjernice koje su se koristile za izgradnju aplikacija s ciljem poboljšanja sigurnosti web aplikacija. [8] Unatoč tome što je WASC igrao važnu ulogu u prošlosti u promicanju najboljih praksi za sigurnost web aplikacija, njihove aktivnosti su se u posljednjim godinama znatno smanjile. Više ne objavljuju nove relevantne članke i smjernice koje bi podržale siguran razvoj web aplikacija s novim tehnologijama koje se konstantno mijenjaju.

2.2.2. OWASP

Otvoreni sigurnosni projekt web aplikacija, odnosno OWASP, je također neprofitna organizacija koja radi na poboljšanju sigurnosti softvera. Sastoji se od otvorenih projekata koje vodi zajednica, desetke tisuća članova te vodećih industrijskih edukacija i konferencija o sigurnosti. Svi alati, dokumenti i projekti koje OWASP zajednica razvija su besplatni i otvoreni za korištenje svima u cilju poboljšanja sigurnosti aplikacija. Sama misija OWASP-a je da više ne bude nesigurnog softvera. Tu misiju žele postići kao najveća svjetska neprofitna organizacija koja se bavi sigurnošću, podržavajući izgradnju utjecajnih projekata, razvijajući događaje, konferencije i sastanke diljem svijeta te nudeći resurse za korištenje javnosti. [9]

OWASP zajednica je tijekom svog vijeka razvila poprilično velik broj projekata. Neki od poznatijih projekata su [10, 11, 12]:

- OWASP ZAP (eng. Zed Attack Proxy) – alat otvorenog koda za penetracijsko testiranje web aplikacije. Alat je veoma proširiv i fleksibilan. Sa alatom je veoma jednostavno pronaći sigurnosne propuste na bilo kojoj web aplikaciji tijekom razvoja i testiranja;
- SAMM (eng. Software Assurance Maturity Model) – okvir koji podržava cijeli životni ciklus softvera, neovisno o tehnologiji i procesu. Pomaže organizaciji da formulira i implementira strategiju za sigurnost softvera;

- Dependecy track – služi za praćenje korištenih biblioteka, okvira, operativnih sustava, firmvera i hardvera i svih ostalih resursa koji se koriste u nekom softveru. Identificira i smanjuje ranjive komponente koje se koriste u softveru.

Osim navedenih projekata, OWASP je također razvio projekt OWASP 10 (eng. OWASP Top Ten). To je dokument za podizanje svijesti za razvojne programere u smislu sigurnosti web aplikacije. OWASP 10 dokumentira najkritičnije sigurnosne rizike za web aplikacije. Ti rizici izvedeni su na temelju sakupljenih podataka od strane zajednice, industrije i njihovih istraživanja. Analizom tih podataka, rangira se relevantnost i značaj sigurnosnih propusta te koliko često se pojavljuju. Na temelju toga, dobivaju se sigurnosni rizici koji utječu na veći broj web aplikacija. Na tablici 1., prikazani su najrelevantiji sigurnosni rizici za 2021. godinu. U koloni „primjer“ navedena je samo jedan jednostavan primjer, koji može dovesti web aplikaciju u rizik od navedene sigurnosne prijetnje [13]

2.1.1. Slabosti i rizici web aplikacija

Kada govorimo o slabostima, odnosno ranjivostima web aplikacija, važno je shvatiti da postoji veliki broj otvorenih vrata kojima visoko educirane osobe mogu dobiti neautorizirani pristup korisničkim ili poslovnim podacima koji se nalaze na web aplikaciji. Moglo bi se zaključiti da se rizične web aplikacije javljaju kada je kod loše napisan, što može biti istina, ali u praksi će jedna ili više loše napisanih metoda biti manje kritične od web aplikacija s lošom, odnosno nepromišljenom arhitekturom. Ako je arhitektura web aplikacije loša od samog korijena, čak i najbolji programeri neće uspjeti zakrpati sve sigurnosne propuste takve web aplikacije. Aplikacija je sigurna samo koliko i najslabija karika u njenoj arhitekturi. Uzmimo za primjer da se za svaki POST zahtjev ručno radi provjera XSS (eng. Cross-Site Scripting). Ako se samo jedan POST zahtjev ne očisti, sve ostale provjere su zapravo postale beskorisne jer aplikacija sada može biti napadnuta kroz ta vrata. [14]

Arhitekturalne slabosti koje se javljaju kod web aplikacija mogle bi se klasificirati kao propuštene slabosti, slabosti u provođenju i slabosti u realizaciji. [15]

Propuštene slabosti su one koji dizajneri arhitekture ili programeri nisu predvidjeli te samim time, takvim slabostima nedostaju sigurnosne mjere odnosno strategije. [15] Za primjer se može uzeti da arhitekti nisu uzeli u obzir zaštitu od uskraćivanja usluge regularnim izrazima (eng. Regular Expression Denial of Service). U takvim slučajevima, napadač može iskoristiti ulazne elemente koji koriste regularne izraze za podudaranje uzorka te time smanjiti dostupnost poslužitelja drugim korisnicima.

Oznaka	Naziv	Engleski naziv	Primjer
A01-2021	Loša kontrola pristupa	Broken Access Control	Zaobilaženje kontrole pristupa izmjenom URL-a
A02-2021	Kriptografske greške	Cryptography Failures	Korištenje zastarjelih ili vlastitih algoritama za kriptiranje podataka
A03-2021	Ubrizgavanje	Injection	Ne provjeravanje (filtriranje) sadržaja koje unosi korisnik
A04-2022	Nesiguran dizajn	Insecure design	Ne određivanje razine sigurnosti koja je potrebna za određeni sustav. Ne može se nadoknaditi tokom implementacije.
A05-2022	Pogrešna konfiguracija sigurnosti	Security Misconfiguration	Izložene varijable nisu postavljene na sigurne vrijednosti
A06-2022	Ranjive i zastarjele komponente	Vulnerable and Outdated Components	Ne redovito traženje ranjivosti komponenti, biblioteki i ostale ovisnosti koje se koriste
A07-2023	Greške kod identifikacije i autentifikacije	Identification and Authentication Failures	Dozvoljavanje slabih zaporki i loša validacija sesije
A08-2023	Greške integriteta softvera i podataka	Software and Data Integrity Failures	Korištenje dodataka koji dolaze iz nepouzdanih izvora i njihov integritet nije ovjeren
A09-2023	Greške praćenja i sigurnosnog dnevnika rada	Security Logging and Monitoring Failures	Ne zapisivanje ključnih akcija korisnika (uspjele/neuspjele prijave i odjave, transakcije...)
A10-2024	Krivotvorenje zahtjeva na strani poslužitelja	Server-Side Request Forgery	Dohvaćanje udaljenih resursa bez provjere valjanosti URL

Tablica 1: OWASP 10 web aplikacijskih sigurnosnih rizika (Izvor: vlastita izrada, prema: [13])

Slabosti u provođenju su slabosti u kojima su arhitekti web aplikacije primjetili slabost i mogućnost napada, no kao mjeru zaštite odabrali su krivu taktiku koja ponovno može dovesti do uspješnog proboja kroz sigurnosne sustave. [15] Najjednostavniji primjer takve slabosti je korištenje lošeg algoritma hashiranja za korisničke lozinke kao što je MD5 (eng. Message Digest Method 5) algoritam.

Realizacijske slabosti su one gdje je odabrana dobra taktika, ali je implementacija te taktike u samu bazu koda (eng. codebase) loše izvedena. [15] Kao primjer možemo navesti kriptiranje lozinke. Arhitekti su se odlučili za korištenje Bcrypt algoritma za kriptiranje, no prilikom slanja zahtjeva na krajnju točku poslužitelja, u parametre upita šalju lozinku kao čisti tekst te tu lozinku kriptiraju tek na poslužitelju. U ovom slučaju, napadač može presresti zahtjev prema poslužitelju te dobiti čitljivu zaporku korisnika.

Sigurnosni aspekti	Slabosti
Dnevnik rada sustava	• Izostavljanje informacija releventnih za sigurnost; • Izlagavanje informacija kroz dnevnik rada; • Bilježenje prekomjernih podataka.
Autentifikacija	• Neispravna autentifikacija; • Korištenje nesigurnih ili brzih hash algoritama.
Autorizacija	• Nepravilno postupanje s nedovoljnim dozvolama; • Nepravila kontrola pristupa; • Izlaganje privatnih podataka; • Nepouzdani put pretraživanja.
Validacija unosa	• Neispravna provjera valjanosti unosa; • Krivotvorenje zahtjeva (CSRF); • Deserijalizacija nepouzdanih podataka.
Enkripcija podataka	• Nedostatak enkripcija osjetljivih podataka; • Upotreba hash algoritma bez soli ili sa predvidljivom soli; • Nedovoljna zaštita podataka za prijavu.
Upravljanje sesijama	• Neadekvatno istjecanje sesije; • Nedovoljna duljina ključa sesije.

Tablica 2: Sigurnosne slabosti web aplikacije prema sigurnosnim aspektima [15]

Slabosti i rizike koje te ranjivosti nose na sigurnost web aplikacije možemo svrstati pod sigurnosne aspekte. Na prethodnoj tablici 2., prikazano je nekoliko sigurnosnih aspekata te

slabosti koje one mogu nositi. U praksi postoji i više sigurnosnih aspekata nego što je to navedeno u tablici 2. kao i mnogo više slabosti, ovisno o vrsti web aplikacija i podacima kojima se ona bavi. Također je važno napomenuti da kako se web tehnologije razvijaju, tako dolaze i nove slabosti odnosno sigurnosne prijetnje i razvoj novih sigurnosnih prijetnji je gotovo nemoguće zaustaviti. Jedini način na koji se sigurnost web aplikacije može održati na visokoj razini, je konstantnim praćenjem novih tehnologija, novih sigurnosnih prijetnji te korištenjem sigurnih, praktičnih i efiksanih metoda zaštite.

2.2. Napadi na web aplikacije

Napadi na web aplikacije danas su uobičajena pojava na internetu. Napadači napadaju web aplikacije iz različitih motiva, kao što su financijski dobitak, pribavljanje privatnih informacija, ideološki motivi ili jednostavno da nanese štetu i naruši stanje web aplikacije. Napadi mogu biti usmjereni na različite sigurnosne aspekte web aplikacije, koristeći različite metode i vrste napada. Profesionalni hakeri analiziraju aplikaciju i njezino okruženje te traže tragove koji bi mogli dovesti do mjesta ranjivosti, kako bi kasnije izvršili napad na samu web aplikaciju.

Da bi se osigurala sigurnost web aplikacije, odnosno zaštitila od vanjskih prijetnji, potrebno je jasno razumjeti prijetnje i napade koji mogu utjecati na web aplikaciju. Web aplikacije oduvijek su meta kriminalnih hakera, a s sve većom migracijom i integracijom različitih vrsta organizacija i poslovanja u web rješenja, zaštita web aplikacija i usluga postaje kritični faktor za uspješno poslovanje. U nastavku će biti detaljnije prikazane i razrađene neke od najčešćih vrsta napada na web aplikacije koje mogu nanijeti štetu korisnicima web aplikacija, organizacijama odnosno vlasnicima web aplikacija te samoj usluzi koju web aplikacija pruža. [7]

2.3.1. Napad skriptama (XSS)

Napad skriptama (eng. Cross-site scripting), skraćeno XSS, jedan je od najčešćih napada na web aplikacije. Općenito, XSS napad iskorištava HTML elemente koji primaju korisnički unos i JavaScript kod koji se može ubaciti u te unose i pokrenuti u pretraživaču bez znanja korisnika. Svaki preglednik, u pozadini, izvršava JavaScript naredbe bez znanja i dozvole korisnika, što zajedno s nečišćenim unosima HTML elemenata može predstavljati veliki sigurnosni rizik. Korisnik može izravno unijeti u HTML element, na primjer, <input>, <script> i bilo koji JavaScript kod unutar elementa. Kad preglednik vidi element <script>, shvatit će to kao JavaScript kod koji treba izvršiti, što samo po sebi nije pogreška već standardno ponašanje preglednika. Međutim, u rukama stručnjaka, ta funkcionalnost, kombinirana s lošom arhitekturom web aplikacije i malo skriptnog koda, može postati izuzetno opasno oružje. [14, 16] Jedan od mogućih primjera XSS napada koji može dovesti korisnike web aplikacije u izuzetno opasnu situaciju je sljedeći:

- Napadač uočava propust web aplikacije sa neočišćenim (eng. unsanitized) odjeljakom za upisivanje komentara na blog;
- Napadač u komentar bloga ubrizgava zloćudnu skriptu koja će se pokrenuti na preglednicima drugih korisnika prilikom otvaranje tog bloga;
- Zloćudna skripta se sprema kao komentar bloga u bazu podataka zbog loše arhitekture web aplikacije i sigurnosnih sustava;
- Drugi korisnici otvaraju taj blog te se komentari dohvaćaju iz baze podataka;
- Prilikom učitavanja (eng. rendering) stranica na strani klijenta, preglednik uočava javascript kod te pokreće interpretetirati kod;
- Zloćudna skripta pokreće skidanje malware programa koji u pozadini instalira malware za praćenje korisničkih unosa te šalje podatke na poslužitelj napadača.

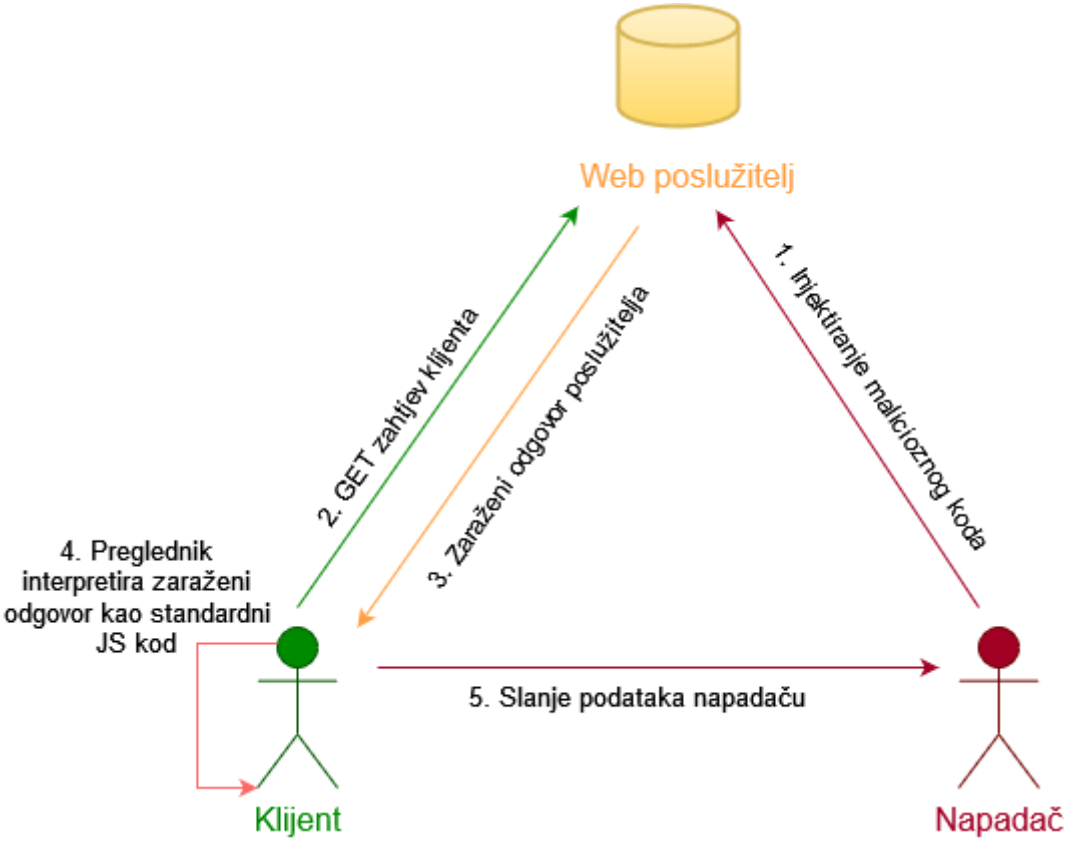
Napad skriptama pokreće kod u pregledniku koji nije napisan od strane vlasnika web aplikacije. Taj kod se ubacuje u dijelove web aplikacije te se pokreće u pozadini, bez znanja korisnika ili potrebe za akcijom kako bi se aktivirao. Opasnost leži u tome što takav zloćudan kod može ukrasti tokene sesije ili kolačiće, može provoditi razne radne tijekove koji štete korisniku ili vlasniku web aplikacije, te može slati ili primati podatke s zloćudnog web servera. Izvor ovakvog napada je propust ili potpuni nedostatak čišćenja elemenata koji omogućuju unos podataka u web aplikaciju. [14]

XSS napadi mogu se izvesti putem različitih vektora na web aplikaciji [16]:

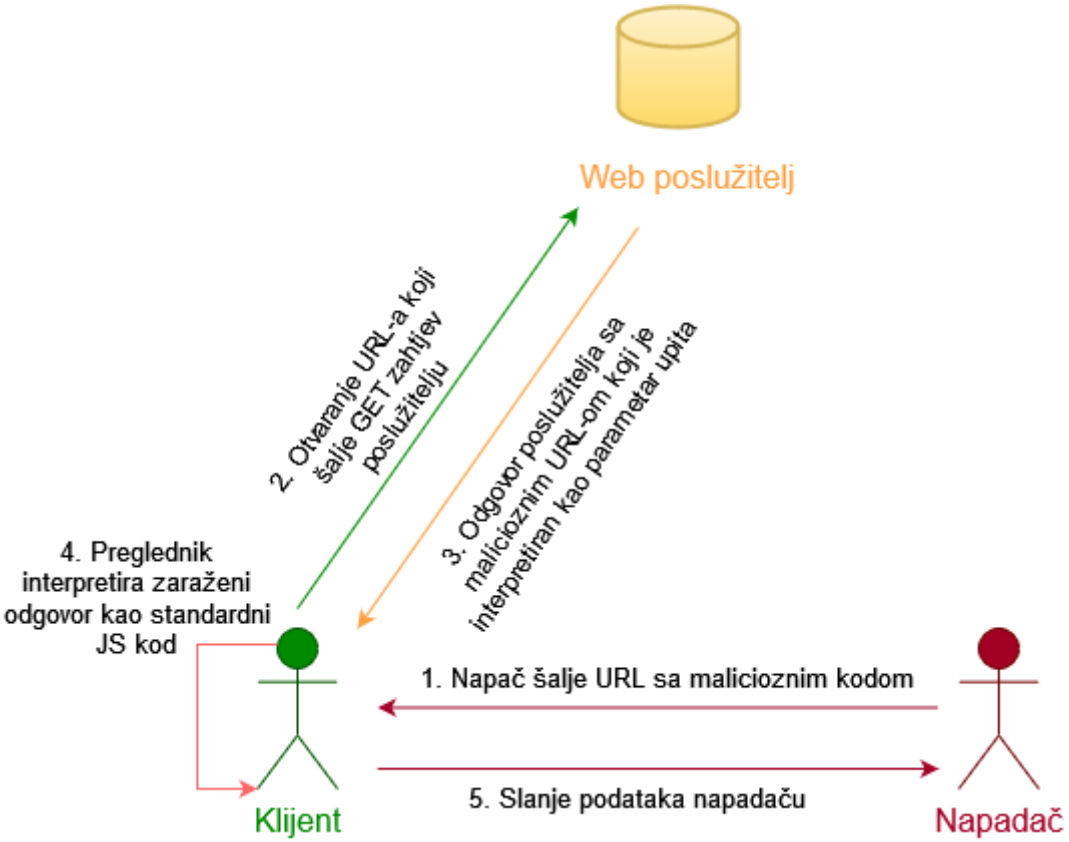
- URI komponente;
- Polja obrasca;
- Stilski jezik CSS (eng. Cascading Style Sheets);
- Zaglavlje HTTP zahtjeva;
- Kolačići;
- JSON (eng. JavaScript Object Notation) objekti;
- Svojsta DOM-a (eng. Document Object Model);
- Korisnički izrađen sadržaj;
- SVG (eng. Scalable Vector Graphics) elementi.

Kao što se može primijetiti, postoji mnogo mogućnosti za napadača da pronađe slabu točku web aplikacije i mjesto napada. Najčešći izbor za izvođenje napada su URI komponente i komponente obrasca. Ako aplikacija prihvaća bilo kakve podatke od preglednika, bez obzira je li to ručni unos od korisnika ili automatski podatak od preglednika, postoji potencijal za ubacivanje skripti i stvaranje sigurnosnih propusta i rizika.

Postoje različite vrste XSS napada. Najčešće vrste XSS napada su spremljeni (perzistentni) i reflektirani napadi. Osim njih, postoje i XSS DOM napadi i mutirani XSS napadi, ali su znatno rjeđi od prethodno navedenih. Razlika između ovih napada leži u načinu spremanja koda i mjestu ubacivanja samog koda. Spremljeni napadi su standardni i najčešći oblik XSS napada jer se spremaju u bazu podataka i dohvaćaju se na zahtjev drugih korisnika. Spremljeni napadi obično se upisuju u određene obrasce na web aplikaciji. S druge strane, reflektirani napadi se ne spremaju u bazu podataka, već se ubacuju izravno u URL i šalju klijentima (žrtvama) koji otvaraju zaražene linkove. Poslužitelj ne treba vratiti zlćudni kod prilikom obrade takvog zahtjeva, već se kod reflektira u korisničkom pregledniku prilikom otvaranja poveznice. Zbog toga su reflektirani napadi teže primjetni od spremljenih XSS napada. Na sljedećim slikama prikazani su tipični scenariji za perzistentne i reflektirane XSS napade.



Slika 2: Tijek rada perzistentog XSS napada (Izvor: vlastita izrada, prema: [14])



Slika 3: Tijek rada reflektiranog XSS napada (Izvor: vlastita izrada, prema: [14])

2.3.2. Ubrizgavanje

Napadi ubrizgavanjem (eng. Injection) su vrsta napada u kojoj napadač ubacuje kod u web aplikaciju. Taj kod se interpretira zajedno s postojećim kodom aplikacije. Napadi ubrizgavanjem obično se koriste za krađu, uništavanje i modifikaciju podataka ili za DoS napade (eng. Denial of Service). Napadi ubrizgavanjem sastoje se od dvije glavne komponente: interpretera i payloada, koji prenose podatke do interpretera. Ovisno o arhitekturi web aplikacije, neke vrste napada ubrizgavanjem imaju veći ili manji sigurnosni rizik. Postoji nekoliko različitih vrsta napada ubrizgavanjem [7, 14]:

- SQL (eng. Structured Query Language) ubrizgavanje;
- Ubrizgavanje naredi;
- XXE (eng. XML (eng. Extensible Markup Language) External Entity Injection) ubrizgavanje;
- XPATH (eng. XML Path Language) ubrizgavanje.

SQL ubrizgavanje je jedan od najstarijih i najčešćih tipova ubrizgavanja. SQL ubrizgavanje omogućuje napadaču manipuliranje naredbama između web aplikacije i njezine baze podataka. Prilikom korištenja web aplikacije, korisnici često stvaraju nove objekte, uređuju postojeće ili ih brišu. Na primjer, korisnik može dodati artikl u svoju košaricu, ažurirati količinu ili izbrisati artikle

iz košarice. Sve te interakcije se u određenom trenutku odražavaju na bazu podataka, odnosno izvršavaju se odgovarajuće SQL naredbe nad bazom. Problem nastaje kada se u kodu koriste konkatenacije stringova za izgradnju SQL naredbi. Na taj način, SQL naredba postaje dinamička, a jedan SQL upit se stvara. Napadač može lako iskoristiti takve SQL upite i dobiti pristup čitanju, brisanju i izmjeni podataka unutar same baze podataka, što može dovesti do potpunog uništenja web aplikacije i čak organizacije, ukoliko nema sigurnosnih kopija baze podataka. [7, 13, 16]

```
String upit = "SELECT * FROM korisnik WHERE korisnikId='" +
request.getParameter("id") + "'";
```

U prikazanom primjeru, dohvaćaju se podaci korisnika gdje je primarni ključ korisnika jednak onome iz zahtjeva. Ovdje je veoma jasno da se unutar samog SQL upita spajaju vrijednosti bez ikakvog saniranja te je veoma jednostavno odraditi napad SQL ubrizgavanjem.

Napad ubrizgavanjem naredbi je napad u kojem napadač unosi Bash naredbe kao parametre u zahtjeve prema API-ju (eng. Application Programming Interface), a takav unos korisnika se spaja u kodu. Te naredbe se izvršavaju na operativnom sustavu poslužitelja i mogu rezultirati potpunom kompromitacijom poslužitelja ako se svaka naredba izvodi kao super korisnik. Ovaj napad je jedan od najopasnijih i visoko rizičnih, jer ako je poslužitelj ranjiv, napadač može dobiti pristup i kontrolu nad svim podacima na poslužitelju, log datotekama, brisati administratorske račune i uništiti cijeli sustav poslužitelja. [14]

XXE (eng. XML External Entity) napadi su još jedna vrsta napada ubrizgavanjem koja se javlja na krajnjim točkama API-ja koje kao parametar primaju podatke u XML formatu. Podaci koji se šalju na krajnju točku API-ja ne moraju biti čisti XML format, već mogu biti izvedenice XML-a kao što su PDF (eng. Portable Document Format), HTML ili SVG format. XML specifikacija ima notaciju za uvoz vanjskih datoteka koja se naziva vanjski entitet (eng. External entity). Za provedbu XXE napada, napadač može stvoriti posebnu XML datoteku i poslati je kao parametar na krajnju točku nekog API-ja. Poslužitelj će zatim tu XML datoteku izvršiti pomoću XML parsera, što može rezultirati probijanjem sigurnosnog sustava na samom poslužitelju. [14]

2.3.3. Napad na autentifikaciju

Autentifikacija je proces kojim neki subjekt dokazuje svoj identitet. Na webu, identitet se obično potvrđuje korištenjem emaila ili korisničkog imena i lozinke. Korisnik prilikom prijave unosi svoje korisničko ime i lozinku, i ako su podaci ispravni, korisnik se autentificira. Nakon uspješne autentifikacije, stvara se sesija ili sjednica koja jedinstveno identificira korisnika ili preglednik na web aplikaciji. Sesija se obično sprema u kolačićima ili u spremištu preglednika. Nakon

autentifikacije, svi zahtjevi prema poslužitelju šalju se zajedno s podacima sesije, često u obliku cjelokupnog kolačića, kako bi se utvrdio identitet korisnika koji šalje zahtjev.

Sustav autentifikacije je ključan dio web aplikacije i njezine sigurnosti. Napadač može sustave autentifikacije napasti raznim metodama [16, 17]:

- Napad grubom silom (eng. Brute force attack);
- Obrnuti inženjering sesije;
- XSS napad;
- SQL ubrizgavanje;
- Njuškanje paketa (eng. Payload sniffing).

Napadi grubom silom su napadi na autentifikacijske sustave koji se temelje samo na zaštiti lozinkom. Napadač grubom silom unosi sve moguće lozinke, najčešće iz nekih rječnika u kojima je definiran veliki broj lozinki. Ovaj napad iskorištava sustave koji koriste brze algoritme hashiranja lozinke, jer se svaki pokušaj provale može puno brže obaviti za svaku jedinstvenu lozinku. Osim toga, ranjivi su korisnički računi koji koriste jednostavne ili često korištene lozinke. Napad grubom silom može se izvesti na način da se za određenog korisnika isprobavaju sve moguće lozinke, ili da se za skup korisnika isprobava jedna lozinka, a zatim se mijenja lozinka nakon svake iteracije. Iako je sam napad vrlo jednostavan, njegova jednostavnost ne bi smjela smanjivati rizik ovakvog sigurnosnog napada. [16, 17]

Ako sustav za autentifikaciju ima lošu arhitekturu i proces rada prilikom kreiranja sesija, napadač može vrlo jednostavno pratiti sesiju i izvršiti obrnuti inženjering kako bi kreirao vlastitu sesiju. Slabosti koje mogu do ovoga su dovesti je ako se koriste nesigurni algoritmi hashiranja kao štu su MD5 ili SHA1. Osim toga sesije su vrlo ranjive ukoliko ne postoji vrijeme koliko dugo sesija traje već se ona mijenja tek kada korisnik promijeni lozinku. U tom slučaju ukoliko napadač uspije dobiti sesiju korisnika, imat će pristup njegovom računu toliko dugo dok korisnik ne promijeni lozinku. Primjer sesije koja je veoma nesigurna i sklona napadima:

```
kolacic = btoa(korisnickoIme + ":" + korisnikId + ":" + MD5(lozinka).toString()
```

Ovakvo kreiranje vrijednosti za sesiju je veoma jednostavno kompromitirati, jer se u sesiju spremaju svi osjetljivi podaci korisnika te se koristi nesiguran hash algoritam. [16]

Kao što je već prethodno spomenuto, XSS i ubrizgavanje su vrlo opasni napadi koji mogu kompromitirati različite sustave. Ako web aplikacija nema zaštitu od XSS-a, korisnička sesija, bez obzira na njezinu sigurnost, može se jednostavno izložiti napadaču ako se napadačev kod može interpretirati u pregledniku korisnika. Također, korisnik može, bez adekvatne zaštite, ubrizgavanjem koda poput SQL-a, vrlo jednostavno doći do sesija koje su spremljene u bazi podataka.

Njuškanje paketa je napad koji iskorištava prijenos podataka putem nesigurnog HTTP protokola. Napadač se smješta između web poslužitelja i preglednika korisnika. Napadačima je najlakše promatrati promet koji se odvija između preglednika i web poslužitelja, osobito ako taj promet nije kriptiran i koristi HTTP protokol. Jednostavnom naredbom poput:

```
tcpdump –nq –s1600 –X port 80
```

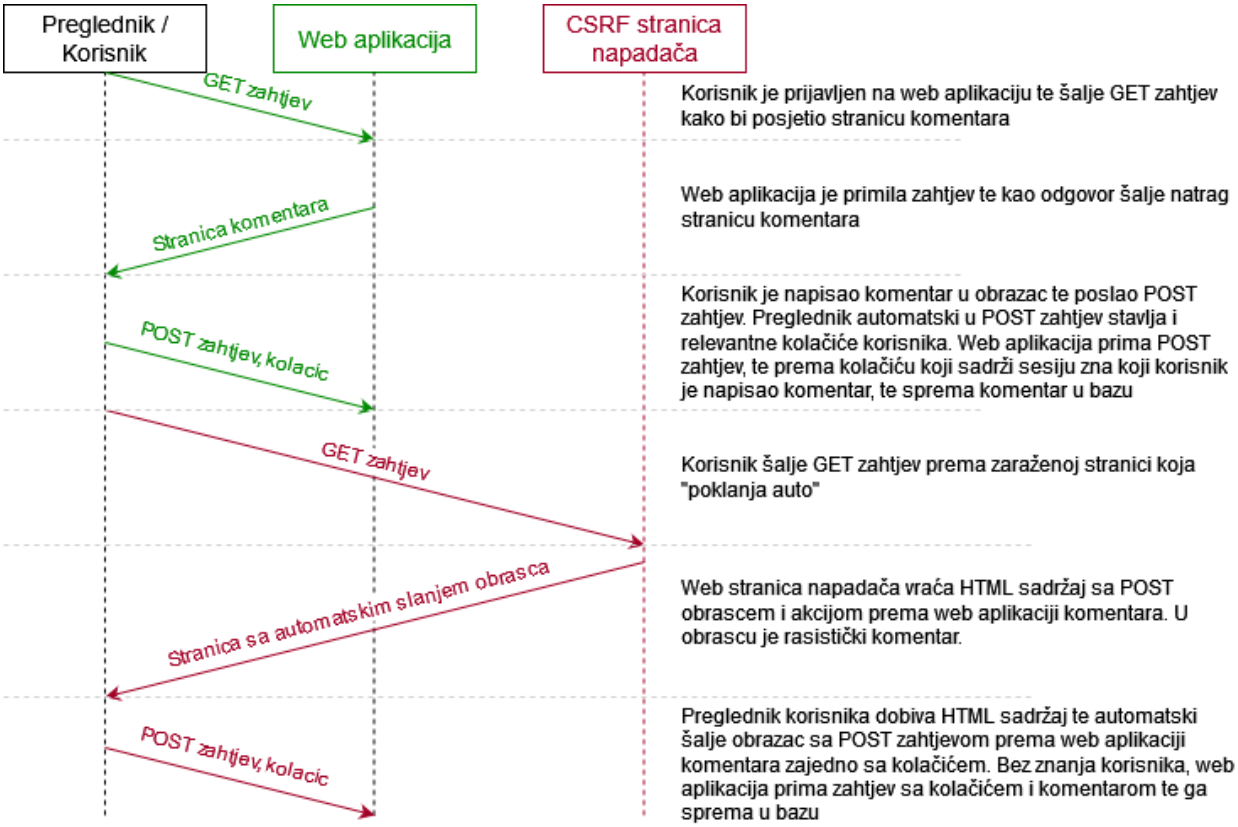
može se promatrati sav promet na nekom portu bez ikakvog znanja korisnika. Na taj način napadač može dobiti pristup kolačićima korisnika koji se šalju kroz zahtjeve te se predstaviti kao napadnuti korisnik. Neke web aplikacije koriste HTTPS (eng. Hypertext Transfer Protocol Secure) protokol samo na stranicama za registraciju i prijavu. Miješanje sigurnih i nesigurnih protokola siguran su put do sigurnosnih rizika koji će prije ili kasnije biti iskorišteni od strane napadača. [16]

2.3.4. CSRF

CSRF (Cross-site request forgery) je napad koji iskorištava način rada preglednika i povjerenje između preglednika i web stranice. Korištenjem CSRF napada moguće je poslati zahtjev web aplikaciji, preko veze, obrasca ili druge web aplikacije, ali u ime korisnika ili preglednika. Ovakvi napadi obično su potpuno nevidljivi korisniku. Drugim riječima, napadač može poslati svoje zahtjeve koristeći kolačiće korisnika te izvršiti određenu akciju na web aplikaciji, ako je korisnik autentificiran na ciljanoj web aplikaciji. CSRF napadi mogu se provesti na sve HTML zahtjeve, no POST, PUT i DELETE zahtjevi su teži za realizaciju, te korisnik najčešće mora otvoriti potpunu web stranicu kako bi se takav napad izveo. Za GET CSRF napad dovoljan je samo link kojeg korisnik klikne, te će se u tom trenutku izvršiti napad, ili slika koja kao izvor ima adresu ciljane web aplikacije zajedno s parametrima. [18]

Neke pretpostavke CSRF napada govore da bi korisnik odnosno preglednik koji se napada trebao biti prijavljen na ciljanu web aplikaciju. U većini slučajeva, to je istina jer se tada može iskoristiti identitet žrtve, no ne treba zanemariti štetu koju CSRF napad može prouzročiti bez autentifikacije. Kada CSRF napad koristi autentificiranog korisnika za napad na ciljanu web aplikaciju, napad može poslati jedan ili mnogo više zahtjeva prema ciljanoj web aplikaciji, te se CSRF napadi veoma dobro kombiniraju s drugim napadima kao što su ubrizgavanje ili XSS napadi. Također, CSRF napadi zaobilaze politiku iste domene (eng. Same Origin Policy) koja inače zabranjuje slanje i primanje zahtjeva ako su domena, protokol i portovi različiti. No, pošto se preko žrtve šalju zahtjevi prema ciljanoj aplikaciji, napadač ustvari zaobilazi ovu sigurnosnu značajku preglednika. Ono što napadač ne može je čitati odgovor web aplikacije nakon napada, osim ako se CSRF ne kombinira s drugim napadima.

Nadalje, prikazan je jedan primjer dijagram toka između korisnika odnosno preglednika, ciljane web aplikacije i aplikacije napadača. [14, 16]



Slika 4: Primjer CSRF napad (Izvor: vlastita izrada, prema: [18])

2.3. Ublažavanje sigurnosnih rizika web aplikacije

Do sad se u radu govorilo o slabostima i rizicima web aplikacija i napadima koji mogu biti provedeni nad takvim web aplikacijama. Važno je shvatiti da niti jedna web aplikacija, niti jedan digitalni sustav ne može biti u potpunosti siguran. Rizik sigurnose kompromizacije uvijek postoji jer se iz dana u dan razvijaju nove metode, novi načini i tehnike kako savladati postojeće sigurnosne sustave obrane. Zbog toga je veoma važno ostati u kontinuitetu sa tehnologijama.

O sigurnosti web aplikacije važno je početi razmišljati već od samog temelja prije nego što se počne razvijati ista. U toj fazi, razvija se arhitektura web aplikacije. Arhitekturu web aplikacije valja veoma ozbiljno shvatiti jer može uvelike pomoću u ublažavnju sigurnosnih rizika ili obrnuto stvoriti rizike na svakim vratima. Kod izrade arhitekture, važno je zaključiti kako će se podaci kretati iz točke A u točku B, kako će se i gdje ti podaci spremati, kako će podaci biti prezentirani korisniku te kako će poslužitelj obrađivati korisničke podatke. Tokom detaljnog razmatranja navedenih točaka, lakše je pronaći mane i te mane popraviti na arhitekturalnoj razini. [14]

Nakon što je prihvaćena sigurna arhitektura web aplikacije, započinje sljedeća faza životnog ciklusa, a to je razvoj same web aplikacije. U ovoj fazi, opreza nikad dovoljno. Normalna je pojava da čovjek pogriješi ili u ovom slučaju napiše liniju koda koja može dovesti do sigurnosnog propusta. Rješenje je korištenje proizvoda za kontrolu verzioniranja. U današnjici među najpopularnijim takvim alatima je Github te je poprilično jednostavan za koristiti. U ovoj fazi također veliku ulogu igra i razina znanja ljudi koji razvijaju web aplikaciju. Stariji i iskusniji developeri uvijek bi trebali preuzimati pod sebe kompleksne i osjetljive zadatke, te one lakše davati mlađim i manje iskusnim programerima. [14] Sa današnjim tehnologijama i velikim brojem različitih otvorenih alata i dodataka, može se kreirati veoma dobar tijek rada koji će osiguravati kvalitetu koda u trenutku kada ona bude dodana na glavnu bazu koda bez obzira o neiskustvu mladih programera, naravno uz konstatnu provjeru iskusnijih. Primjer takvog work flowa za kontrolu verzioniranja na Git-u:

- Korištenje alata koji provjeravaju opis svakog dodavanja koda (npr. Husky)
- Korištenje alata koji provjeravaju ispravnost koda (npr. ESLint)
- Korištenje alata za formatiranje koda (npr. Prettier)
- Pokretanje koda prilikom svakog zahtjeva za spajanjem na glavnu granu
- Postavljanje aplikacije lokalno sa jediničnim i penetracijskim testiranjem

Sigurnosni rizik web aplikacija može se ublažiti traženjem, analizom i penetracijskim testiranje web aplikacije tokom implementacije, testiranja pa i tokom produkcije. Jedan od načina dobivanja informacija o ranjivosti web aplikacije je dobivanjem povratnih informacije. Ova metoda je i danas poprilično česta pošto se rukovodstvo organizacija ne voli spuštati u dodatne troškove sa zasebnim vremenom za traženje potencijalni bugova i sigurnosnih propusta ili unajmljivanje eksternih penetracijskih testera. No, takve odluke mogu se obiti u glavu ukoliko kriva osoba sazna za propuste. Nakon što se pronađu sigurnosni rizici, potrebna je njihova analiza kako bi se razjasnilo koji su sljedeći koraci sa takvim rizicima pošto svaki sigurnosni propust sa sobom nosi veći ili manji rizik. Primjerice ako se radi o kompromizacija korisničkih podataka, takav propust nosi sa sobom veliki rizik te bi trebao biti prvi na prioritetnoj listi za sanaciju. Najbolja praksa za organizaciju koja stvarno brine o sigurnosti svoje web aplikacije je aktivan trud da se smanje rizici od sigurnosnih propusta. Takva strategija ublažavanja trebala bi biti raširena cijelom aplikacijom te posebno na osjetljivim i ključnim područjima web aplikacije. Strategija ublažavanje trebala bi od samog početka imati sigurnu aplikacijsku arhitekturu, koristiti najbolje prakse sigurnog kodiranja, okvire za regresijsko testiranje nakon svakog ispravka greške te programski način razmišljanja siguran po dizajnu. [14]

3. Privatnost osobnih podataka

Privatnost osobnih podataka svodi se na sigurnost, kontrolu i zaštitu povjerljivosti nad informacijama koje se odnose na pojedinca te mogu jedinstveno identificirati tog pojedinca. Osobni podaci se veoma često prikupljani i obrađuju putem različitih kanala i medija što uključuje i web aplikacije. Privatnost osobnih podataka je iznimno značajna kako bi se osiguralo da se sa takvim podacima postupa u skladu sa propisima i željama osobe te da se spriječi krađa i zloupotreba takvih podataka. Jedna od agencija koja štiti prava osobnih podataka u Republici Hrvatskoj je Agenzija za zaštitu osobnih podataka. Oni nadziru provedbu zakona o zaštiti podataka prema Općoj uredbi o zaštiti podataka. Opća uredba o zaštiti podataka je najstroži zakon koji propisuje iznimno visoke standarde privatnosti i sigurnosti podataka koje organizacije obavezno moraju poštivati. Pojedinici prema uredbi imaju pravo na pristup, ispravak, brisanje, ograničenje obrade, prenosivnost podataka te prigovor. Kako bi se postigli takvi standardni potrebno je koristiti smjernice za najbolju praksu zaštite osobnih podataka kod web aplikacije. Konzorcij W3C (eng. World Wide Web Consortium) objavio je 2012. godine dokument u kojemu su propisane smjernice za postizanje najbolje prakse privatnosti osobnih podataka kod web aplikacija. U nastavku detaljnije će biti istraženi osobni podaci njihova zaštita te organizacije koje prinosi zaštiti osobnih podataka i postizanju najboljih praksa privatnosti kod web aplikacija.

3.1. Osobni podaci i njihova obrada

Osobni podaci su podaci koji identificiraju jedinstvenog individualca. Jedna individu identificira se svojim osobnim podacima odnosno dokumentima kao što su osobno iskaznica, vozačka dozvola ili putovnica. Ti dokumenti sadrže osobne podatke kao što su ime, prezime, mjesto rođenja, adresa, broj isprave i identifikacijski brojevi kao što je OIB (osobni identifikacijski broj). Danas velika većina stanovništa koristi pamenti mobitel koji također sadrži veliki broj osobnih podataka kao što su IP (eng. Internet Protocol) adresa, MAC (eng. Media Access Control Address) adresa, GPS (eng. Global Positioning System) lokacija, broj telefona, privatni mediji (slike, videozapisi, zvučni zapisi) te biometrijski podaci. Također, definirane su i posebne kategorije osobnih podataka u koje spadaju osobni podaci koji se odnose na rasno ili etničko podrijetlo, politička stajališta, vjerska uvjerenja, osobni podaci koji se odnosno za zdravlje i spolni život, podaci o kaznenim i prekršajnim postupcima. Svi navedeni podaci odnose se na privatne osobe. Podaci pravnih osoba, primjerice e-mail adresa, financijski podaci, matični broj i naziv ne smatraju se osobnim podacima te smiju biti javno dostupni. [19, 20]

Pod obradom osobnim podataka uvrštavaju se radnje kao što su prikupljanje, bilježenje, korištenje, čuvanje, otkrivanje trećim stranja te uništavanje. Obrada se odnosi i na fizičke i

digitalne osobne podatke. Primjerice ukoliko se osoba zaposli kod novog poslodavca, te osobne podatke poslodavac ustvari obrađuje. Isto vrijedi i za digitalne platforme, odnosno ukoliko se osoba registrira na nekoj društvenoj mreži ili nekoj drugoj web aplikacija koja zahtijeva osobne podatke, ti podaci su na obradi od strane vlasnika tih aplikacija. [20]

3.2. Zaštita osobnih podataka

Pravo na zaštitu osobnih podataka je jedno od temeljnih prava svakog čovjeka gdje je svrha zaštititi privatni život, ljudska prava te slobodu prikupljanja, obrade i korištenje osobnih podataka. Razvojem tehnologija i interneta, osobni podaci postaju sve više izloženiji i dostupni onima koji ih žele iskoristiti u svoju nezakonitu korist. Danas je postala normala koristiti mobilno bankarstvo gdje se mogu pronaći veoma osjetljivi podaci. Prema istraživanjima iz 2018-te godine, Hrvatska je ispod Europskog prosjeka u korištenju mobilnog bankarstva no to ne umanjuje prijetnje kojima mogu biti podložni korisnici. Takvi sustavi vrlo su dobro zaštićeni no napadači često dobivaju pristup takvim podacima putem napada krađe identitea (eng. phishing). Postavi se lažna web aplikacija u koju zatim žrtve iz neznanja upisuju svoje podatke koji mogu dovesti do pristupa osjetljivim osobnim podacima. Zbog sve veće digitalizacije zaštita osobnih podataka je važnija nego ikad. Korištenjem mobilnih uređaja i interneta povećao se opseg prikupljanja osobnih podataka te i sam izazov zaštite istih. Osobni podaci obrađuju se svakodnevno prilikom korištenja raznih aplikacija, web aplikacija i usluga. Obrada takvih podataka je zakonita tek kada je osoba odnosno ispitanik dao privolu za obradu svojih osobnih podataka, kada je ona nužna radi izvršavanja ugovora, kada je ona nužna da bi se zaštitili interesi ispitanika. [19, 20]

3.3. AZOP i GDPR

U Republici Hrvatskoj poznata je Agencija za zaštitu osobnih podataka ili skraćeno AZOP koja je samostalna i neovisno državno tijelo te nadziru provedbu zakona o općoj uredbi o zaštiti podataka (eng. General Data Protection Regulation - GDPR). Oni nadgledaju zakonitost obrade osobnih podataka te se na njihovoj web stranici mogu pronaći odgovori na mnoga pitanja o zaštiti podataka i o pravima osoba vezano uz njihove osobne podatke kao i prava na korištenje podataka. Također je na njihovim stranicama moguće pronaći prava koje osoba ima, a definirana su općom uredbom o zaštiti podataka, te vrijede za građane Republike Hrvatske i za sve građane Europske unije. Ta prava građana su pravo na pristup osobnim podacima, pravo na ispravak osobnih podataka, pravo na brisanje osobnih podataka, pravo na ograničenje obrade osobnih podataka, pravo na prigovor, pravo na prenosivost podataka te pravo u vezi automatiziranog pojedinačno donošenja odluke. [20]

Prethodno je već bila spomenuta Opća uredba o zaštiti podataka ili skraćeno GDPR. To je Europski zakon o privatnosti i sigurnosti podataka koji uključuje veliki broj novih zahtjeva za organizacije diljem svijeta. Trenutno je GDPR najstroži zakon o privatnosti i sigurnosti na svijetu. Usvojen je u Europskoj uniji no nameće se kao obavezan svim organizacijava koje ciljaju ili prikupljaju podatke od ljudi iz Europske unije. Uredba je na snagu stupila 2018. godine te se sa njima nameću milijunske kazne protiv onih koji krše standarde privatnosti i sigurnosti. Uredba sadrži 99 članaka koji se odnose na principe zaštite podataka. Ukoliko organizacija obrađuje osobne podatke oni trebaju biti obrađivani prema sedam načelima zaštite i odgovornosti. Ta načela su:

- Zakonitost, poštenje i transparentnost;
- Ograničenje namjene;
- Minimizacija podataka;
- Točnost;
- Ograničenje pohrane;
- Integritet i povjerljivost;
- Odgovornost.

Organizacija koja obrađuje podatke mora poštivati niz novih prava koje subjekt podataka ima, a čiji je cilj pojedincu dati veću kontrolu nad podacima koje daju organizaciji na obradu. Prava privatnosti ljudi su [21, 22]:

- Pravo na informiranost;
- Pravo pristupa;
- Pravo na ispravak;
- Pravo na brisanje;
- Pravo na ograničenje obrade;
- Pravo na prenosivnost podataka;
- Pravo na prigovor;
- Prava u vezi sa automatiziranim odlučivanjem i profiliranjem.

3.4. Najbolje prakse privatnosti web aplikacija

Konzorcij W3C je 2012. godine objavio dokument najbolje prakse privatnosti web aplikacija. Dokument se sastoji od 13 smjernica koje su podijeljenje 5 kategorija, a to su [23]:

- Privatnost prema dizajnu (eng. Privacy By Design);
- Dizajn usmjeren na korisnika (eng. User Centric Design);
- Minimalizacija prikupljanja i prijenosa osobnih podataka;

- Održavanje povjerljivosti osobnih podataka;
- Pristup kontroli i dnevniku rada (eng. Control and log access)

Privatnost prema dizajnu sadrži smjernicu koja naglašava proaktivno razmatranje privatnosti te očuvanje privatnosti i transprentnost kao zadana postavka. To se može provesti principima prevencija – ne sanacija, privatnost kao zadana postavka, privatnost ugrađena u dizajn, potpuna funkcionalnost (pozitivna, a ne nulta suma), potpuna zaštita životnog ciklusa (od kraja do kraja), vidljivost i transparentnost te poštivanje privatnosti korisnika. [23]

Dizajn usmjeren na korisnika sadrži sedam smjernica koje upućuju na važnost korisnika u dizajnu odnosno davanje korisniku razumijevanje i kontrolu nad upotrebom svojih osobnih podataka. U nastavku su ukratko objašnjene prakse koje utječu na navedenu kategoriju [23]:

- Omogućiti korisniku donošenje informirane odluka o dijeljenju svoji osobnih podataka;
- Omogućiti korisniku donošenje odluke u odgovarajuće vrijeme s točnim kontekstualnim informacijama;
- Dopustiti korisniku da jednostavno pregleda i promijeno svoje odluke vezane uz privatnost i pružanje zadanih postavki;
- Fokus na upotrebljivost i izbjegavanje nepotrebnih upita;
- Slobodan aktivni pristanak za specifične podatke sa informiranjem;
- Jasnost i transparentnost prema korisnicima u pogledu potencijalnih problema u vezi s privatnošću;
- Jasno određivanje vremenskog razdoblje u kojemu su neke informacije potrebne.

Minimalizacija prikupljanja i prijenosa osobnih podataka sadrži dvije smjernice koje pomažu da se pregledaju podaci, njihova struktura i korištenje te da se minimalizira količina i detalji podataka potrebnih za pružanje usluge. Prva smjernica naglašava traženje minimalnog broja podataka na minimalnoj razini detalja dok druga smjernica naglašava zadržavanje minimalne količine podatak na minimalno potrebno vrijeme radi moguće zlouporabe zadržanih podataka. [23]

Održavanje povjerljivosti osobnih podataka sadrži dvije smjernice. Prva smjernica naglašava održavanje povjerljivosti korisničkih podataka prilikom prijenosta podataka korištenjem sigurnog protokola za prijenos kao što je HTTPS dok druga smjernica ističe važnost održavanja povjerljivosti korisničkih podataka u pohrani. [23]

Zadnja kategorija sadrži jednu smjernicu koja izražava važnost kontrole pristupa prema informacijama korištenjem metoda za kontrolu pristupa te dnevnika rada za pregled pristupa podacima. [23]

4. Utjecaj sigurnosti i privatnosti na dizajn i arhitekturu web aplikacija

Prije samog početka razvoja web aplikacije, potrebno je razmisliti o sigurnosti i privatnosti korisnika. Ta područja danas su veoma kritični dijelovi prilikom pristupanja, odlučivanja i razvoja jer je potrebno, ne samo razviti sigurnu web aplikaciju sa arhitekturom koja će smanjiti mogućnost uspjelih napada, nego uzeti u obzir i privatnost korisničkih osobnih podataka. Razlozi

su jasno vidljivi, ukoliko je sigurnost loša, aplikacija će biti žrtva napada koji mogu oštetiti sve dionike web aplikacija. Ukoliko je privatnost korisnika loša, organizacija može biti strogo kažnjena od strane organizacije koje žele očuvati osobne podatke korisnika. Kako bi se u arhitekturu i dizajn web aplikacije uspješno ugradila sigurnost i privatnost postoji pet načela koje bi organizacijama trebale pomoći prilikom dizajna i implementacije [23]:

- Definiranje ciljeva sigurnosti i privatnosti;
- Odabir odgovarajuće arhitekture;
- Primjena uzoraka sigurnosti i privatnosti;
- Procjena arhitekture;
- Prilagođavanje arhitekture.

4.1. Definiranje ciljeva sigurnosti i privatnosti

Definiranje ciljeva sigurnosti i privatnosti prvi je korak za uključivanje načela u arhitekturu i dizajn aplikacije. Generalno gledano, ciljevi sigurnosti trebali bi biti očuvanje integriteta, dostupnosti i povjerljivosti dok privanosti zaštita podataka, minimalizacija spremanja korisničkig podataka te pristanak korisnika. Navedeni generalni ciljevi trebali bi biti korišteni na svakoj web aplikaciji, no postoji još mnogo ciljeva kao što su osiguravanje autentifikacije i autorizacije, minimiziranje ranjivosti i incidenata, pridržavanje relevantinih zakona, standarda i uredba, korištenje najboljih praksi, poboljšanje korisničkog iskustva. Ciljevi ovise o vrsti web aplikacije, podacima koji su dostupni na web aplikaciji te o korisničkim korištenju same web aplikacije. Recimo da organizacija izrađuje web aplikaciju koja služi u svrhu marketinga. Takva web aplikacija treba imati definiranje ciljeve sigurnosti i privatnosti, no rizici i regulacijske obaveze biti će puno manje nego za web aplikaciju koja se bavi praćenjem korisničkih financija. Štoviše, bez obzira na tip web aplikacije uvijek je potrebno provesti procjenu rizika, definirati ciljeve te primjerniti odgovarajuće mjere sigurnosti i privatnosti. [23]

4.2. Odabir odgovarajuće programske arhitekture

Kod odabira odgovarajuće arhitekture važno je znati prednosti i nedostatke svakog tipa arhitekture. U obzir je potrebno uzeti aspekte web aplikacije kao što su namjena, skalabilnost, funkcionalnosti, potrebni sustavi jer se na temelju tih aspekata odabire odgovarajuća arhitektura. Primejrice ako se razvija jednostavna web aplikacija sa relativnom malim budžetom i vremenom, monolitna arhitekture je najčešće odgovarajući tip arhitekture. No ukoliko se radi o projektu koji primjerice sadrži dvije web aplikacije, jednu za klijente i jednu za organizaciju, monolitna arhitektura je prejednostavna i gotovo nemoguća za implementirati i održavati te bi se s toga trebala koristiti, ovisno o zahtjevima, mikroservisna ili servisno orijentirana arhitektura. Postoji poprilično mnogo tipovi arhitekture od kojih su neke već bile spomenute. U nastavku je prikazano i detaljnije obrađeno nekoliko tipovi arhitekture: [23, 24, 25, 26]:

- Monolitna arhitektura
- Mikroservisna arhitektura
- Arhitektura bez poslužitelja
- Servisno orijentirana arhitektura

Arhitektura ne ovisi samo o sigurnosti i privatnosti već i o mnogim drugim faktorima kao što su funkcionalni zahtjevi, željena skalabilnost, korisnička očekivanja, vrsta tehnolgije itd. Važno je kod odabira arhitekture također obratiti pažnju na sigurnosne prednosti i nedostatke arhitekture kao i ciljeve koje su postavljeni. Temeljem nedostataka ili prednosti sigurnosti arhitekture fokus se može više posvetiti dijelovima koji su slabije ne zaštićeni i obrnuto. U nastavku će ukratko biti objašnjena svaka arhitektura te sigurnosne prednosti i nedostatci.

4.2.1. Monolitna arhitektura

Monolitna arhitektura je tradicionalni model razvoja programskog proizvoda. Kod monolitne arhitekture cijeli sustav je razvijen kao jedna cjelina. Sve komponente su međuzavisne i povezano te je svaka komponenta potrebna kako bi web aplikacija radila. Monolitna arhitektura je izrazito nefleksibilna i neskalabilna jer za dodavanje ili promjenu značajki potrebno je često mjenjati velike cijeline projekta. Na pozitivnu stranu, razvoj je veoma jednostavno te je arhitektura veoma dobra za male i jednostavne aplikacije gdje postoji mali budžet za razvoj. Sigurnost je također pozitivna strana monolitne arhitekture. Pošto se svi sustavi koji se koriste nalaze u jednoj bazi koda, jednostavnije je osigurati takvi sustav nego sustav koji je distributiran. Također površina koju napadač može napasti je mala te je zbog toga i sami nadzor aplikacije jednostavniji. Nedostatak monolitne arhitekture je problem sa pronalaženjem grešaka i sigurnosnih propusta pošto je otklanjanje grešaka teži proces od sustava koji su fragmentirani na dijelove. [23, 27]

4.2.2. Mikroservisna arhitektura

Mikroservisna arhitektura je pristup razvoju na način da se dijelovi funkcionalnosti razvijaju kao servisi koji rade kao nezavisne aplikacije. Takvi servisi komuniciraju najčešće putem HTTP protokola i njezinih metoda. Sa mikroservisnom arhitekturom dolaze mnoge prednosti kao što su korištenje različitih tehologija za svaki servis, zakazivanje jednog servisa ne utječe na rad ostalih servisa te se svaki servis može zasebno rasporediti (eng. deploy). Osim navedenih predonosti, mikroservisna arhitektura skalabilna je i održiva jer se može po potrebi kreirati novi servis te pozivati na potrebnim mjestima ili se već postojećem servisu lakše može dodati ili promijeniti funkcionalnost pošto je baza koda samog servisa puno manja. Sigurnost mikroservisne arhitekture kompleksnija je od jednostavne monolitne arhitekture. Tok podataka između servisa može biti različiti od servisa to servisa kao i način spremanja podataka i prijenose te zbog toga postoji rizik od curenja podataka ili presretanja od strane zlonamjernih aktera. Ti problemi se mogu riješiti snažnom enkripcijom kako u tranzitu podataka tako i u pohrani. Česti problem je konfiguracija i raspoređivanje servisa. Pošto jedan projekt može imati na tisuće mikroservisa koji koriste svoje ovisnosti (eng. dependecies), postoji potencijal za pojavljivanje konfiguracijskih grešaka ili korištenje ovisnosti koji možda imaju kompromitiranu sigurnost. Praćenje i zapisivanje aktivnosti i događaja također je aspekt koji je teži za realizirati korištenjem mikroservisne arhitekture zbog distribucije servisa. Zbog toga je potrebno implementirati centraliziran i konzistentan sustav praćenja koji može prikupljati, spremati i analizirati podatke od svih mikroservisa. [28, 29]

4.2.3. Arhitektura bez poslužitelja

Arhitektura bez poslužitelja zamjenjuje komponentu poslužitelja kod tradicionalnih aplikacija. Poslužitelj je najvažniji dio cijelog sustava koji treba biti dobro promišljen jer će obavljati najkompleksnije zadatke kao što su obavljanja važnih funkcija, spremanja podataka o korisnicima, izdavanje sigurnosnih vjerodajnica i slično. Svi zadaci pozivaju se putem strane korisnika odnosno preglednika. To se može zamijeniti arhitekturom bez poslužitelja u kojoj se zadaci neće pozivati iz preglednika nego će se aplikacijska logika premjestiti u sami preglednik. Na taj način, cijelu aplikaciju moguće je posluživati sa jednostavnim statičnim web poslužiteljom. Arhitektura bez poslužitelja još poznata i kao funkcija kao usluga (eng. FaaS – Function as a Service)je arhitektura gdje se aplikacija razgrađuje na okidače i akcije koji na potreban zahtjev pozivaju platformu koja pruža hosting i okruženje izvršenja za određenu funkciju. Ključne značajke i prednosti koje dolaze sa arhitekturom bez poslužitelja su:

- Cijena – korištenjem arhitekture bez poslužitelja smanjuje se operativni troškovi i troškovi skaliranja pošto nije potrebno upravljati poslužiteljima i bazama podataka, nije potreban

hardware niti licenca za operacijski sustav. Osim navedenih, prednost cijene je što se plaćaju samo potrebni računalni resursi koje se koriste tijekom vremena za izvršavanje;

- Skalabilnost – usporedno sa tradicionalnom arhitekturom, skaliranje se može obaviti ručno dodavanjem ili uklanjanjem instaci za obradu zahtjeva. Platforme koje nude svoje usluge automatski upravljaju skaliranjem funkcija na temelju broja akcija, odnosno kako se povećava opterećenje, povećava se i broj instanci za obradu;
- Održavanje – održavanje je minimalno pošto održavanje obavlja davatelj usluge;
- Sigurnost – prilikom razvoja, fokus se može u potpunosti pružiti kodu umjesto konfiguraciji poslužitelja jer davatelji upravljaju njihovim konfiguracijama te sigurnosnim zakrpima.

Korištenjem arhitekture bez poslužitelja dolaze i neki nedostaci. Klijenti koji koriste takvu arhitekturu ovise o dobavljačima usluga treće strane. Promjena dobavljača usluge uključuje promjenu koda ili arhitekture. Praćenje rada je također nedostatak jer dobavljači imaju različite sustave za praćenje koji su često veoma kompleksni te loše dokumentirani. Sigurnost je dvosjekli mač arhitekture bez poslužitelja. Razina sigurnosti klijenta povezana je sa sigurnosnom razinom dobavljača odnosno sigurnosnim značajkama koje daje dobavljač kao što su enkripcija, dopuštenja, sigurnosne operacije koje se izvode na poslužiteljima (ažuriranja i konfiguracija). Čak i ako je sigurnost poslužitelja na visokoj razini, ne znači da će aplikacija biti sigurna ako je kod aplikacije loše napisan jer će se javljati standardni napadi kao što su SQL ubrizgavanje, XSS ili ubrizgavanje naredbi. [30, 31]

4.2.4. Servisno orijentirana arhitektura

Servisno orijentirana arhitektura, skraćeno SOA, je metoda razvoja softvera za izgradnju distribuiranih sustava koji isporučuju servise drugim aplikacijama putem nekog protokola. Svaki servis pruža neku poslovnu sposobnost te servisi mogu međusobno komunicirati preko različitih platformi i programskih jezika. Ovakva arhiktetura omogućava korištenje servisa u različitim sustavima i kombiniranje različitih neovisnih servisa kako bi se obavili kompleksni zadaci. Prednosti SOA arhitekture su štednja vremena i troškova ponovnim korištenjem servisa u različitim poslovnim procesima, efektivno održavanje jer je jednostavno kreirati, ažurirati i osigurati mali servise te prilagodljivost različitim tehnologijama. Ključni principi servisno orijentirane arhitekture su:

- Interoperabilnost – svaki klijent može pokrenuti servis bez obzira na temeljenu platformu ili programski jezik;
- Labavo povezivanje – Servisi unutar SOA moraju biti što je manje moguće povezani sa vanjskim resursima te ne smiju zadržavati pinformacije o prethodnim sesijama i transakcijama (eng. Stateless);
- Apstrakcija – Korisnici servisa ne moraju znati logiku koda servisa niti detalje implementacije već trebaju dobiti potrebne informacije što servis radi i kako ga koristiti;
- Zrnatost – Servisi trebaju imati odgovarajuću veličinu, u idealnom slučaju jednu poslovnu funkciju.

SOA servisi funkcioniraju na način da klijent pošalje zahtjev servisa zajedno sa nekim ulaznim podacima. Servis obrađuje podatke, odrađuje zadatak te vraća odgovor putem komunikacijskog protokola. Servisi komuniciraju putem komunikacijskih protokola koji imaju uspostavljena svoja pravila za prijenos podataka putem mreže. Najpoznatiji SOA komunikacijski protokoli su SOAP (eng. Simple Object Transfer Protocol) i REST (eng. Representational State Transfer). [32, 33]

Servisno orijentirana arhitektura ima neke sigurnosne rizike koje valja prepoznati. Iscrpljenost resursa može uzrokovati uskraćivanje usluga te se takvi napadi mogu poprilično jednostavno provesti protiv servisa na način da napadač iscrpi poslužiteljski sustav odnosno memoriju, procesnu snagu i mrežne kapacitete na način da primjerice pošalje velik broj HTTP-GET zahtjeva prema krajnjoj točki. Neadekvatno zaštićene krajnje točke servisa mogu također biti žrtve napada ubrizgavanjem i CSRF-a. SOAP servisi ranjivi su na XXE napade te na petljanje akcijama na način da se pokušaju pozivati operacija različite od onih koje su specificirane unutar tijela SOAP zahtjeva. [33]

4.3. Primjena uzoraka sigurnosti i privatnosti

Primjena uzoraka dizajna za privatnost i sigurnost pružaju rješenja za višekratnu upotrebu za probleme i izazove u sigurnsnim i privatnim aspektima arhitekture aplikacije. Uzorak dizajna je dobro osmišljen dizajn koji ocrtava problem, kontekst, rješenje i posljedice njegove primjene. Uzorci dizajna za sigurnost uključuju enkripciju za sprječavanje neovlaštenog pristupa ili modifikacije, dnevnik rada za sigurnost, validaciju ulaznih podataka na implementacijskoj razini. Na razini dizajna, uzorci su Secure Factory, Secure Chain of Responsibility, Secure State Machine itd. Uzorci privatnosti često su uzorci transparentnosti orijentirani prema korisnicima. Neki od uzoraka transparetnosti privatnosti su pravila o privatnosti, pitanja i odgovori o privatnosti, tablica osobnih podataka, nadzorna ploča privatnosti itd. Uzorci sigurnosti i privatnosti biti će detaljnije prikazani u sljedećem poglavlju. [23, 34, 35]

Odabir tehnologija također ima utjecaj na sigurnost i privatnost. Uz odgovarajuću arhitekturu potrebno je odabrati i odgovarajuće programske jezike kojima će biti najjednostavnije, ali i najsigurnije implementirati aplikaciju. Još važniji aspekt je odabir tehnologija odnosno ovisnosti treće strane koji će igrati ulogu u funkcionalnosti i arhitekturi. Funkcionalnim dijelovima aplikacije koji su veoma osjetljivi, ali i vitalni za rad cijelog sustava, trebala bi biti pružena posebna pažnja i razmišljanje. Recimo da se izrađuje aplikacija koja korisnicima dopušta praćenje financijskih aktivnosti. Takva aplikacija mora imati veoma dobar autentifikacijski i autorizacijski sustav te način na koji će spremati financijske i osobne podatke. Za autentifikaciju potrebno je odabrati prikladan algoritam enkripcije te dvofaktorsku autorizaciju. Naravno moguće je kreirati svoj algoritam enkripcije i metode dvofaktorske autorizacije, no vrlo vjerojatno postoje obje tehnologije koje su rigorozno testirane te prikladnije nego izrada vlastitih. Spremanje financijskih podataka korisnika u ovakvoj aplikaciji također može biti problematično jer su potrebne posebne metode sigurnosti i spremanja. U takvoj situaciji može se razmisliti o korištenju specijaliziranih tvrtki i njihovih usluga za pohranu takvih podataka. [23, 14]

4.4. Procjena i prilagođavanje arhitekture

Nakon što je kreirana arhitektura koja zadovoljava funkcionalne i sigurnosne zahtjeve, potrebna je evaluacija kako bi se osiguralo da ta arhitektura ispunjava sve uvjete. Procjena je najbolje provoditi prije početka same implementacija te se može provesti u bilo kojem ciklusu razvoja arhitekture, odnosno prije nego što je arhitektura i završena. Procjena arhitekture ima za cilj donijeti odgovore na pitanja da li je arhitektura prikladna za sustav u kojem će biti implementirana te ukoliko je dizajnirano više arhitektura, koja arhitektura je prikladnija za traženi sustav. Arhitektura će biti prikladna ako su sljedeći kritetriji zadovoljeni [23, 36]:

- Sustav će ispuniti svoje ciljeve kvalitete;
- Sustav će raditi predvidljivo i dovoljno brzo kako bi postigao tražene performanse
- Sustav će biti moguće izmjeniti na planirane načine;
- Sustav će zadovoljavati sigurnosne zahtjeve;
- Sustav će zadovoljavati funkcionalne zahtjeve.

Arhitektura nekog sustava također se može procijeniti prema aspektima kao što su performanse, pouzdanost, dostupnost, sigurnost, mogućnost izmjene, prenosivost, funkcionalnost. [36]

Prilagođavanje arhitekture potrebno je provesti u slučaju da je potrebno implementirati nove sigurnosne sustave, funkcionalne zahtjeve ili sustave privatnosti. Prilagođavanje trebalo bi biti namjerno i kontrolirano kako bi se poboljšala kvaliteta te riješili nedostaci ili problemi te spriječili sigurnosne prijetnje. Također razlog prilagođavanja arhitekture može biti izmjena regulacija ili standarda privatnosti korisnika. [23]

4.5. Zaštita od napada na web aplikacija

Iako se zaštita od različitih vrsta napada implementira na razini koda te nema direktan utjecaj na samu arhitekturu, važno je razumijeti na koji način komponente aplikacije međusobno komuniciraju kako bi se mogle provesti strategije zaštiti od različitih vrsta napada. U nastavku će biti prikazane metode i načini kako se zaštiti od napada koji su bili ranije navedeni u drugom poglavlju.

4.5.1. Zaštita od XSS napada

Najbolja zaštita protiv XSS napada je da se svi ulazni podaci od strane korisnika ne proslijeđuju u DOM na bilo koji drugi način osim kao podaci tipa string. Kako bi se osiguralo kvalitetno i sigurno prosljeđivanje podataka, potrebno je očistiti podatke unesene od strane korisnika. Iako je sama potpuna zaštita od XSS napada teška za izvesti jer postoji veoma mnogo vektora koja je moguće iskoristiti, dobra praksa je da se sve što konvertira tekst u DOM ili tekst u skriptu izbjegava jer su takve konverzije potencijalni vektori napada. Neki od takvih objekata i svojstva su `element.innerHTML`, `element.outerHTML`, `document.write`, `DOMParser.parseFromString` te objekt Blob i element SVG. SVG i Blob elementi su opasni jer imaju mogućnost izvršavanja koda. Sigunosni konfiguracijski alat CSP (eng. Content Security Policy) je također dobra zaštita od XSS napada. CSP pravila se mogu definirati na način da se olakšaju ili pojačaju sigurnosna pravila vezana uz kod koji se izvodi unutar aplikacije. Sa CSP pravila može se definirati kakve vanjske skripte se smiju uvesti u web aplikaciju te gdje se mogu izvršavati te koji DOM API-ji smiju pokretati takve skripte. Kako bi se definirale koje skripte odnosno sa kojeg izvora se skripte smiju uvesti i pokretati, definira se CSP konfiguracija koja dozvoljava skripte sa određenih odredišnih adresa. CSP je odlična mjera protiv zaštite od XSS napada, ali neće zaštiti od svake vrste XSS napada. Zbog toga je uvijek važan Anti XSS pristup pisanju koda. [14, 16]

4.5.2. Zaštita od ubrizgavanja

Jedan od najčešćih napada ubrizgavanja je SQL ubrizgavanje. Pošto ovaj napada može biti iskorišten protiv bilo kakve kompleksnije aplikacije, postoji mnogo metoda obrane koje su veoma jednostavne za implementirati te kreirati efektivnu obranu. Validacija korisničkog unosa je prvi način obrane. Normalizacija skupa znakova na primjerice UTF-8, provjera tipa podataka i očekivanih vrijednosti te odbacivanje loših podataka umjesto njihovog čišćenja je prvi korak prema zaštiti. Priprema parametriziranih upita je puno bolji način konstruiranja upita od spajanja vrijednosti. Parametrizirani upiti kreiraju šablonu gdje se označavju dijelovi upita u koje je moguće dodati vrijednost. Vrijednost se u upit dodaje korištenjem metode ovisno o

programskom jeziku. U nastavku je prikazano kreiranje parametriziranog upita i dodavanje vrijednosti u Java programskom jeziku:

```
String upit = "SELECT naziv FROM korisnici WHERE id = ?";
Connection konekcija = DriverManager.getConnection(baza, korisnik, lozinka);
PreparedStatement siguranUpit = konekcija.prepareStatement(upit);
siguranUpit.setInt(1, idKorisnik);
```

Osim korištenja pripremljenih upita, različite SQL baze podataka nude funkcionalnosti kako bi poboljšale sigurnost. Tako većina baza podataka nude metode koje automatski izbjegavaju riskantne znakove za korištenje u upitima. Korištenjem takvih metoda smanjuje vjerojatnost od ubrizgavanje neželjenog sadržaja u upite. [14, 16]

Napad ubrizgavanje XXE se također može veoma jednostavno riješiti prilikom izrade novih aplikacija. Jedna od najboljih metoda je zamijeniti XML format podataka sa JSON formatom ukoliko je to moguće. Razlog tome je taj što je format podataka JSON puno sigurniji, jednostavniji i efikasniji. Razlog tome je taj što je sam XML puno kompleksniji te može prenositi kompleksnije podatke kao što su mediji što naravno sa sobom nosi više vektora za napade. [14]

4.5.3. Zaštita autentifikacije

Sustav autentifikacije jedan je od najvitalnijih sustave web aplikacije te zbog toga mora sadržavati rigoroznu sigurnost. Kako bi se uspostavio dobar sustav autentifikacije, prvo je potrebno koristiti odgovarajuće protokole za prijenos podataka. Unutar cijele web aplikacije potrebno je koristiti HTTPS protokol. Ukoliko se HTTPS koristi samo za autentifikaciju dok za ostatak web aplikacije HTTP, napadač može bez problema doći do sesije korisnika. Lozinke korisnika moraju što je manje vremena moguće prevoditi u normalnom formatu, odnosno što prije moguće potrebno je hashirati lozinku sigurnim sigurnim algoritmom te dodati sol na vrijednost. Dobra ideja je također umjesto implementacije vlastitog sustave autentifikacije koristiti postojeće okvire kao što su OAuth, OAuth2 ili OpenID. Sesija korisnika mora u kolačić biti spremljena sa zastavicom Secure kako bi se spriječio prijenos kolačić preko konekcija koje nisu HTTPS. Sesija također treba ima definirano vrijeme istjecanja te prilikom istjecanja sesije ona mora uništiti i na klijentu i na poslužitelju. Sesija bi također trebala imati pseudo slučajnu vrijednost. Još jedan kritičan sustav autentifikacije je resetiranje lozinke. Sustav resetiranja lozinke mora koristiti također pseudo nasumične tokena za ponovno postavljanje lozinke. Ti tokeni ne smiju u sebi sadržavati vrijednosti kao što su e-mail adresa, korisničko ime ili primarni ključ korisnika. Također bi trebali imati vrijeme koje taj token vrijedi. Dobra praksa je također korisniku prilikom prijave ponuditi pregled informaciji o lokaciji i vremenu prijava te pokušaje neupjelih prijava. Prilikom mijenjanja osobnih podataka, potrebno je odjaviti korisnika te tražiti

ponovnu prijavu. Ukoliko korisnik ima veći broj neuspjelih prijava, potrebno je zaključati račun ili dodati sustav koji će provjeriti da li su zahtjevi za prijavi dani od strane korisnika ili računala (napadi grubom silom). Takve provjere mogu se realizirati implementiranje CAPTCHA (eng. Completely Automated Public Turing test to tell Computers and Humans) testova. [16]

4.5.4. Zaštita od CSRF napada

Zaštita od CSRF napada može se riješiti korištenje tzv. anti CSRF tokena. Zaštita CSRF tokenima je najčešće korištena obrana od CSRF napada jer se može implementirati na relativno jednostavan način. Tokeni za CSRF zaštitu moraju biti nepredvidljivi i jedinstveni kako bi zaštita bila efektivna. Poslužitelj klijentu šalje jedinstveni token kriptografski generiran. Token se najčešće generira jednom po sesiji ili po zahtjevu te je uvijek jedinstven. Nakon što klijent pošalje zahtjev poslužitelju, zajedno uz zahtjev šalje se i token koji je spremljen u kolačiće. Kada poslužitelj zaprimi zahtjev, verificira se vrijednost tokena. Ukoliko verifikacija tokena ne uspije, zahtjev se bilježi u dnevnik rada, ali se ne provodi. Ova zaštita je veoma efektivna zato jer prilikom CSRF napada zbog SOP pravila, zahtjev sa drugog porijekla odnosno web aplikacije, ne može čitati kolačiće niti poslati ispravan zahtjev. Kako bi napadač uspješno proveo napad, potreban mu je CSRF od specifičnog korisnika. Ova zaštita također se može implementira i na REST arhitekturu koja ne smije zadržavati nikakvo stanje. [14]

5. Uzorci dizajna sigurnosti i privatnosti

Uzorci dizajna sigurnosti i privatnosti jedan su od načina kojim se može implementirati dodatna razina sigurnosti prilikom izrade web aplikacija. Uzorci dizajna su općenita rješenja koja se mogu ponovno koristiti za uobičajene probleme dizajna softvera sa kojima se susrećemo tokom razvoja. Uzorci dizajna su također najbolje prakse i dokazana rješenja koju su se tokom vremena razvila kako bi odgovorila na specifične i ponavljajuće probleme. Svaki uzorak opisuje problem koji se uvijek iznova pojavljuje u okruženja te je sami cilj uzorka dizajna da se može koristiti koliko god puta je to potrebno. Generalno gledano svaki uzorak ima četiri glavna elementa [37]:

- Naziv uzorka – opisuje problem, rješenje i posljedice u jednoj ili dvije riječi te proširuje rječnik dizajna. Naziv uzorka pomaže prilikom dizajniranja i komunikacije sa drugima;
- Problem – opisuje problem i kontekst problema te kada se uzorak može primijeniti. Može opisivati specifični problem dizajna ili strukturu klasa i objekata koje su nefleksibilne. Problem može uključivati i listu uvjeta koji moraju biti ispunjeni kako bi imalo smisla koristiti uzorak;
- Rješenje – opisuje elemente od kojih se sastoji uzorak te njihove odnose, suradnje i odgovornosti. Daje apstraktni opis problema dizajna i kako se raspoređuju elementi koji rješavaju problem;
- Posljedice – opisuje rezultate i kompromise koji se javljaju primjenom uzorka dizajna. Posljedice pomažu da se procijeni i olakša odluka o korištenju uzorka. Ponovna upotreba često je ključna u dizajnu te zbog toga posljedice uzorka uključuju utjecaj na fleksibilnost, proširivost ili prenosivost sustava.

5.1. Uzorci sigurnosti

Uzorci sigurnosti opisuju rješenje problema kontroliranja setova specifičnih prijetni koristeći neku vrstu sigurnosnog mehanizma definiranog u danom kontekstu. Kao cilj, uzorci sigurnosti trebali bi eliminirati ranjivosti koji su se pojavili te ublažiti posljedice ranjivosti. Prema glavnim elementima uzoraka, posljedice indiciraju koliko dobro je riješen problem nekog napada. Za razliku od uzoraka dizajna koji se primjenjuju na razini dizajna softvera, uzorci sigurnosti se primjenjuju na raznim razinama od arhitektonske razine pa do implementacijske razine te daju smjernice kako implementirati metode u sustave. [34, 38]

Smanjivanje cijene održavanje i rizika of sigurnosnih propusta je jedan od glavnih ciljeva tokom razvoja softvera. Postoje mnoge najbolje prakse za poboljšanje sigurnosti web aplikacija ali su često implementacijski specifične i nisu lako ponovno koristive. Sigurnosni uzorci igraju ključnu ulogu u rješavanju sigurnosnih pitanja u različitim fazama razvoja od arhitektonskog

dizajna do same implementacije. Uzroci su nastali generaliziranjem i katalogiziranjem postojećih najboljih praksi što olakšava usvajanje i primjenu. Sigurnosni uzorci se isto kao i standardni uzorci mogu ponovno koristiti te je sa njime moguće učinkovitije učinite sustave sigurnijima što u konačnici rezultira smanjivanjem troškova i rizika povezanih sa sigurnosnim propustima. [34]

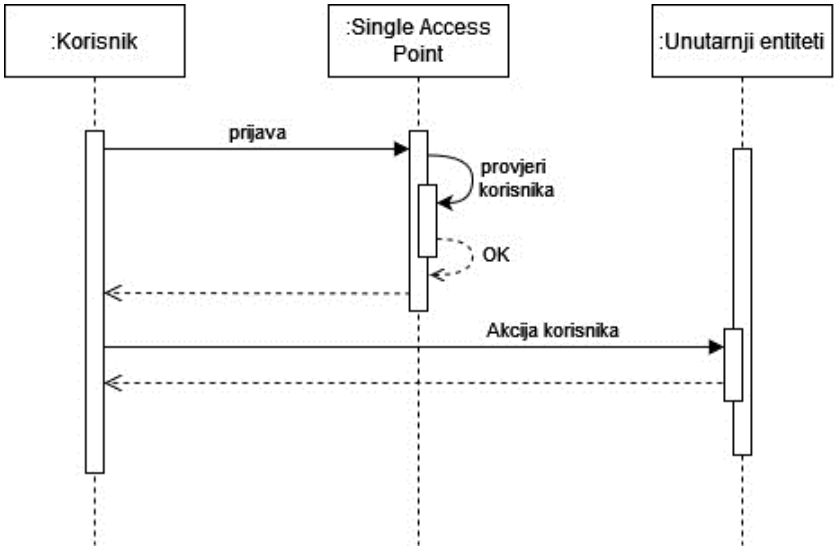
Uzorci sigurnosti nude opće smjernice kako bi se pokušalo spriječiti uvođenje ranjivosti u kod ili smanjile posljedice ranjivosti. Uzorci sigurnosti mogu se podijeliti na razine [34, 38]:

- Uzorci na razini arhitekture – uzorci koji se fokusiraju na raspodjelu odgovornosti i interakciju između različitih komponenti sustava na visokoj razini. Opisuju koncepte arhitekture koji se odnose na sigurnost te primjenjuju sigurnost na cijeli sustav;
- Uzorci na razini dizajna – bave se internim problemima dizajna pojedinačnih komponenti sustava visoke razina;
- Uzorci na razini implementacije – bave se sigurnosnim problemima niske razine i primjenjivi su na implementaciju specifičnih funkcija i metoda u sustavu.

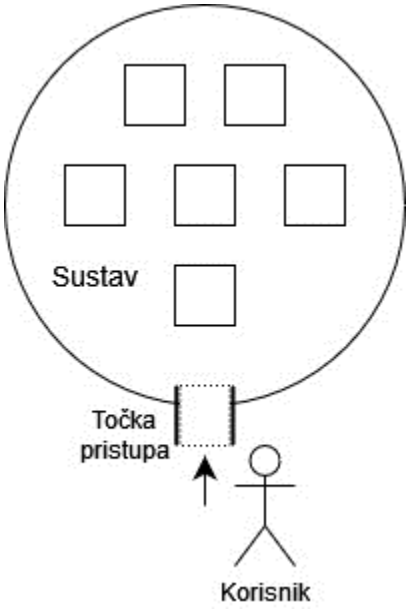
Kao što se uzorci sigurnosti mogu podijeliti na razinu, mogu se slično podijeliti i na namjenu uzorka prema pravim uzorcima dizajna a to su uzorci za kreiranje, uzorci za ponašanje i uzorci za strukturu. [37, 39] U nastavku će biti konkretnije obrađeni neki od uzoraka sigurnosti na razini arhitekture, dizajna i implementacije.

5.1.1. Single Access Point

Single Access Point (u nastavku SAP) je uzorak koji pruža jedinstveno sučelje za komunikaciju sa vanjskim entitetima, poboljšavajući kontrolu i nadzor. Uzorak se bavi izazovom zaštite sustava od vanjski napada, minimiziranjem skrivenih stražnjih vrata i nedosljednih sigurnosnih politika. Usmjeravanjem cjelokupnog dolaznog prometa kroz jednu točku, *Single Access Point* uzorak olakšava praćenje i bilježi informacije o dolaznim subjetika te provjerava i određuje prava. Ovaj uzorak se može navoditi kao uzorak za strukturu te se primjenjuje na arhitekturalnoj razini. Na sljedećim slikama prikazan je sekvencijski dijagram i gruba shema sustava. [39]



Slika 5: Sekvencijski dijagram uzorka SAP (Izvor: vlastita izrada, prema: [40])



Slika 6: Prikaz sustava sa korištenim SAP uzorkom (Izvor: vlastita izrada, prema: [40])

Sudionici

U ovom uzorku sudjeluju [39]:

- Vanjski entiteti – komponente koje se nalaze izvan sustava te kontaktiraju točku pristupa kako bi mogle komunicirati sa unutarnjim entitetima
- Unutarnji entiteti – sve komponente koje se nalaze unutar sustava
- *Single Access Point* – klasa ili mehanizam koji daje ili odbija pristup vanjskim entitetima

Implementacija

Kako bi se implementirao uzorak sigurnosti SAP potrebno je pratiti sljedeće korake [40]:

- Definirati sigurnosnu politiku za sustav uključujući odnose povjerenja između internih podsustava;
- Uspostaviti istaknutu poziciju za jedinstvenu pristupnu točku ili je prikazati transparentnom za legitimne korisnike;
- Opcionalno se može implementirati autentifikacija i autorizacija odnosno za inicijalizaciju sesija i postavljanje parametara;
- Osigurati granice sustava kako bi se zatvorila potencijalna stražnja vrata (eng. backdoor).

Posljedice

Primjenom ovog uzorka dolaze neke prednosti i nedostatci. Prednosti primjene uzorka su taj što postoji jedna ulazna točka za pristup sustavu, pojednostavljeno je unutarnja struktura bez suvišnih provjera autorizacije, može se primjeniti na više razina apstrakcije te je poboljšana sigurnost pošto postoji samo jedna pristupna točka. Potencijalni problemi primjenom uzorka su da sustav može postati glomazan ili teže upotrebljiv zbog samo jedne pristupne točke, mogući sporiji sustav na strani klijenta zbog ulaznih provjera te jedna točka postaje jedinstvena točka kvara koja potecijalno ugrožava upotrebljivost i sigurnost sustava. [40]

5.1.2. Secure Logger

Secure Logger je implementacijski sigurnosni uzorak ponašanja. Uzorak štiti podatke dnevnika rada sustava od iskorištavanja od strane napadača. Osigurava cjelovitost sustava zapisivanje te otežava neovlašteni pristup i manipulaciju podacima. Implementacijom uzorka dobivamo funkcionalnost zapisivanja raznih podataka o akcijama sustava i korisnika te se mogu detektirati i dijagnosticirati napadi na sustav. Na sljedećoj slici prikazana je struktura i odnos između sudionika uzorka *Secure Logger*. [34]

Slika 7: Struktura uzorka *Secure Logger* (Izvor: vlastita izrada, prema: [34])

Sudionici

U ovom uzorku sudjeluju [34]:

- Aplikacija – glavna aplikacija koja generira podatke koje obrađuje i sprema neki sustav za dnevnik;
- *Secure Logger* – odgovoran za spremanja podataka dnevnika na način da je neautoriziranim korisnicima teško ili nemoguće pristupiti podacima dnevnika;
- Čitač dnevnika – specijalizirani mehanizam za čitanje sadržaja dnevnika pohranjenog u sigurnom sustavu zapisivanja. Standardni čitači tekstualnih sadržaja neće moći pročitati podatke zbog zaštićenog formata već ovlašteni korisnici mogu pristupiti podacima putem ugrađenog mehanizma;
- Preglednik dnevnika – autorizirani korisnika koji može čitati podatke dnevnika pomoću neke vrste aplikacije za pregled dnevnika.

Implementacija

Za implementaciju uzorka *Secure Logger* potrebno je nekoliko koraka [34]:

- Odabir sigurni podsustav za bilježenje u dnevnik (npr. SmartInspect, CLogIt);
- Uvijek koristiti odabrani podsustav za sigurno zapisivanje podataka u dnevnik sustava, izbjegavajući bilo kakav izlaz, standardni izlaz ili standardnu pogrešku.
- Dodatno se može kreirati apstraktno sučelje kako bi podsustav bilježenje ostao transparentan I kako bi detalji interakcije sa podsustavom za zapisivanje ostali sakriveni. Također se može iskoristiti uzorak dizajna *Abstract Factory* za generiranje objekta koji zapisuje podatke u dnevnik rda kako bi se jednostavnije prebacivalo između različitih podsustava za zapisivanje.

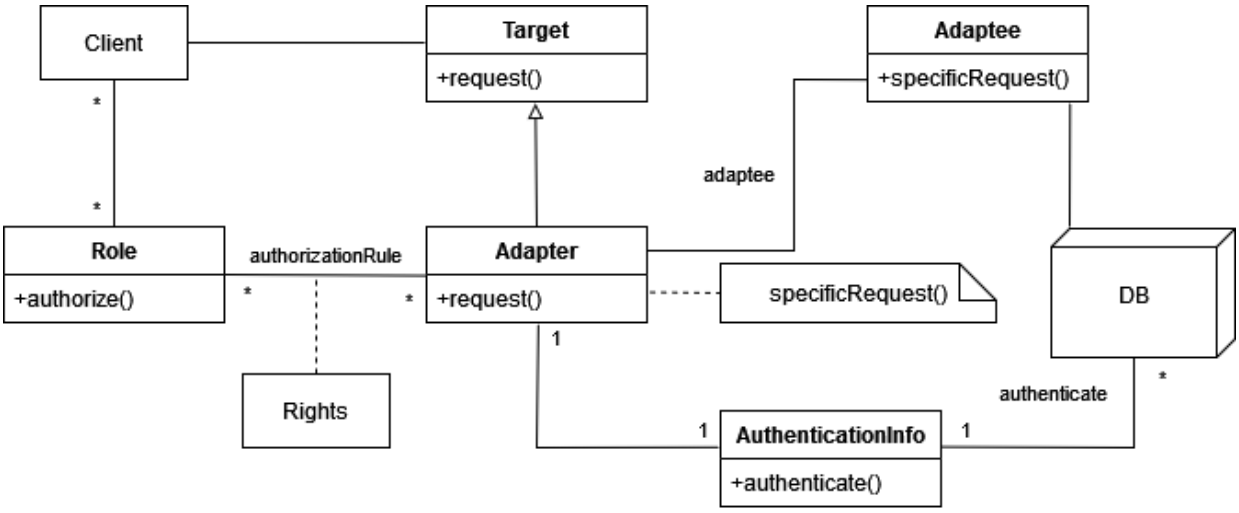
Posljedice

Posljedice ovog uzorka su ograničena vidljivost za napadače. Ukoliko napadač dobije pristup podacima dnevnika kojima upravlja sustav sigurnog zapisivanje, imat će ograničenu ili nikakvu vidljivost podataka dnevnika. To će napadača kasnije sprječiti za planiranje sofisticiranijih napada na sustav. Osim toga ovlašteni korisnici mogu otkriti sve pokušaje napadača da modificiraju podatke dnevnika. Otkriće pomaže u održavanju cjelovitosti i pouzdanosti zabilježenih informacija. [34]

5.1.3. Secure Adapter

Secure Adapter još poznat i kao *Secure Wrapper* je uzorak dizajna strukture koji pretvara sučelje postojeće klase u prikladnije sučelje istovremeno održavajući sigurnost adaptiranog entiteta. Primjerice pretvaranje obične poruke u XML poruku za kompleksnije transakcije.

Poruka sadrži osjetljive podatke te je ovakva slučaj odličan primjer korištenja uzorka *Secure Adapter* jer osigurava kompatibilnost između nekompatibilnih sučelja te štiti osjetljive podatke od neovlaštenog pristupa. Prijetnje adapteru mogu se također ublažiti autentifikacijom, kontrolom pristupa i korištenjem sigurnog komunikacijskog kanala. Na sljedećoj slici prikazan je dijagram klasa uzorka *Secure Adapter*. Prema uzorku dizajna *Adapter* dodane su još neke klase i relacije. Dodana je kontrola pristupa bazirana na ulogama i odgovarajućem skupu autorizacijskih pravila. Svi zahtjevi prema `Adapter` moraju biti autorizirani odnosno klijent mora imati pravo da pošalje zahtjev. [38]



Slika 8: Dijagram klasa uzorka Secure Adapter (Izvor: vlastita izrada, prema: [38])

Sudionici

U ovom uzorku sudjeluju [38]:

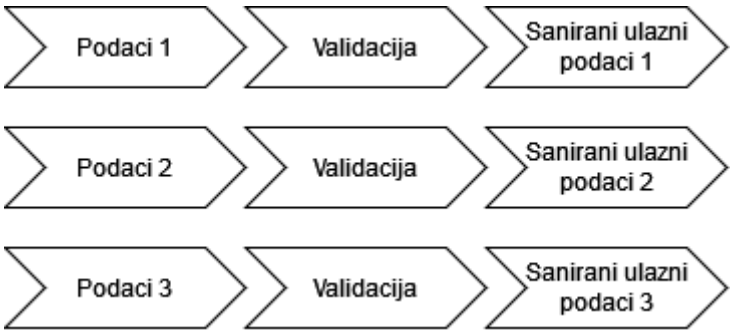
- *Client* – surađuje s objektima koje odgovaraju sučelju *Target*;
- *Target* – sučelje kojem *Client* pristupa;
- *Adapter* – posredna komponta koje prilagođava sučelje sustava kako bi odgovaralo sučelju klijenta odnosno *Adaptee* na *Target*;
- *Adaptee* – definira sučelje koje treba adaptiranje;
- *Role* – određuje razinu dopuštenja *Client*-a;
- *AuthenticationInfo* – definira sučelje za podatke koji se odnose na provjeru autentičnosti;
- *Right* – definira sučelje koje predstavlja određene dozvole za autorizaciju.

Posljedice

Posljedice ovog uzorka su da se baza podataka i klijent mogu autentificirati te se autorizacija može primjeniti na zahtjeve korisnika. Može se koristiti kriptografija kako bi se kriptirale poruke interakcije između sudionika u uzorku [38].

5.1.4. Input Validation

Input Validation je sigurnosi uzorak na razini implementacije koji osigurava integritet sustava provjerom podataka iz nepouzdanih izvora. Kao što je u prethodnim poglavljima objašnjeno, nepotvrđeni korisnički unosi mogu dovesti do teških sigurnosnih incidenata na različitim mjestima kao što su ubrizgavanja SQL-a, napada skriptama i sl. Uzorak *Input Validation* primjenjuje se na bilo kakvi softver koji prihvaća podatke iz nepouzdanih izvora, osiguravajući aplikaciju od potencijalnih sigurnosnih prijetnji i incidenata te neovlaštenog pristupa. Iako se često validacija unešenih podataka provodi na strani korisnika, ne treba se oslanjati na nju već je dobra praksa provesti provjeru i na strani poslužitelja. [34]



Slika 9: Struktura uzorka Input Validation (Izvor: vlastita izrada, prema: [34])

Sudionici

U ovom uzorku sudjeluju [34]:

- Aplikacija ili sustav koji prihvaća podatke – prihvaća i validira odnosno sanira podatke;
- Vanjski resursi koji daju podatke – vanjski resursi mogu biti datoteke, ljudi, baze podataka.

Posljedice

Posljedice primjene ovog uzorka na sve vanjske podatke iz ulaza su dodatno sigurnost cijelog sustava. Napadi koji se oslanjaju na loše sanirane ulaze su u potpunosti uklonjeni. Povećava se pouzdanost cijelog sustava jer se svi ulazi ponašaju na očekivan i predvidljiv način. Negativna posljedica je da validacija ulaza može smanjiti performanse sustava te količina vremena koja je potrebno da se pronađe i sanira svaki ulazni tok podataka. [34]

Implementacija

Generalno gledano proces implementacije sigurnosnog uzorka *Input Validation* može se provesti sljedećim koracima [34]:

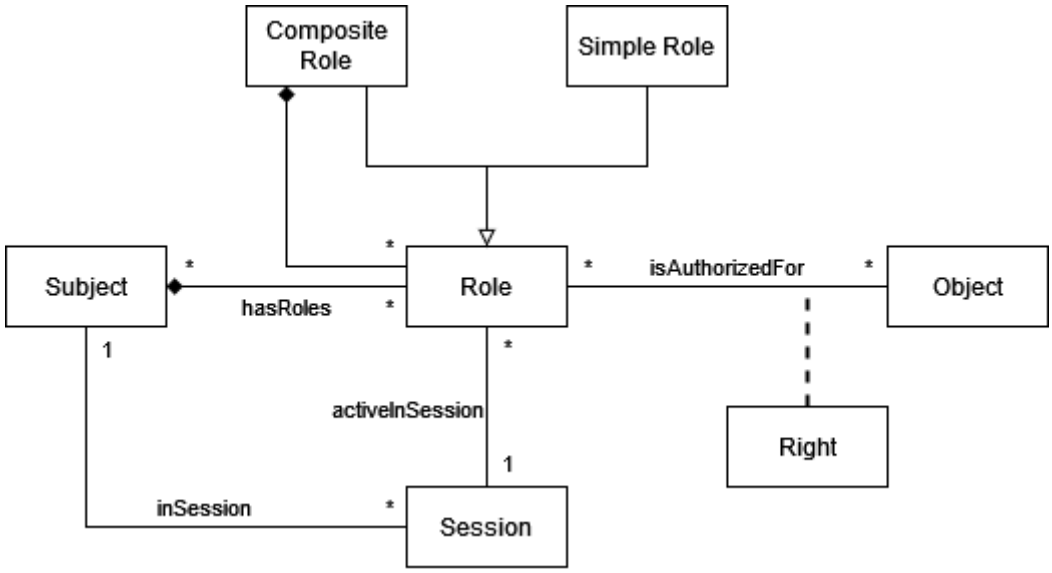
1. Identifikacija svih izvora unosa – izvori kao što su korisnički unosi, datoteke, baze podataka i mrežni promet;

2. Identifikacija svih čitanja izvora unosa – identificiranje mjesta na kojima se podaci početno čitaju u sustavu;
3. Definiranje kriterija za ispravne podatke – uspostaviti pravila provjere temeljene na vrsti podataka i namjeravanoj upotrebi;
4. Određivanje postupanja sa nevažećim podacima – definiranje odgovora sustava na nevažeći unos u rasponu od upozoravanja do ponovnog zahtjeva za podacima;
5. Implementacija koda za provjeru i rukovanje nevažećim podacima – implementacija koda za provjeru valjanosti podataka osiguravajući da su u skladu sa kriterijima.

5.1.5. Session-Based Role-Based Access Control

Uzorak sigurnosti Session-Base Role-Based Access Control je arhitekturni sigurnosni uzorak koji odobrava pristup resursima na temelju uloga subjekta te ograničava prava subjekata tijekom sesije. Primjenjuje se na okruženja u kojima pristup resursima mora biti kontroliran odnosno korisnici su kategorizirani prema svojim zadacima ili mogućnostima te su prava pristupa dodijeljena ulogama. Uzrok se za implementaciju oslonjava na sesije te sesije upravljaju upotrebom uloga I provode ekskluziju uloga tijekom izvođenja. [38]

Na sljedećoj slici prikazan je dijagram klasa uzorka. Između subjekta i objekta stoji uloga. Uloga drži sva autorizacijska prava koja korisnik ima te se ponaša kao kontekst pokretanja. Unutar sesije samo neke uloge mogu izvršavati namjerene zadatke. Uloge se mogu sasaviti pomoću uzorka dizajna *Composite* u kojem više razine uloge stječu prava uloga niže razine. [38]



Slika 10: Dijagram klasa uzorka *Session-Based Role-Based Access Control*

(Izvor: vlastita izrada, prema: [38])

Posljedice

Posljedice korištenja ovog uzorka sigurnosti su mogućnost uključivanja više uloga za određene subjekte u sesiju. Time dolazi do funkcionalne fleksibilnosti dopuštajući korisnicima da aktiviraju više sesija istovremena za zadatke koji zahtijevaju višestruke uloge. Precizna prava mogu se dodijeliti ulogama kako bi se provela politika koja se mora poštivati te sesije mogu isključiti uloge koje krše politiku. Neki nedostaci korištenja uzorka su dodatno kompleksnost definiranja uloga koje mogu biti korištene zajedno i obrnuto te zbunjenost korisnika koji imaju više uloga. [38]

Implementacija

Implementacija uzorka može se provesti u sljedećim koracima [38]:

1. Definiranje uloga koje sustav mora sadržavati prema korisničkim funkcijama i zadacima;
2. Kreiranje liste nekompatibilnih uloga te koristiti tu listu kad se kreira sesija;
3. Određivanje broja uloga koje mogu biti aktivne u jednoj sesiji;
4. Otvaranjem sesije korisnika, korisnik mora odabrati uloge koje namjerava koristiti te sustav otvara odgovarajuću sesiju.

5.2. Uzorci privatnosti

Problemi vezani uz privatnost složeni su i subjektivni pa ih je teško pretočiti u tehničke artefakte tijek primjene privatnosti prema dizajnu. Inspirirani uzorcima dizajna, uzorci privatnosti imaju za cilj standardizirati jezik za tehnologije koje čuvaju privatnost, dokumentirati uobičajena rješenja za probleme privatnosti te pomoći arhitektima da se sa problemima privatnosti suoče u ranoj fazi razvoja aplikacije. Uzroci transparentnosti privatnosti igraju ključnu ulogu u postizanju odgovarajuće razine transparentnosti privatnosti, pružajući pravu količinu i vrstu informacija o osobnim podacima korisnika na prikladan prezentacijski način. Implementacija uzoraka privatnosti korisniku mogu ponuditi sveobuhvatni i prilagođeni pristup pitanjima privatnosti, podacima i razumijevanje. Uzorci privatnosti kategorizirani su u četiri kategorije [35, 41]:

- Generičke informacije o privatnosti (eng. Generic Privacy Information) – pružaju informacije o namjerama pružatelja sustava u vezi s osobnim podacima korisnika;
- Uvid u podatke u stvarnom vremenu (eng. Real-time Data Insight) – uvid korisnika u pohranu i rukovanje stvarnim podacima u stvarnom vremenu;
- Svijest o privatnosti (eng. Privacy Awarness) – povećavaju svijest korisnika o mogućim rizicima privatnosti prilikom dijeljenja osobnih podataka;
- Oznaka privatnosti (eng. Privacy Mark) – potvrda tijela za zaštitu privatnosti koja pokazuje da su ispunjeni određeni kriteriji od strane pružatelja aplikacije.

5.2.1. Prikaz politike privatnosti

Politike privatnosti spada pod kategoriju generičkih informacija o privatnost. To je uzorak privatnosti koji je ključan za voditelje obrade podataka kako bi informirali korisnike o aktivnostima obrade i pridržavali se zakona. Međutim, postizanje ravnoteže između pristupačnosti i pravne sveobuhvatnosti može biti izazovno. Korisnici obično izbjegavaju čitanje dugih i složenih pravila, ali moraju razumjeti relevantne rizike. Kontrolori moraju osigurati da su korisnici dobro informirani prije traženja pristanka. Prilikom traženja korisničkih podataka, ključna je jasnoća u vezi potrebnih podataka, njihove svrhe i tko ih traži. Korisnici preferiraju sažete, relevantne informacije umjesto preopterećenih detalja, dok kontrolori žele izgraditi povjerenje transparentnošću u pogledu potencijalnih rizika [41].

Implementacija

Traženjem osobnih podataka, prije traženja privole korisnika, potrebno je jasno navesti koje informacije su potrebne, od koga, u koje svrhe i na koji način. Preporučeni slojeviti format uključuje tri razine [41]:

- kratka obavijest – pruža ključne pojedinosti za razumijevanje korisnika;
- sažeta obavijest – uključuje sažetak bitnih informacija povezanih s GDPR-om, a
- puna obavijest – predstavlja sve pojedinosti koje zahtijeva GDPR kako bi se osigurala usklađenost.

Posljedice

Podsjećanje korisnika na implikacije dijeljenja njihovih informacija pomaže im da razmotre svoje izbore vezane za osobne podatke. Međutim, česti podsjetnici mogu dovesti do zamora korisnika i desenzibilizacije prema pravilima privatnosti. Balansiranje vidljivosti i ozbiljnosti informacija je ključno. [41]

5.2.2. Tablica osobnih podataka

Tablica osobnih podataka (eng. Personal Data Table) je uzorak privatnosti koji korisnicima daje uvid u podatke u stvarnom vremenu. Namjera je pružiti podatkovnu tablicu koja sadrži osobne podatke korisnika, osiguravajući korisnicima transparentnost o tome kako se njihovi podaci obrađuju. Ovo rješenje posebno je korisno za informatički pismene korisnike koji žele detaljne informacije o svojim osobnim podacima. Podaci moraju biti sigurno pohranjeni i prikazani u tabličnom obliku u korisničkom sučelju. Tablica bi trebala prikazivati prikupljene podatke, događaje pristupa, događaje otkrivanja i još mnogo toga, a istovremeno omogućiti kategorizaciju za laku navigaciju. Cilj je povećati povjerenje korisnika kroz transparentnost uz očuvanje sigurnosti i dostupnosti podataka. [35, 41]

Implementacija

Kako bi se uspješno implementirao navedeni uzorak privatnosti potrebno je korisnicima ponuditi sučelje koje prikazuje njihove podatke u tablici. Tablica bi trebala sadržavati četiri stupca sa vrstom podatka, podatak, datum spremanja te popis osoba koje su pristupile tom podatku. Za potpunu transparentnost također je prikladno prikazati razloge zašto i kako se podaci koriste te koji pojedinci mogu pristupiti osobnih podacima korisnika. Tablica može prikazati sve podatke istovremeno ili ih se može kategorizirati. [35, 41]

Posljedice

Pozitivne posljedice odnosno prednosti tablice osobnih podataka su [41]:

- Pristup u stvarnom vremenu – korisnici mogu vidjeti osobne podatke u stvarnom vremenu;
- Strukturirani detalji – tablica daje prikaz osobnih podataka u strukturiranom formatu;
- Identifikacija pogrešaka – korisnici mogu uočiti pogreške u svojim podacima te na taj način poboljšati kvalitetu podataka putem ispravljanja;
- Sigurnost i dostupnost – podaci ostaju sigurni, ali i dostupni te pogodni za korisnika;
- Poboljšano razumijevanje – korisnici stječu uvid u svoje osobne podatke te uvid u korištenje podataka što dovodi do poboljšane kontrole podataka;

Neke negativne posljedice ovog uzorka su [41]:

- Sveobuhvatno bilježenje – sve aktivnosti korisničkih podataka moraju se bilježiti, što potencijalno dovodi do zabrinutosti zbog privatnosti;
- Preopterećenost informacijama – korisnici mogu biti preopterećeni količinom podataka, što potencijalno može dovesti do smanjene upotrebe sustava;
- Ograničenje kontrole podataka – Neki korisnici možda žele analizirati podatke na svojim sustavima, što ovaj obrazac ne olakšava;
- Napomena u stvarnom vremenu - Iako je cilj ovog uzorka pružiti uvid u stvarnom vremenu, trenutna praksa često uključuje formalne zahtjeve i razdoblja čekanja za pristup podacima;

5.2.3. Nadzorna ploča privatnosti

Nadzorna ploča privatnosti (eng. Privacy Dashboard) rješava izazov transparentnog komuniciranja i upravljanja obrađenim osobnim podacima. Ovaj uzorak omogućuje korisnicima da razumiju opseg prikupljenih podataka, pružajući sažete primjere, modele, vizualne prikaze i statistiku. To korisnicima omogućuje donošenje informiranih odluka o dijeljenju podataka, sprječavajući rizike prekomjernog dijeljenja i privatnosti. Nadzorna ploča nudi centralizirani

prostor za korisnike za upravljanje svojim podacima, potičući angažman putem kontrole za brisanje i ispravljanje. Ovo rješenje uspostavlja ravnotežu između svijesti korisnika, kontrole i besprijekornog iskustva. [41]

Implementacija

Implementacija ovog uzorka uključuje ponudu različitih korisničkih pristupa njihovim relevantim podacima. Pristup može biti putenje filtriranja, informativnih izvješća i obavijesti. Voditelji obrade imaju za cilj informirati korisnike o prikupljenim i agregiranim osobnim podacima, a posebno o podacima koji se mijenjaju, prolaze kroz neočekivanu obradu, lako se zaboravljaju ili ih korisnici mogu ispraviti i izbrisati. [41]

Posljedice

Potrebna je pažljiva implementacije jer otkrivanje korisničkih podataka na nadzornoj ploči može dovesti novih problema vezanih uz privatnost. Presudno je spriječiti pristup osjetljivim podacima bilo kome osim korisniku. Primjerice, prikazivanje povijesti pretraživanja vezane uz kolačiće moglo bi izložiti pregledavanje jednog člana obitelji drugome na zajedničkom uređaju. Štoviše, iako povezivanje svih podataka o korištenju s računom može poboljšati cjelovitost nadzorne ploče, to može dovesti do davanja podataka koji nisu izvorno povezani s korisničkim računom. [41]

6. Izrada web aplikacije

U praktičnom dijelu rada, izrađena je web aplikacija koja stavlja sigurnost i privatnost korisnika na prvo mjesto. Iskorištene su tehnologije, principi te uzorci sigurnosti i privatnosti koji bi podigli razinu sigurnosti web aplikacije na višu razinu. Stoga će u nadolazećim podpoglavljima detaljno biti opisana domena web aplikacije i njezine funkcionalnosti. Zatim će biti prikazan i raspisan ERA (eng. Entity Relationship Model) model podataka zajedno sa svim relacijama, atributima te vezama između istih. Nakon što je poznata domena web aplikacije, funkcionalnosti i model podataka, predstavit će se korištene tehnologije, razlog odabira tih tehnologija, pristup razvoju i implementaciji web aplikacije te principi i metode koji su korišteni prilikom razvoja. Shodno tome, rastumačit će se implementacija na strani poslužitelja, arhitektura na određenim razinama, te aspekti i uzorci dizajna koji utječu na sigurnost i privatnost korisnika prilikom korištenja web aplikacije ili korištenja poslužitelja. Na isti način bit će prikazana implementacija na razini klijenta te aspekti i sigurnosni uzorci koji su korišteni. Razvoj i implementacija web aplikacije sa fokusom na sigurnost su vrlo zahtjevni te od strane arhitekta i programera zahtijevaju razmišljanje i planiranje daleko unaprijed. Takve vještine dobivaju se s vremenom i iskustvom. Bit će prikazani problemi i izazovi koji su se pojavljivali tijekom cijelog životnog ciklusa razvoja web aplikacije, od planiranja i strategije, izrade arhitekture i odabira tehnologija pa do dizajna i samog razvoja web aplikacije.

6.1. Opis web aplikacije

Tema praktičnog dijela rada je web aplikacija "Odmaralica". "Odmaralica" je web aplikacija putničke agencije koja korisnicima omogućuje dodavanje, pretraživanje i rezerviranje smještaja za odmor. Aplikacija je podijeljena prema korisničkim ulogama, koje su redom: neregistrirani korisnik, registrirani korisnik, iznajmljivač, moderator i administrator. Svaka od korisničkih uloga ima svoj set mogućnosti i prava koja se proširuju prema većoj razini uloge, odnosno administrator ima sve mogućnosti kao i ostale uloge unutar aplikacije.

Glavni fokus i funkcionalnost oko koje se vrti cijela aplikacija je mogućnost rezerviranja smještaja za odmor. Kako bi ta funkcionalnost bila moguća, bilo je potrebno implementirati set raznih mogućnosti na način da je aplikacija potpuno dinamička te se zadaci svih uloga aplikacije mogu izvršavati unutar jednog web aplikacijskog okvira. Korisnici prilikom traženja smještaja najčešće imaju neke zahtjeve koje žele da njihov smještaj zadovoljava. Ti zahtjevi mogu biti u smislu pogodnosti koje objekt odnosno iznajmljivač nudi korisniku, vrste objekta gdje korisnik želi odsjesti, atributa koje smještaj sadrži ili udaljenosti od nekih ključnih točaka. Osim navedenih, korisnici pretražuju smještaje i prema specifičnim lokacijama u kojima žele boraviti.

Uzevši navedeno u obzir, potrebno je na poprilično kompleksan način povezati sve ključne faktore u jednu cjelinu kako bi korisnik korištenje web aplikacije dobio najbolje moguće korisničko iskustvo.

Glavna stranica web aplikacije je tražilica koja omogućava korisniku da pretražuje i filtrira smještaje prema lokaciji, sadržaju, ocjeni drugih korisnika, cijeni i dostupnosti u odabranom periodu. Korisnik zatim ima mogućnost pregledavanja određenog objekta i svih njegovih atributa zajedno s galerijom objekta. Unutar pregleda objekta ponuđene su smještajne jedinice koje nisu zauzete u odabranom terminu. Odabirom smještajne jedinice prikazuju se atributi, detalji i galerija smještajne jedinice zajedno s informacijama o prijavi, odjavi, individualnoj cijeni po svakoj noći te ukupnoj cijeni za cijeli boravak. Ukoliko se korisnik odluči da želi rezervirati smještaj, ta rezervacija će biti dostupna registriranom korisniku unutar njegovog profila pod sekcijom rezervacija. Unutar sekcije korisničkih rezervacija, korisnik može odgoditi rezervaciju ako je rezervacija nadolazeća, ocijeniti smještaj ukoliko je rezervacija u prošlosti te dobiti pregled cijene odnosno računa za svaku rezervaciju.

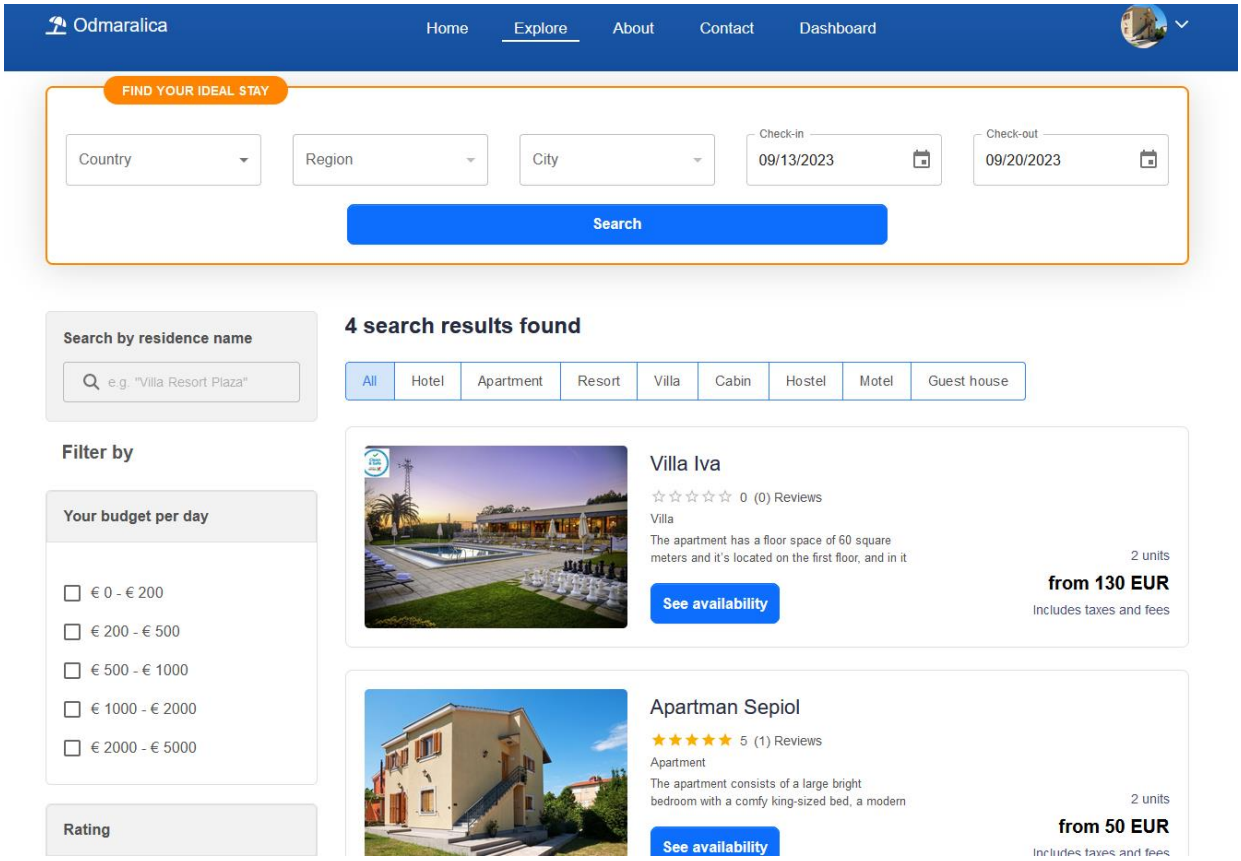
Za uspješnu rezervaciju smještajne jedinice, korisnik mora biti prijavljen. Korisnik se može registrirati, pri čemu se prilikom registracije šalje aktivacijski e-mail. Nakon aktiviranja računa, korisnik se može prijaviti. Osim direktnog registriranja, korisnik se može registrirati ili prijaviti u aplikaciju korištenjem svojeg Google računa. Svi korisnički podaci, zajedno s avatarom korisnika, mogu se mijenjati na stranici profila.

Kako bi se zadovoljio uvjet glavne funkcionalnosti web aplikacije, potrebna je uloga iznajmljivača koji može svoj objekt i smještajne jedinice oglašavati na iznajmljivanje. Unutar web aplikacije postoji nadzorna ploča, odnosno stranica na kojoj iznajmljivači mogu obavljati sve potrebne akcije vezane uz svoje objekte. Prilikom kreiranja objekta, iznajmljivač mora odabrati tip objekta, naziv, lokaciju odnosno adresu, opis objekta te dodati slike u galeriju objekta. Iznajmljivač može imati više objekata, pa postoji tablica u kojoj su prikazani svi objekti iznajmljivača zajedno s akcijskim elementima koji omogućuju uređivanje i brisanje objekta. Stranica za uređivanje objekta sadržava mnoge mogućnosti, kao što su ažuriranje detalja objekta, adrese i galerije. Unutar uređivanja objekta postoji zaseban prozor gdje iznajmljivač ima mogućnost kreiranja smještajnih jedinica unutar objekta. Svaka smještajna jedinica također ima svoju galeriju kako bi korisnik mogao vidjeti kako sama jedinica izgleda. Za svaku smještajnu jedinicu moguće je vidjeti rezervacije korisnika, no prije nego što se smještajna jedinica može rezervirati, potrebno je definirati cijenu. Iznajmljivač definira cijenu smještaja u određenom periodu, odnosno u različitim vremenskim razdobljima, pri čemu cijena može biti različita. Kako bi objekt i smještajne jedinice bile uspješno vidljive na stranici pretraživanja, objekt mora imati

minimalno jednu sliku u galeriji, te jedna smještajna jedinica koja mora imati jednu sliku u galeriji te definiranu cijenu za traženi period.

Osim navedenih uvjeta, postoji još jedan uvjet na koji iznajmljivač nema direktnog utjecaja. Bilo koji novokreirani objekt mora biti odobren od strane moderatora. Moderator ima dužnost pregledati sadržaj objekta i eventualno kontaktirati iznajmljivača ako su navedene pogrešne informacije. Nakon što je objekt odobren, moderator ga javno objavljuje. Moderator također ima mogućnost dodavanja novih objekata u ime iznajmljivača te može dodijeliti korisnicima ulogu iznajmljivača.

Zadnja uloga unutar web aplikacije je administrator. Administrator može kreirati, uređivati ili brisati korisničke račune. Također može pristupiti lokacijskim objektima, državama, regijama i gradovima te uređivati ili dodavati nove. Svaka akcija koja se događa unutar aplikacije bilježi se u dnevniku rada. Dnevnik rada dostupan je samo administratoru web aplikacije.

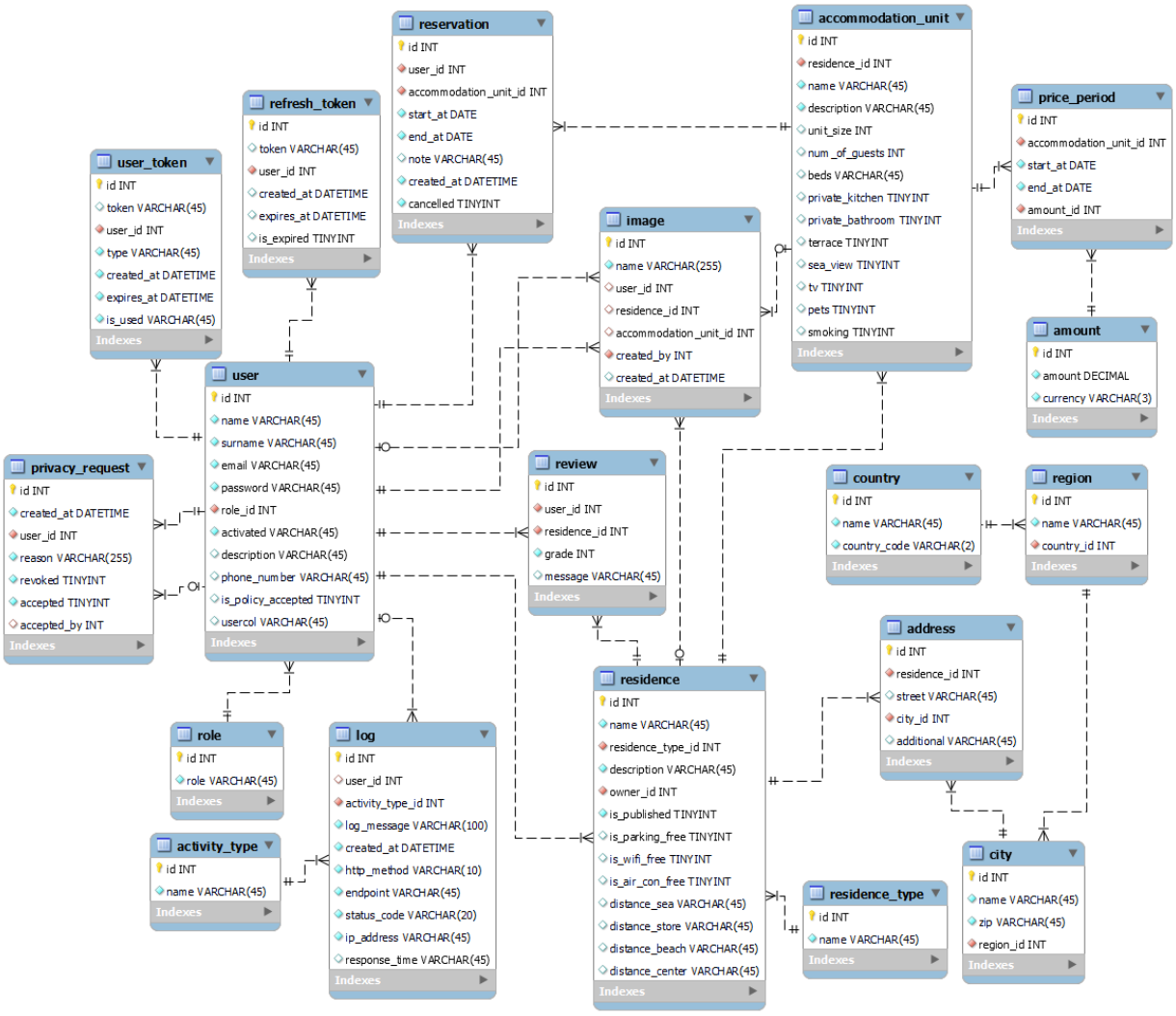


Slika 11: Stranica pretrazivanje web aplikacije

6.2. ERA model

Za pohranu podataka koristila se relacijska baza podataka. Na slici 12. prikazan je ERA model web aplikacije "Odmaralica". Model se sastoji od 17 entiteta. Opis ERA modela započet

će s entitetom "korisnik" (`user`), koji ima obavezne atribute: ime, prezime, e-mail, lozinka te zastavicu koja označava je li korisnik aktiviran. Korisnik mora sadržavati neku ulogu, koje se definiraju u entitetu "uloga" (`role`). Uz entitet "korisnik" vežu se tri entiteta koji su direktno povezani sa sigurnošću aplikacije. Entitet "korisnički token" (`user_token`) koristi se za pohranu tokena vezanih uz registraciju ili obnovu izgubljene lozinke. Entitet "token za osvježavanje" (`refresh_token`) također ima vezu s entitetom "korisnik", te se u njemu pohranjuju tokeni za osvježavanje autorizacijskih tokena. Treći entitet, "dnevnik rada" (`log`), koristi se za pohranu podataka o svakom zahtjevu prema poslužitelju. Dnevnik rada ima vanjski ključ na entitet "vrsta aktivnosti" (`activity_type`). Entitet "korisnik" također je povezan s jednom tablicom koja se odnosi na sigurnost. Tablica "zahtjev za privatnost" (`privacy_request`) sadrži zahtjeve korisnika koji žele da se njihovi privatni podaci izbrišu iz aplikacije..



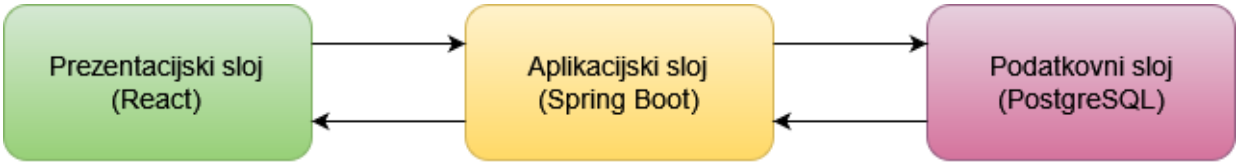
Slika 12: ERA model

Prema slici 12., lijeva strana podatkovnog modela je objašnjena, i sada se fokus prebacuje na poslovnu logiku same aplikacije. Najvažniji entitet, "objekt" (`residence`), sadrži atribute koji opisuju taj objekt, i on mora imati vlasnika, odnosno relaciju s tablicom "korisnik". Osim korisnika, objekt također mora imati vanjski ključ prema entitetu "tip objekta" (`residence_type`). "Adresa" (`address`) je entitet u kojeg se sprema adresa objekta. Zbog toga adresa ima relaciju prema objektu. Osim objekta, adresa također ima vanjske ključeve prema "gradu" (`city`), gradu prema "regiji" (`region`) i regiji prema "državi" (`country`). Na ovaj način, sa podacima zapisanim u tablici adresa, preko grada se može doći do regija i države, te se sprječava zapisivanje nevažećih lokacijskih podataka. Svaki objekt ima svoju prosječnu ocjenu. Relacija "ocjene korisnika" (`review`) služi za pohranjivanje povratnih informacija i ocjena od korisnika koji su boravili u objektu. Stoga relacija "ocjene korisnika" ima obavezne atribute "korisnik" i "objekt". Svaki objekt sadrži svoje smještajne jedinice. Entitet "smještajna jedinica" (`accommodation_unit`) sadrži vanjski ključ prema relaciji "objekt", te sadrži velik broj atributa koji opisuju samu smještajnu jedinicu. Za svaku smještajnu jedinicu potrebno je definirati cjenik u nekom razdoblju. Za tu potrebu služi entitet "razdoblje cijene" (`price_period`), te ona ima relaciju prema "smještajnoj jedinici", te prema tablici "cijena" (`amount`) u koju se spremaju podaci o cijeni. Svaka rezervacija sprema se u tablicu "rezervacija" (`reservation`). Ona sadrži atribute vezane uz period rezervacije, i mora imati vezu prema korisniku koji je rezervirao smještaj, te shodno tome prema smještajnoj jedinici. Zadnja relacija je "slika" (`image`) u koju se spremaju podaci vezani uz sliku kako bi se kasnije lakše dohvatila ista. Svaka slika vezana je uz neki entitet, te stoga slika ima neobavezne relacije prema "objektu", "korisniku" ili "smještajnoj jedinici", te jednu obaveznu vezu prema "korisniku" kako bi se znalo tko je dodao zapis.

6.3. Arhitektura web aplikacije

Odabir arhitekture web aplikacije igra važnu ulogu kada se fokus stavlja na sigurnost. Važno je odabrati arhitekturu koja segmentira i izolira sigurnost na više razina, odnosno komponente cijele arhitekture trebaju biti pojedinačno zaštićene kako probijanje jednog elementa arhitekture ne bi utjecalo na sigurnost drugih. Za razvoj web aplikacije odabrana je arhitektura koja se temelji na MVC (eng. Model-View-Controller) sa RESTful principima za stvaranje web API krajnjih točaka. Komponenta "model", odnosno podatkovni sloj, definira strukturu podataka, entitete i njihove relacije. Za razvoj ove web aplikacije, PostgreSQL baza podataka može se definirati kao model u arhitekturi ili podatkovni sloj. Aplikacijski sloj temelji se na Java programskom okviru Spring Boot, koji služi kao most između podatkovnog i prezentacijskog sloja, odnosno obrađuje HTTP zahtjeve koji dolaze od pogleda te ih prosljeđuje servisima koji komuniciraju sa modelom.

Arhitektura unutar aplikacijskog sloja i njene komponente će u nadolazećem dijelu biti detaljnije prikazane. Na sljedećoj slici je prikazana visoka razina arhitekture web aplikacije:



Slika 13: Arhitektura web aplikacije

6.4. Korištene tehnologije

Za izradu web aplikacije korištene su razne tehnologije, kombinirane kako bi se postigao optimalan rezultat. Za pohranu podataka korištena je Neon Postgres baza podataka u oblaku.

Na strani poslužitelja, odnosno za aplikacijski sloj, korišten je programski okvir Spring Boot u programskom jeziku Java. Spring Boot je vrlo popularan Java programski okvir koji nudi veliki set prednosti i dodataka koji olakšavaju razvoj, povećavaju produktivnost i efikasnost tijekom razvoja. Sa sobom donosi gotove konfiguracije koje značajno smanjuju potrebno vrijeme za konfiguraciju te smanjuju potrebu za pisanjem repetitivnog koda. Također, Spring uključuje ugrađene poslužitelje te se zbog toga može jednostavno koristiti u uređivačima koda kao što je Visual Studio Code.

Ono što Spring okvir čini logičnim izborom za izradu sigurne web aplikacije su ovisnosti, odnosno vanjski moduli koje Spring Boot nudi. Neki od sigurnosnih i ostalih modula koji se koriste u razvoju aplikacije su:

- spring-boot-starter-security – nudi veliki set sigurnosnih značajki kao što su konfiguracija sigurnosnih filtera, CSRF zaštita, definiranje autorizacije i kontrole pristupa te kriptiranje lozinka;
- spring-boot-starter-oauth2-client – konfiguriranje i postavljanje Spring Boot aplikacije kao Oauth2 klijenta za pojednostavljenu integraciju OAuth2 autentifikacije i autorizacije;
- spring-boot-starter-validation – set anotacija za provjeru valjanosti podataka od strane klijenta i za postavljanje ograničenja na atribute modela;
- spring-boot-data-jpa –objektno-relacijsko mapiranje (eng. Object-relational Mapping) za interakciju sa bazom podataka putem objekata i repozitorijskih sučelja

- spring-boot-starter-mail – za konfiguriranje SMTP (eng. Simple Mail Transfer Protocol) poslužitelja i slanje email-ova.

Prezentacijski sloj web aplikacije razvijen je u JavaScript programskom jeziku korištenjem biblioteke React. React omogućava izgradnju odličnog programerskog iskustva te omogućava izradu modularnog korisničkog sučelja koje se sastoji od komponenata. React komponente imaju ekstenziju .jsx (JavaScript XML) te podržavaju dinamičnu promjenu sadržaja, što znači da promjena u jednoj komponenti ne pokreće ponovno prikazivanje cijele stranice. Uz React, korišten je i programski jezik TypeScript koji se temelji na JavaScript-u i pretvara JavaScript u jezik sa strogim tipovima.

Za slanje zahtjeva prema poslužitelju korištena su 2 paketa: Redux Toolkit i Axios. Axios se koristi za slanje i dohvaćanje slika, budući da Redux Toolkit ne podržava slike kao vrstu medija. Korisničko sučelje izgrađeno je korištenjem vlastitih komponenata koje su stilizirane pomoću React Bootstrap paketa i ekstenzije Sass za CSS. Također su korištene i gotove komponente putem paketa MaterialUI i React Bootstrap.

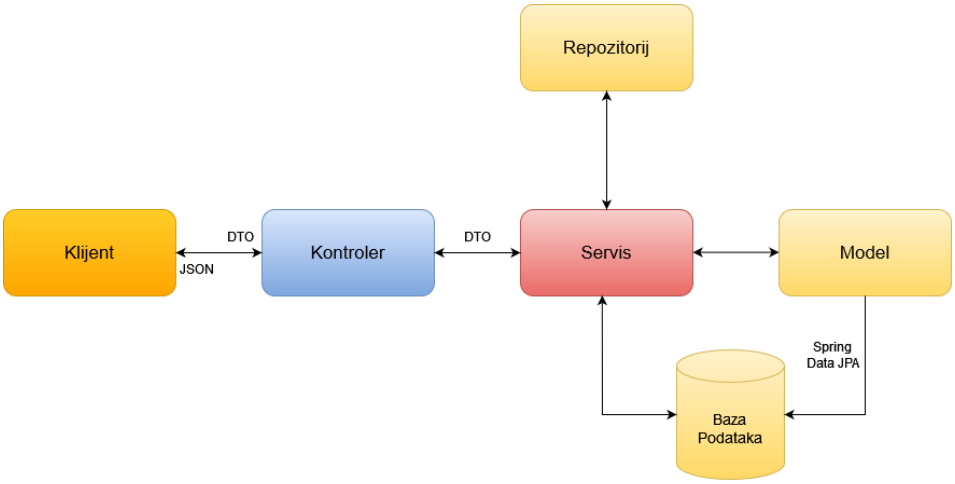
6.5. Razvoj web aplikacije

Razvoj aplikacije prvo je započeo s izgradnjom na strani poslužitelja. Implementirana je bila većina funkcionalnosti, te su sve krajnje točke testirane pomoću Postman alata. Nakon toga počeo je razvoj na strani klijenta, gdje su se gradile komponente i stranice, te su se slali zahtjevi prema krajnjim točkama poslužitelja. U nastavku, ukratko su objašnjeni najvažniji dijelovi razvoja na strani poslužitelja i na strani klijenta.

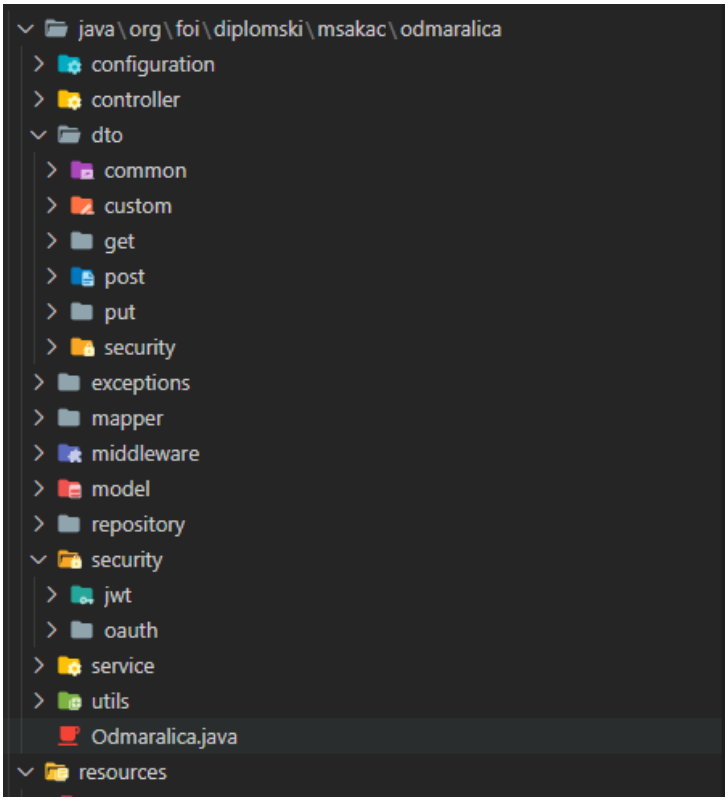
6.5.1. Implementacija na strani poslužitelja

Aplikacija na strani poslužitelja implementirana je korištenjem Spring Boot programskog okvira kao RESTful API aplikacija. Spring Boot donosi arhitekturu koja se bazira na MVC arhitekturi. Na slici 14. prikazana je arhitektura unutar Spring Boot aplikacije, te će se prema komponentama unutar arhitekture objasniti razvoj na strani poslužitelja i prikazati neki dijelovi koda.

Kada izdvojimo klijenta, prikazano je 5 komponenti arhitekture Spring Boot aplikacije. Prema komponentama iz arhitekture može se kreirati dobra struktura projekta, te je takva struktura često konvencija prilikom izrade Spring Boot aplikacije. Takva struktura projekta korištena je prilikom implementacije. Struktura veoma jednostavno i organizirano smješta svaku klasu i sučelje u njezin direktorij ovisno o kakvoj se vrsti komponente radi. Sama finalna struktura vidljiva je na slici 15.



Slika. 14 Arhitektura na strani poslužitelja (Izvor: vlastita izrada, prema [42])



Slika 13. Struktura projekta na strani poslužitelja

Prema strukturi projekta, vidljive su najvažnije komponente arhitekture aplikacije i njezini direktoriji. U nastavku će biti prikazani primjeri implementacije navedenih komponenata:

- Kontroler;
- Servis;
- Model;
- Maper;
- Repozitorij.

Osim prethodno nabrojenih komponenti, u projektnoj strukturi postoji direktorij za konfiguracije gdje se nalaze klase koje konfiguriraju razne aspekte aplikacije, objekti prijenosa podataka (eng. Data Transfer Object, u nastavku DTO), iznimke, međuprogramske klase, sigurnost te ostale pomoćne klase.

6.5.1.1. Model

Direktorij "model" sadrži sve entitete prema ERA modelu. Za svaki entitet koji je prethodno prikazan u ERA modelu, definirana je jedna klasa sa svim atributima i označena je JPA anotacijom `@Entity`, koja označava da se radi o entitetu koji će se koristiti za preslikavanje u bazu podataka. Osim anotacije `@Entity`, dodana je anotacija `@Table` sa nazivom tablice kao parametrom. Za svaki atribut su primijenjena ograničenja prema ERA modelu korištenjem anotacija koje nudi Spring Data JPA. Na ovaj način su kreirani svi modeli, te se prilikom pokretanja aplikacije i uspostavljanja veze s bazom podataka stvaraju entiteti, atributi i relacije između njih. Ovaj pristup smanjuje potrebu za direktnim pristupom bazi podataka i omogućava primjenu pristupa "prvo kod, zatim baza". Prvo se kreiraju svi modeli unutar koda ili projekta, a zatim se prema tim modelima generira baza podataka.

```
@Entity
@Builder
@Table(name = "address")
public class Address implements Serializable {
  private static final long serialVersionUID = 1L;

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Column(name = "id")
  private Long id;

  @ManyToOne
  @JoinColumn(name = "residence_id", referencedColumnName = "id",
          nullable = false)
  private Residence residence;

  @Column(name = "street")
  private String street;

  @ManyToOne
  @JoinColumn(name = "city_id", referencedColumnName = "id", nullable =
false)
  private City city;

  @Column(name = "additional")
  private String additional;
}
```

6.5.1.2. Kontroler

Sva komunikacija između klijenta i poslužitelja odvija se putem HTTP zahtjeva koje kontroler prima. Svaki model ima svoj vlastiti kontroler u kojem je definirana krajnja točka na koju korisnik može slati zahtjeve i vrste HTTP zahtjeva koje kontroler može obraditi. Dužnost

kontrolera je primiti zahtjev od klijenta i proslijediti ga servisu na obradu. Servis može uspješno obraditi zahtjev ili se može pojaviti pogreška, u kojem slučaju servis izaziva iznimku koju kontroler uhvati. Na temelju iznimke, kontroler stvara prikladan odgovor za klijenta. U slučaju uspješne obrade, kontroler vraća odgovor sa statusom 200 i generičkim objektom koji sadrži podatke. S obzirom na to da je programski kod kontrolera za većinu entiteta vrlo sličan, kreirano je generičko sučelje za kontroler i generička apstraktna klasa koja nasljeđuje i implementira metode sučelja. Apstraktni kontroler se potom proširuje i po potrebi izmjenjuje u kontrolerima slične funkcionalnosti. Kontroler podržava zahtjeve za stvaranje (POST), brisanje (DELETE), ažuriranje (PUT), te dohvaćanje svih, pojedinih ili traženje objekata pomoću JPA CriteriaBuilder-a. Ovisno o vrsti zahtjeva, različiti DTO objekti se vraćaju kao odgovor.

```
@PostMapping
public ResponseEntity<Object> create(@Valid @RequestBody PostDTO dto) {
  try {
    GetDTO createdEntity = service.create(dto);
    return ResponseEntity.ok(new CreateResponseDTO<GetDTO>(createdEntity,
HttpStatus.OK));
  } catch (Exception e) {
    return
ResponseEntity.status(HttpStatus.CONFLICT).body(InvalidRequestResponseBuilder
.createResponse(e));
  }
}

@PutMapping()
public ResponseEntity<Object> update(@Valid @RequestBody PutDTO dto) {
  try {
    GetDTO updatedEntity = service.update(dto);
    return ResponseEntity.ok(new CreateResponseDTO<GetDTO>(updatedEntity,
HttpStatus.OK));
  } catch (Exception e) {
    return
ResponseEntity.status(HttpStatus.CONFLICT).body(InvalidRequestResponseBuilder
.createResponse(e));
  }
}

@GetMapping("/find")
public ResponseEntity<Object> queryCountries(@RequestParam("q") String query-
Params) {
  try {
    List<GetDTO> entities = service.find(queryParams);
    return ResponseEntity.ok(new CreateResponseDTO<List<GetDTO>>(entities,
HttpStatus.OK));
  } catch (Exception e) {
    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(InvalidReques-
tResponseBuilder.createResponse(e));
  }
}
```

6.5.1.3. Servis

Servisi unutar aplikacije su odgovorni za obradu poslovne logike. Svaki servis ima definirano svoje sučelje koje mora implementirati. Kontroleri prosljeđuju sve zahtjeve servisu, a servis dalje koristi modele i repozitorije koji su ubrizgani korištenjem ovisnosti (eng. Dependency Injection). Nakon što servis primi zahtjev, prvi korak je pretvoriti DTO objekt u objekt modela. Ovaj proces se naziva mapiranje, a za mapiranje iz jednog tipa objekta u drugi koristi se MapStruct, dodatak za mapiranje objekata. Ovisno o modelu, ako model ima vezu s drugim entitetom, stvara se apstraktna klasa mappera kako bi se ručno pronašao odgovarajući modelni objekt na koji se odnosi veza. Ako nema vanjskih veza, stvara se sučelje mapera. U dijelu koda prikazan je apstraktni servis i neke metode koje se pozivaju unutar kontrolera.

```
public abstract class AbstractBaseService<T, RepositoryType extends
JpaRepository<T, Long>, Mapper, GetDTO, PostDTO, PutDTO>
        implements IBaseService<T, RepositoryType, Mapper, GetDTO, PostDTO,
PutDTO> {

  protected final RepositoryType repository;
  protected final Mapper mapper;
  private final EntityManager entityManager;
  public AbstractBaseService(RepositoryType repository, Mapper mapper,
EntityManager entityManager) {
    this.repository = repository;
    this.mapper = mapper;
    this.entityManager = entityManager;
  }

  @Override
  public GetDTO create(PostDTO entityPost) {
    final T entity = this.convertPost(entityPost);
    T savedEntity = repository.save(entity);
    return this.convertGet(savedEntity);
  }

  @Override
  public List<GetDTO> findAll() {
    List<T> entities = repository.findAll();
    List<GetDTO> getEntities = new ArrayList<>();
    for (T entity : entities) {
      getEntities.add(this.convertGet(entity));
    }
    return getEntities;
  }

  @Override
  public GetDTO update(PutDTO entityPut) {
    final T entity = this.convertPut(entityPut);
    T savedEntity = repository.save(entity);
    return this.convertGet(savedEntity);
  }

  @Override
  public void delete(Long id) {
    repository.deleteById(id);
  }
}
```

6.5.1.4. Maper

Maper se također ubrizgava u servis putem ovisnosti. Ovisno o vrsti zahtjeva, poziva se metoda mapera za preslikavanje iz jednog tipa objekta u drugi. Kao i kod kontrolera, servisi za neke modele mogu biti vrlo slični, pa je važno stvoriti rješenje koje će smanjiti nepotrebnu i ponavljajuću količinu koda. Zbog toga je za servis također stvoreno generičko sučelje i generička apstraktna klasa koja se proširuje u servisima koji imaju sličan način rada. U dijelu koda prikazan je primjer apstraktnog mappera koji se pretvara u konkretnu klasu pomoću MapStruct dodatka.

```
@Mapper(componentModel = "spring", unmappedTargetPolicy =
ReportingPolicy.IGNORE)
public abstract class CityMapper {
  @Autowired
  protected IRegionService regionService;
  protected RegionMapper regionMapper =
Mappers.getMapper(RegionMapper.class);

  @Mapping(source = "cityPostDTO.regionId", target = "region")
  public abstract City convert(CityPostDTO cityPostDTO);

  @Mapping(source = "cityPutDTO.regionId", target = "region")
  public abstract City convert(CityPutDTO cityPutDTO);

  public abstract CityGetDTO convert(City city);

  public abstract City convert(CityGetDTO cityGetDTO);

  protected Region mapToRegion(Long regionId) {
    RegionGetDTO regionGetDTO = regionService.findById(regionId);
    return regionMapper.convertToRegion(regionGetDTO);
  }
}
```

6.5.1.5. Repozitorij

Zadnja komponenta arhitekture je repozitorij. Repozitorij se stvara kao sučelje koje proširuje generičko sučelje JpaRepository. Spring generira upite koje izvršava nad bazom podataka. Repozitorij nudi nekoliko gotovih metoda kao što su findAll, findById, delete, save, create i slične, zbog čega je često potrebno samo stvoriti sučelje repozitorija i ubrizgati ga. Za neke složenije upite, može se ručno stvoriti metoda u sučelju, a Spring će ponovno automatski generirati metodu za izvođenje upita nad bazom.

```
public interface ImageRepository extends JpaRepository<Image, Long> {
  void deleteByAccommodationUnitId(Long id);

  void deleteByResidenceId(Long id);

  void deleteByUserId(Long id);

  List<Image> findAllByAccommodationUnitId(Long id);

  List<Image> findAllByResidenceId(Long id);

  List<Image> findAllByUserId(Long id);
}
```

6.5.2. Implementacija na strani klijenta

Aplikacija na strani klijenta implementirana je pomoću JavaScript React biblioteke. Stranice aplikacije sastavljene su od prilagođenih komponenti, kao i komponenata koje dolaze iz React Bootstrap i Material UI biblioteka. Prilagođene komponente koje su samostalno kreirane stilizirane su korištenjem Sass-a i React Bootstrap klasa.

S obzirom na velik broj entiteta i različitih podataka na strani poslužitelja, u projekt je uključen i Typescript kako bi se olakšalo razvijanje samih komponenti i kako bi pružio dodatnu pomoć pri uređivanju koda. Za svaki entitet na strani poslužitelja, stvorena je odgovarajuća Typescript datoteka u kojoj su definirana sva sučelja potrebna za komunikaciju s kontrolerom tog entiteta. U sljedećem dijelu koda prikazana su sučelja za entitet države.

```
export interface ICountry {
  id: string;
  name: string;
  countryCode: string;
}

export type ICountryPostDTO = Omit<ICountry, 'id'>;

export interface ICustomCityDTO {
  id: string;
  name: string;
  zip: string;
}

export interface ICustomRegionDTO {
  id: string;
  name: string;
  cities: ICustomCityDTO[];
}

export interface ICountryRegionCityResponseDTO {
  id: string;
  name: string;
  regions: ICustomRegionDTO[];
}
```

Pošto se komunikacija između poslužitelja i klijenta odvija putem HTTP zahtjeva, potrebna je biblioteka koja će olakšati kreiranje tih zahtjeva i omogućiti slanje potrebnih podataka bez ponavljanja iste logike više puta. To je postignuto korištenjem Redux Toolkit dodatka koji omogućava jednostavno stvaranje osnovnih krajnjih točaka, konfiguraciju potrebnih parametara koji se šalju pri svakom zahtjevu, te stvaranje svih drugih zahtjeva.

Zahtjevi se lako stvaraju i dodaju u osnovnu krajnju točku. Zatim se svi zahtjevi izvoze kao React kuke i pozivaju se u potrebnim komponentama ili stranicama. Postoje dvije vrste zahtjeva: mutacija, koja označava da se radi o izmjeni podataka (POST, PUT, DELETE), i upit (query), koja označava da se podaci samo dohvaćaju.

U sljedećem bloku koda vidljiva je inicijalizacija osnovnog upita (krajnje točke) i dodavanje krajnjih točaka za adresu.

.

```
const baseQuery = fetchBaseQuery({
  baseUrl: process.env.REACT_APP_BASE_URL,
  prepareHeaders: (headers) => {
    const token = sessionStorage.getItem('accessToken')
        || localStorage.getItem('accessToken');
    if (token) {
      headers.set('authorization', `Bearer ${token}`);
    }
    return headers;
  },
});

const addressApi = api.injectEndpoints({
  endpoints: (builder) => ({
    getAddresses: builder.query<IResponse<IAddress[]>, null>({
      query: () => ({
        url: 'address',
        method: 'GET',
      }),
    }),

    createAddress: builder.mutation<IResponse<IAddress>, IAddressPostDTO>({
      query: (body) => ({
        url: 'address',
        method: 'POST',
        body,
      }),
    }),

    getSingleAddress: builder.query<IResponse<IAddress>, { id: string }>({
      query: ({ id }) => ({
        url: `address/${id}`,
        method: 'GET',
      }),
    }),
  }),
});
```

6.6. Sigurnost web aplikacije

Sigurnost na strani poslužitelja izuzetno je važna jer poslužitelj, odnosno aplikacijski sloj, ima pristup svim podacima u bazi podataka. Spring Boot već u osnovi nudi dobre module i alate za poboljšanje sigurnosti. Budući da je sigurnost glavni fokus u izradi web aplikacije, osim ugrađenih značajki Spring Boota, implementirani su dodatni sigurnosni mehanizmi. U nastavku ovog poglavlja detaljno će se razmotriti i prikazati svi navedeni aspekti sigurnosti web aplikacije, uključujući korištene uzorke, tehnologije i pristupe razvoju koda:

- Alati za kvalitetu i dosljednost programskog koda;
- Autentifikacija;
- Autorizacija (Role Based Access Control)
- DTO;
- SQL ubrizgavanje;
- Dnevnik rada (Secure Logger);
- Validacija korisničkog unosa.

6.6.1. Alati za kvalitetu i dosljednost programskog koda

Na strani klijenta, upotreba alata koji utječu na održivost, čitljivost i produktivnost tijekom razvoja i održavanja igra ključnu ulogu. Iako ti alati ne utječu izravno na sigurnost, doprinose smanjenju šansi za stvaranje grešaka tijekom implementacije te olakšavaju buduće nadogradnje.

Jedan od takvih alata je ESLint, koji se koristi za statičku analizu koda tijekom razvoja aplikacija u JavaScript programskom jeziku. ESLint je visoko konfigurabilan i omogućuje uključivanje i isključivanje različitih postavki koje upozoravaju na probleme tijekom implementacije. Također, pronalazi greške u kodu, cikluse ovisnosti, opasno postavljanje unutarnjih HTML elemenata i, uz određenu konfiguraciju, može spriječiti pokretanje klijenta ako su pronađene takve pogreške.

Pored ESLinta, Typescript igra ključnu ulogu. Budući da se komunikacija između poslužitelja i klijenta odvija isključivo putem HTTP zahtjeva, ključno je znati koje parametre treba poslati u zahtjevu i koje parametre poslužitelj vraća prilikom odgovora na određeni zahtjev. Typescript se koristi za definiranje sučelja koja identično odgovaraju sučeljima i klasama na poslužitelju. Time se znatno smanjuje mogućnost grešaka tijekom izvođenja, olakšava pronalaženje kritičnih točaka za ulazak napadača i poboljšava održivost i kvalitetu koda, što na kraju pridonosi sigurnosti podataka i aplikacije.

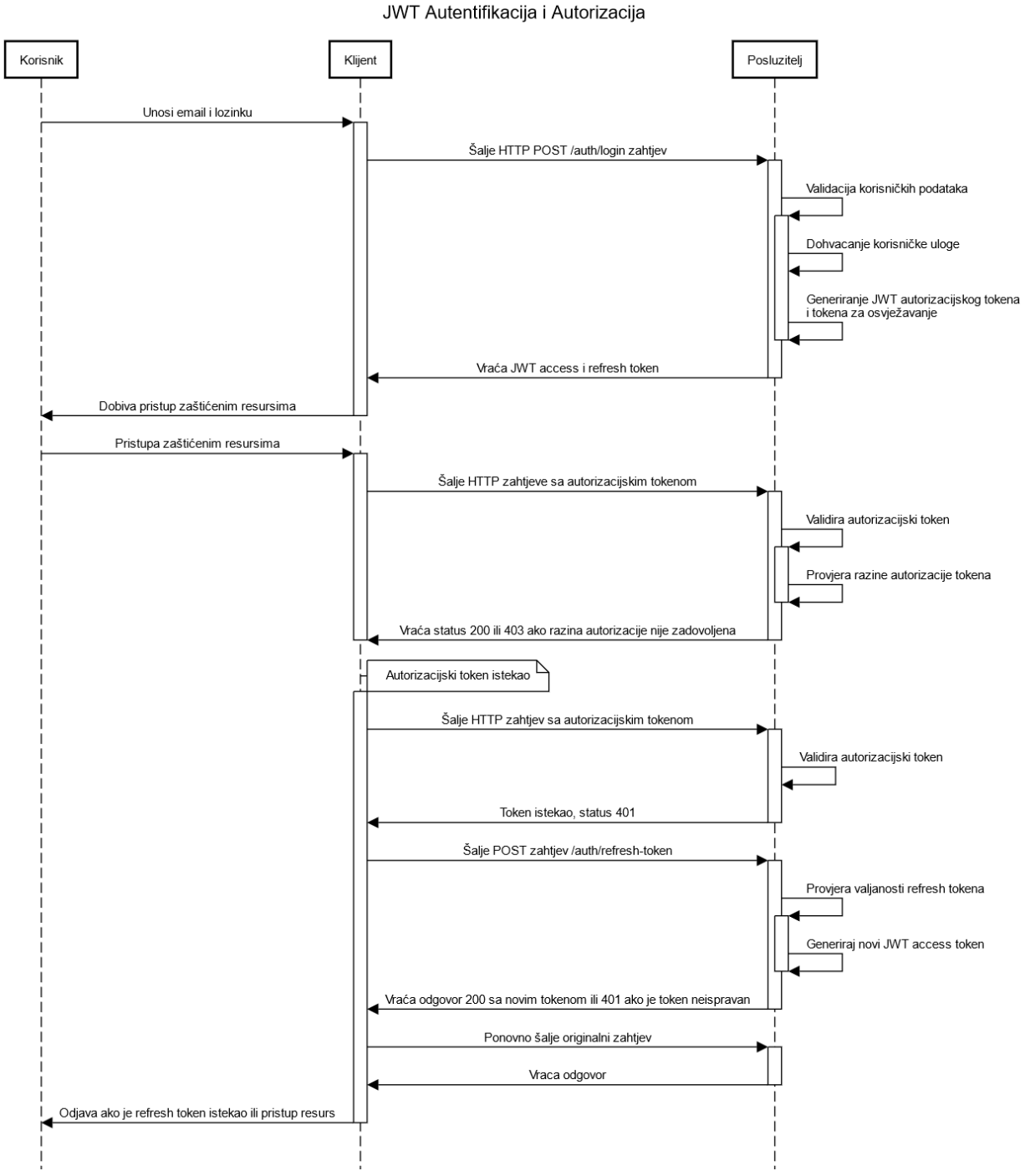
.

6.6.2. Autentifikacija

Autentikacija je jedan od izazovnijih problema prilikom izrade dinamičkih web aplikacija. Autentikaciju je potrebno implementirati na način da se osigura sigurnost i tajnost korisničkih podataka prilikom prijave, a da se pritom pretjerano ne ugrozi korisničko iskustvo prilikom korištenja aplikacije.

Korisnik se u Odmaralicu može prijaviti odnosno autentificirati na dva načina, putem standardne prijave s emailom i lozinkom ili s Google pružateljem za OAuth2 (eng. Open Authorization). U oba slučaja, korisnik prilikom uspješne autentifikacije dobiva natrag dva tokena, autorizacijski token i token za osvježavanje. Tokeni su generirani korištenjem JWT-a (eng. JSON Web Token). Pošto REST poslužitelj mora biti bez stanja, to znači da se u kolačiće ili bazu podataka ne bi smjela zapisivati sesija. Zbog toga se generiraju JWT tokeni koji su bez stanja. Za generiranje JWT tokena potrebno je podesiti nekoliko varijabli. Potrebno je imati davatelja tokena (u ovom slučaju localhost), tajni ključ koji se koristi za kriptiranje i potpisivanje odnosno kriptiranje tokena. U token se sprema nekoliko ključnih podataka koji su od velike važnosti za buduće slanje zahtjeva. U token su spremljeni primarni ključ i uloga korisnika. Dobra praksa je ne zapisivati nikakve privatne podatke u JWT tokene, stoga se koristi privatni ključ kako bi se jedinstveno mogao identificirati osoba odnosno korisnik koji je vlasnik tokena. Autorizacijski i token za osvježavanje generiraju se na isti način, no uz jednu ključnu razliku. Pošto autorizacijski token služi kao ključ kojim korisnik može pristupiti ostalim resursima poslužitelja, sigurnosna je praksa i potreba da JWT autorizacijski token ima veoma kratko trajanje. Autorizacijski token postavljen je na trajanje od 5 minuta. Nakon što prođe 5 minuta od kreiranja tokena, token više nije validan i ne može se koristiti kao jedinstveni identifikator autentificiranog korisnika. Korisnik se zatim ponovno mora prijaviti odnosno autentificirati kako bi dobio novi autorizacijski, odnosno pristupni token. Iako je ovakva metoda autentifikacije veoma sigurna, korisničko iskustvo postaje veoma loše. Zbog toga se u kombinaciju sa pristupnim tokenom šalje i token za osvježavanje koji ima trajanje od 60 sati. Kada korisnik pošalje zahtjev poslužitelju sa isteklim tokenom, poslužitelj odgovara sa HTTP statusom 401, odnosno neovlašten. Na klijentsku aplikaciju dodan je međuprogramski kod (eng. middleware) koji prilikom primanja odgovora 401, šalje POST zahtjev poslužitelju na krajnju točku /auth/refresh-token. Nakon što poslužitelj zaprimi taj zahtjev, provjerava se valjanost tokena za osvježavanje te ukoliko nije istekao, generira se novi autorizacijski token te se zajedno sa tokenom za osvježavanje šalje natrag klijentu. Klijentska aplikacija zatim provjerava odgovor poslužitelja. Ukoliko je odgovor poslužitelja uspješan, novi pristupni token se sprema u spremište te se ponovno radi zahtjev prema originalnom zahtjevu koji je inicijalno vratio odgovor 401. Ukoliko je odgovor poslužitelja neuspješan, odnosno token za osvježavanje je neispravan ili

istekao, korisnika se odjavljuje s aplikacije. Na sljedećem sekvencijskom dijagramu prikazan je cijeli prethodno opisani postupak autentifikacije i neki dijelovi opisane autorizacije. Nakon dijagrama prikazan je blok koda na klijentu koji služi za osvježavanje tokena.



Slika 14: Sekvencijski dijagram sustava JWT autentifikacije

```
const baseQueryWithReauth: BaseQueryFn<string | FetchArgs, unknown,
FetchBaseQueryError> = async (
    args,
    api,
    extraOptions
) => {
  await mutex.waitForUnlock();
  let result = await baseQuery(args, api, extraOptions);
  if (result.error && result.error.status === 401) {
    if (!mutex.isLocked()) {
      const release = await mutex.acquire();
      try {
        const rememberMe = localStorage.getItem('rememberMe');
        const refreshToken = sessionStorage.getItem('refreshToken')
            || localStorage.getItem('refreshToken');
        if (refreshToken) {
          const refreshResult = await baseQuery(
              {
                url: 'auth/refresh-tokens',
                method: 'POST',
                body: {refreshToken},
              },
              api,
              extraOptions
          );
          if (refreshResult.data) {
            const userWithTokens = refreshResult.data as ILoginResponseDTO;
            if (rememberMe === 'true') {
              localStorage.setItem('accessToken',
userWithTokens.accessToken);
              localStorage.setItem('refreshToken',
userWithTokens.refreshToken);
            } else {
              sessionStorage.setItem('accessToken',
userWithTokens.accessToken);
              sessionStorage.setItem('refreshToken',
userWithTokens.refreshToken);
            }
          } else {
            resetAuth();
          }
        }
      } finally {
        release();
      }
    }
    result = await baseQuery(args, api, extraOptions);
  }
  return result;
};
```

6.6.2.1. Registracija

Prilikom registracije, korisnik treba unijeti svoju email adresu i lozinku koja mora biti dulja od minimalno 8 znakova. Ako email adresa nije zauzeta, korisnik je registriran, a lozinka se kriptira pomoću Bcrypt funkcije za hashiranje lozinki, koja je jedna od najboljih metoda zaštite lozinki. Poslužitelj stvara novog korisnika u bazi podataka i generira token za aktivaciju, koji se sprema

u bazu podataka u entitet korisničkih tokena. Nakon registracije, korisniku se šalje email s jedinstvenom poveznicom. Ta poveznica u upitu sadrži generirani token.

Prilikom otvaranja te poveznice, korisnik se vodi na klijentsku aplikaciju, koja odmah šalje zahtjev poslužitelju s tim tokenom. Nakon provjere tokena, poslužitelj aktivira korisnički račun i deaktivira token. Token za aktivaciju ima trajanje od 24 sata. Ako se korisnik pokuša prijaviti s neaktiviranim korisničkim računom, prikazuje mu se poruka o pogrešci da korisnički račun nije aktivan. Kada token za aktivaciju istekne i korisnik se ponovno pokuša prijaviti, stari token se deaktivira, a poslužitelj generira novi token i šalje ga ponovno korisniku na email.

6.6.2.2. Ponovno postavljanje lozinke

Ponovno postavljanje lozinke osjetljiva je akcija koja mora biti dostupna u aplikaciji koja omogućuje korisnicima prijavu. U Odmaralici, ponovno postavljanje lozinke definirano je na sličan način kao i registracija. Nakon što korisnik na klijentskoj aplikaciji otvori stranicu za ponovno postavljanje lozinke i upiše svoju email adresu, zahtjev se šalje poslužitelju na krajnju točku /auth/forgot-password. Poslužitelj provjerava postoji li email adresa u sustavu. Ako email adresa postoji, generira se jedinstveni token za resetiranje lozinke. Taj token ima trajanje od 20 minuta, s obzirom na osjetljivu prirodu ove akcije.

Poslužitelj šalje email poruku koja sadrži poveznicu s tokenom za resetiranje lozinke. Bez obzira na to postoji li email adresa ili ne, klijent uvijek dobiva odgovor s HTTP statusom 200, kako se ne bi davale povratne informacije korisnicima o tome je li token poslan na navedenu adresu ili ne. Nakon što korisnik otvori poveznicu iz poruke, otvara se klijentska stranica gdje korisnik upisuje novu zaporku i šalje zahtjev zajedno s prethodno generiranim tokenom. Ako je token ispravan, korisniku se postavlja nova lozinka.

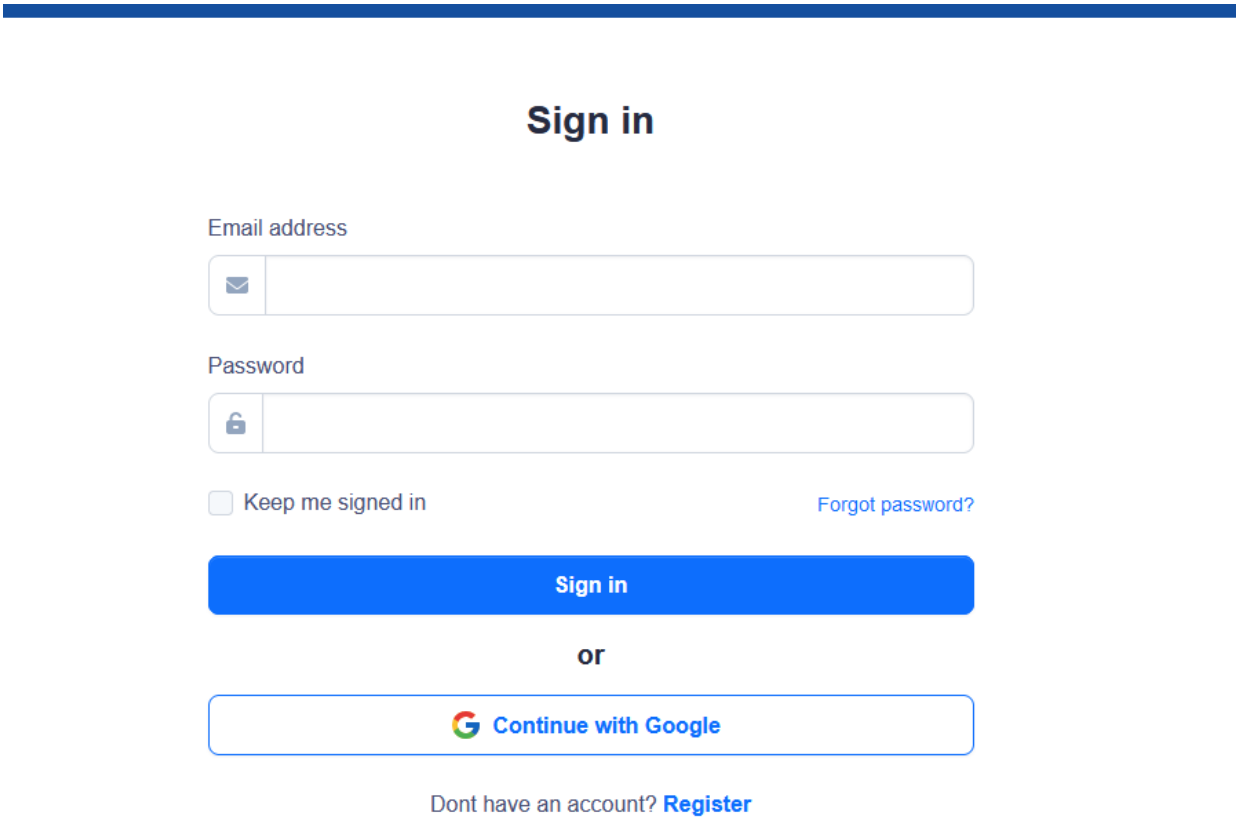
6.6.2.3. Autentifikacija putem OAuth2

Autentifikacija putem OAuth2 je autorizacija putem otvorenog standarda koja omogućuje korištenje druge usluge, odnosno davatelja, bez potrebe za sigurnosnim detaljima. U ovoj web aplikaciji implementirana je prijava putem Google računa. Kako bi se uspješno implementirao ovaj način autorizacije, prvo je bilo potrebno postaviti projekt na Google konzoli. Kreiran je novi projekt te su stvorene vjerodajnice za aplikaciju, kao i ovlašteni URI za preusmjeravanje prilikom prijave. Zatim je na strani poslužitelja kreirana konfiguracija za korištenje OAuth2 klijenta za interakciju sa Google OAuth2 autentifikacijskim sustavom. U sljedećem bloku koda prikazana je OAuth2 konfiguracija u Spring projektu.

```
@Configuration
@ConfigurationProperties(prefix = "oauth2-google")
public class OAuth2Config {

    private String clientId;
    private String clientSecret;
    private String redirectUri;

    @Bean
    public ClientRegistrationRepository clientRegistrationRepository() {
        ClientRegistration clientRegistration =
ClientRegistration.withRegistrationId("google")
                .clientId(clientId)
                .clientSecret(clientSecret)

.clientAuthenticationMethod(ClientAuthenticationMethod.CLIENT_SECRET_BASIC)

.authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)
                .redirectUri(redirectUri)
                .scope("email", "profile")

.authorizationUri("https://accounts.google.com/o/oauth2/v2/auth")
                .tokenUri("https://www.googleapis.com/oauth2/v4/token")
                .userInfoUri("https://www.googleapis.com/oauth2/v3/userinfo")
                .userNameAttributeName("email")
                .clientName("Google")
                .build();

        InMemoryClientRegistrationRepository clientRegistrationRepository =
new InMemoryClientRegistrationRepository(clientRegistration);

        System.out.println(clientRegistrationRepository);

        return new InMemoryClientRegistrationRepository(clientRegistration);
    }
}
```

Prednost autentifikacije korištenjem OAuth2 je taj što se sama logika prebacuje na davatelja usluge. Ukoliko je korisnik neregistriran i želi se registrirati, može se prijaviti sa svojim Google računom. Korisnik pokreće prijavu putem Google na način da klikne poveznicu koja šalje zahtjev poslužitelju. Poslužitelj zatim šalje autorizacijski zahtjev Oauth2 Google davatelju. Korisniku se prikazuje stranica za odabir Google računa. Nakon što korisnik odabere račun OAuth2 preusmjerava korisnika na danu adresu za preusmjeravanje (eng. Redirect URI). U ovom slučaju je to adresa na strani klijenta. No prije nego što je zahtjev uopće poslan na strani poslužitelja kreirana je klasa koja nasljeđuje `DefaultOAuth2UserService` te su nadjačane neke OAuth 2 metode kako bi se dobiveni OAuth2 korisnik mogao procesuirati. Za dobivenu instancu OAuth2 korisnika, provjerava se da li već isti postoji u repozitoriju. Ukoliko isti ne postoji, registrira se novi korisnik sa atributima iz instance OAuth2 korisnika te se za taj račun postavlja zastavica koja indicira da je korisnik aktivirao svoj račun. Ključan korak koji se događa prije samog procesuiranja korisnika je validacija tokena. Spring provjeru tokena sa OAuth2 davateljom odrađuje u pozadini. Kada poslužitelj zaprimi pristupni token, Spring automatski provjerava

potpis tokena, trajanje tokena i druge sigurnosne provjere. Zatim se na stranu klijenta korisnik preusmjerava na danu adresu za preusmjeravanje te se šalje parametar upita prilikom usmjeravanja u kojemu je zapisan autorizacijski JWT token kojega je OAuth2 davatelj generirao. U trenutku kada se na klijentu počinje prikazivati stranica na koju je korisnik preusmjeren, novi zahtjev se šalje poslužitelju koji generira token za osvježavanje i vraća odgovor. U trenutku kada klijent dobije odgovor za tokenom za osvježavanje i pristupnim tokenom, istoga se preusmjerava na početnu stranicu.



Slika 15: Stranica za prijavu

6.6.3. Autorizacija

Prije nego što korisnik pristupi nekom resursu slanjem zahtjeva na poslužitelj, ovisno o resursu, mora imati odgovarajuću razinu autorizacije. U web aplikaciji postoje četiri vrste uloga: korisnik, iznajmljivač, moderator i administrator. Kao što je prethodno opisano, svakom korisniku prilikom prijave, odnosno autentifikacije, u pristupni token upisuje se njegova razina autorizacije, odnosno uloga. Uloga korisnika zatim se koristi na strani klijenta prilikom pokušaja pristupanja nekoj stranici. Za svaku stranicu definirana je uloga koju korisnik mora imati kako bi joj uspješno

pristupio. Ako korisnik nije prijavljen, prilikom pokušaja pristupa stranici koja je namijenjena za višu razinu korisnika, isti je preusmjeren na stranicu prijave. Ukoliko je korisnik prijavljen i pokuša pristupiti stranici namijenjenoj višoj ulozi, bit će preusmjeren na početnu stranicu web aplikacije. Na ovaj način osiguran je ograničen pristup zaštićenim resursima na strani klijenta.

Na strani poslužitelja primjenjuje se kontrola pristupa temeljena na ulogama (engl. Role Based Access Control). Napredniji korisnici i napadači mogu analizirati zahtjeve koji se šalju prema poslužitelju te, prema tome, izraditi vlastite neautorizirane zahtjeve namijenjene krađi podataka, mijenjanju podataka ili nekoj drugoj malicioznoj akciji. Tu u zaštitu stupa komponenta Spring `SecurityFilterChain`, koja je dio lanca filtera koji omogućavaju konfiguriranje niza filtera za obradu HTTP zahtjeva i odgovora u sigurnosnom kontekstu web aplikacije.

Prva komponenta u lancu filtera je JWT autentifikacijski filter koji prvo provjerava da li je korisnik autentificiran i sprema korisničke podatke u kontekst `SecurityContext` koji se zatim dalje koristi prilikom određivanja može li korisnik pristupiti određenom resursu ili krajnjoj točki definiranoj korištenjem sigurnosne metode `antMatchers()`. Ova metoda dopušta definiranje sigurnosnih pravila i kontrole pristupa za specifične krajnje točke, omogućujući određivanje dostupnosti različitim ulogama na temelju navedenih kriterija u pravilu.

Nakon što korisnik napravi zahtjev prema zaštićenom resursu, JWT filter prvo provjerava da li postoji pristupni token i, ako postoji, iz njega izvlači korisničku ulogu koja se sprema u sigurnosni kontekst. Zatim se traži postoji li za dobiveni zahtjev definirano `antMatchers()` pravilo. Ukoliko pravilo postoji, provjerava se kome je dopušten pristup tom resursu. Ukoliko pravilo ne postoji, pokreće se drugi filter koji dopušta pristup resursu samo ako je korisnik prijavljen, odnosno autentificiran.

Za poslužitelja web aplikacije Odmaralica, za svaku krajnju točku definirano je `antMatchers()` pravilo. Za resurse kojima smiju pristupati svi korisnici poziva se metoda permitAll(), koja specificira da navedenom resursu smiju pristupati svi zahtjevi. Za resurse koji imaju ograničeni pristup, definirana su pravila sa metodama `hasAnyAuthority()`, ukoliko resursu može pristupiti više različitih uloga, ili `hasAuthority()`, ukoliko resursu smije pristupiti samo jedan tip uloge.

U sljedećem bloku koda prikazana je konfiguracijska Spring `SecurityFilterChain` Bean. Na početku su prikazana onemogućena CORS pravila i CSRF zaštita zbog lokalnog razvoja, prilagođeni JWT autentifikacijski filter, početak kontrole pristupa bazirane na uzorku sigurnosti Role-Based Access Control, neka od ograničenja pristupa na temelju uloga te na kraju lanca, filteri vezani uz OAuth2 autentifikaciju.

.

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
  http
  .cors().and().csrf().disable()
  .addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class)
  .authorizeRequests()
  .antMatchers(HttpMethod.POST, "/auth/login", "/auth/login-open-auth",
"/auth/register", "/auth/activate").permitAll()
  .antMatchers(HttpMethod.GET, "/role").hasAuthority("admin")
  .antMatchers(HttpMethod.POST, "/role", "/user").hasAuthority("admin")
  .antMatchers(HttpMethod.PUT, "/role", "/user/**").hasAnyAuthority("user",
"moderator", "renter", "admin")
  .antMatchers(HttpMethod.DELETE, "/role", "/user/**").hasAuthority("admin")
  .antMatchers(HttpMethod.GET, "/user").hasAnyAuthority("admin", "moderator")
  .antMatchers(HttpMethod.GET, "/user/**").hasAnyAuthority("user",
"moderator", "renter", "admin")
  .antMatchers(HttpMethod.PUT, "/review").hasAnyAuthority("user",
"moderator", "renter", "admin")
.anyRequest().authenticated().and().exceptionHandling().authenticationEntryPo
int(unauthorizedHandler).and()
.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).a
nd()
  .oauth2Login().authorizationEndpoint().baseUri("/oauth2/authorize")
.authorizationRequestRepository(cookieAuthorizationRequestRepository()).and()
.redirectionEndpoint()
.baseUri("/oauth2/callback/*").and().userInfoEndpoint().userService(customOAu
th2UserService)
.and().successHandler(oAuth2AuthenticationSuccessHandler).failureHandler(oAut
h2AuthenticationFailureHandler);
  return http.build();
}
```

6.6.4. DTO

Objekti za prijenos podataka koriste se za prijenos podataka između različitih dijelova aplikacije, kao što su sloj kontrolera i servisa i najvažnije između klijenta i poslužitelja. Glavni cilj DTO objekata je olakšati prijenos podataka tako da se između različitih komponenti ne prenosi cijeli objekt već samo atribute objekta koji su potrebni za obavljanje određene akcije.

Iako DTO sam po sebi ne igra direktnu ulogu u sigurnosti, itekako pomažu u očuvanju sigurnosti i integriteta aplikacije. DTO-ovi definiraju točnu strukturu koju poslužitelj očekuje prilikom zahtjeva klijenta te definiraju točnu strukturu koju poslužitelj klijentu vraća prilikom odgovora. Specificiranje točnih atributa i njihovih tipova, izvodi se validacija podataka te se osigurava da dolazeći podaci odgovaraju potrebama aplikacije. Korisnik ne može poslati podatke ili tip podatka koji je van strukture DTO objekta. Ukoliko korisnik i pošalje neki atribut koji nije u strukturi DTO-a, prilikom serijalizacije iz JSON objekta u pravi objekt, taj atribut biti će zanemaren. Na ovaj način stvara se zaštita od napada kao što su SQL ubrizgavanje ili XSS napadi.

Osim u domeni sigurnosti DTO utječe i privatnost. Korištenjem istih, klijentima su izloženi samo potrebni podaci, ograničavajući izloženost osjetljivih informacija i podataka. To pomaže u zaštiti aplikacije od otkrivanja privatnih korisničkih podataka koji nisu namijenjeni za klijenta.

Unutar aplikacije, za svaki entitet kreirane su 3 vrste DTO objekta, Put, Get i Post objekt. Svaki od objekta, ovisno o entitetu točno definirana koji podaci moraju biti poslani prilikom slanja Put ili Post zahtjeva te koji podaci će klijentu biti izloženi prilikom Get zahtjeva. Na ovaj način dobivena je potpuna kontrola nad prijenosom podataka te je izloženost ranjivih podataka smanjena na minimum. Primjerice, entitet korisnik prilikom slanja get zahtjeva vraća tek osnovne podatke o korisniku kao što su ime, prezime, email, opis i broj mobitela. Bilo kakvih sigurnosno osjetljivi podaci nisu dostupni u odgovoru poslužitelja. Prilikom primanja zahtjeva, kontroler je označen anotacijom `@Valid` te ako podaci unutar zahtjeva nisu ispravni, kontroler neće prihvatiti zahtjev te će vratiti odgovor 400 odnosno loš zahtjev.

6.6.5. SQL ubrizgavanje

SQL ubrizgavanje vrlo su česti i problematični napadi na koje su poslužiteljske aplikacije posebice ranjive. Prilikom izrade aplikacije, korišten je Spring Data JPA (eng. Jakarta Persistance API). Spring Data JPA omogućava komunikaciju sa bazom podataka koristeći Java objekte i metode te će Spring koristeći iste generirati upite koji će se izvršavati nad bazom podataka. To znači da SQL upiti generiraju potpuni dinamički na temelju pozvanih metoda i parametara koji su njima proslijeđeni. U aplikaciji nije napisan niti jedan ručni SQL upit. Sve akcije sa bazom podataka izvode se striktno koristeći Spring Data JPA. U pozadini, svi upiti koriste pripremljene izjave te parametrizirane upite koji preventiraju ubrizgavanje bilo kakvog malicioznog koda u upit. Pošto nisu pisani upiti koji bi potencijalno mogli sadržavati programersku slučajnu grešku, aplikacije je sigurna od napada SQL ubrizgavanjem.

6.6.6. Dnevnik rada

Sve korisničke akcije na aplikaciji Odmaralica uključuje slanje zahtjeva odnosno primanje ili slanje podataka na poslužitelj. Dnevnik rada igra vitalnu ulogu u sigurnosnom nadzoru i odgovoru na incidente. Događaji povezani sa sigurnošću, kao primjerice neuspjeli pokušaji prijave trebali bi se zabilježiti kako bi bilo moguće otkriti potencijalne prijetnje i pravovremeno odgovoriti na njih. Dnevnik rada omogućuje praćanje radnji korisnik i identifikaciju bilo kakvih sumnjivih ili neovlaštenih aktivnosti.

Web aplikacija Odmaralica bilježi svaki zahtjev korisnika prema poslužitelju koji šalju podatke u JSON obliku. U prethodnom poglavlju, prikazan je bio ERA model u kojem su navedeni svi atributi koji se bilježe u dnevnik rada. Najvažniji od tih atribute su korisnik, vrsta zahtjeva odnosno metoda, krajnja točka, IP adresa te poruka. Ukoliko je zahtjev neuspješan,

najvažniji izvor informacija su krajnja točka, metoda, poruke te IP adresa ili korisnik koje identificiraju osobu koja pokušava neovlaštene radnje.

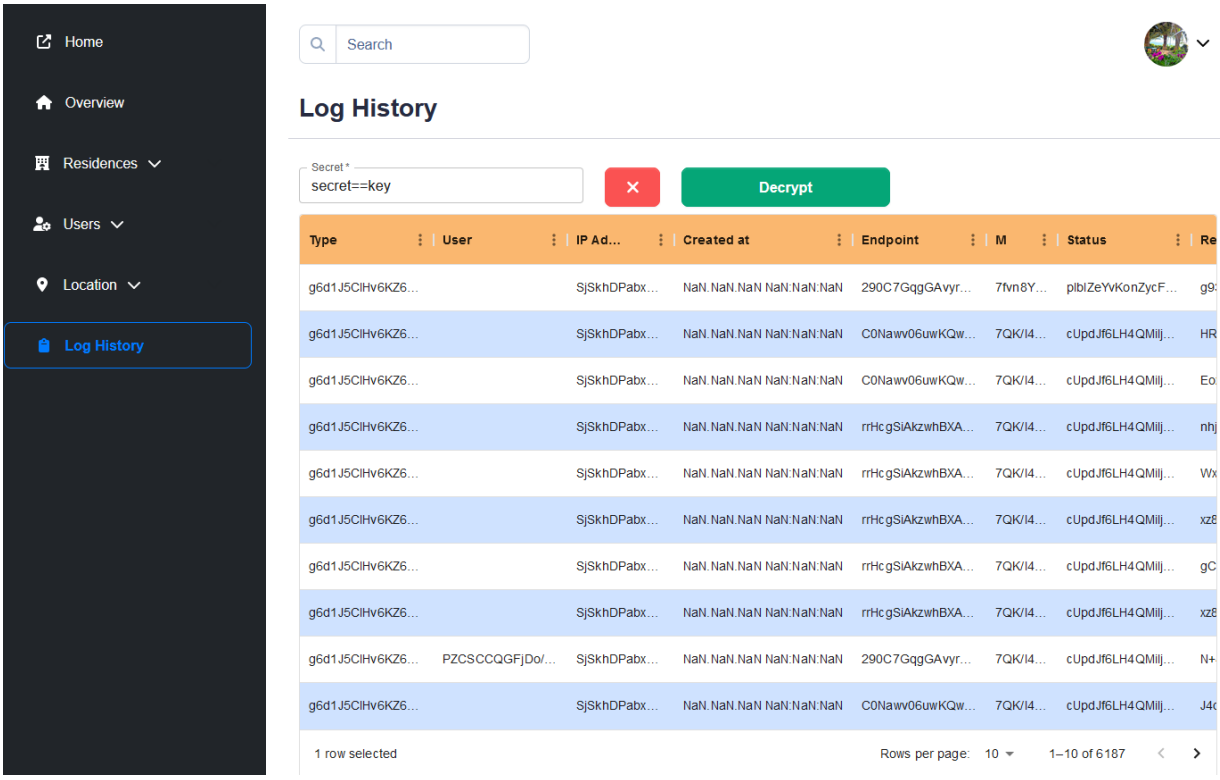
Dnevnik rada implementiran je pomoću aspekatno orijentiranog programiranja. Aspekti predstavljaju snažan alat za modularizaciju i upravljanje specifičnim problemima koji se protežu kroz različite dijelove aplikacije, često obuhvaćajući različite vrste objekata i funkcionalnosti. U kontekstu aplikacije, aspekt je konfiguriran kako bi sustav pratio svaki poziv metode, bilježio parametre koje metoda prima te zabilježio vrijednosti koje metoda vraća. Prije nego što se može definirati aspekt, ključno je definirati točke u izvršavanju aplikacije gdje će se aspekt primijeniti, a te točke se nazivaju "Pointcut". U okviru implementacije dnevnika rada, razvijena je klasa `LoggingAspect` koja definira tri ključne točke ili Pointcuta. Aspekt će biti izvršen prilikom poziva bilo koje metode iz repozitorija, servisa ili REST kontrolera, kao i kontrolera u projektu, pod uvjetom da metoda ima odgovarajuće Spring anotacije poput `GetMapping`, `PutMapping`, `DeleteMapping` ili `PostMapping`. Ovaj pristup omogućuje sustavu temeljitu evidenciju i analizu izvršenih operacija u aplikaciji. Ukoliko se prilikom rada dogodi pogreška u log razvojne okoline ispisati će se lokacija pogreške kao i razlog pogreške [43].

Novi zapis će se u dnevnik rada zapisati tek kada se ubrizgavanjem ovisnosti mogu pronaći HTTP zahtjev i odgovor. To znači da u trenutku kad korisnik napravi zahtjev, aspekt sadrži instancu `HttpServletRequest` objekta no još ne postoji odgovor jer kontroler nije započeo obradu zahtjeva. Nakon što kontroler započinje obradu, pozivaju se razne metode iz repozitorija i servisa koje također aspekt prati u slučaju da se dogodi pogreška. Nakon što ostali slojevi završe svoje aktivnosti i vrate odgovor kontroleru, kontroler priprema odgovor za klijenta. Nakon što kontroler pošalje HTTP odgovor klijentu, aspekt hvata instancu `HttpServlerResponse` objekta te se pošto postoje instance zahtjeva i odgovor, kreira se novi zapis u dnevnik rada sa svim potrebnim informacijama. U sljedećem bloku koda je prikazan jedan dio `LoggingAspect` klase.

Pristup dnevniku rada omogućen je samo administratoru unutar web aplikacije. Pošto postoji rizik od napada na ili presretanja visoko osjetljivih podataka kao što su podaci dnevnik rada, implementirana je dodatna razina sigurnosti. Nakon što administrator zatraži GET zahtjeva prema kontroleru dnevnika rada, svi podaci u dnenvika rada se kriptiraju korištenjem simetričnog kriptografskog algoritma AES sa tajnim ključem (eng. Advanced Encryption Standard). Podaci se u kriptiranom stanju šalju natrag klijentu. Administrator u ovom slučaju dobiva popunjenu tablicu svih podataka dnevnika rada, ali u kriptiranom i nečitljivom stanju. Na klijentskoj stranici omogućen je unos tajnog ključa kojega zna samo administrator web aplikacije. Prilikom unosa ključa, isti se pokušava iskoristiti za dekripciju podataka. Ukoliko dekripcija uspješno prođe prikazani su podaci u čitljivom stanju, a u suprotnom se prikazuje pogreška. Na ovaj način

implementiran je uzorak sigurnosti sličan uzorku Secure Logger. Na sličan način, štiti sustav od iskorištavanja od strane napadača te osigurava cjelovitost sustava zapisivanje te otežava neovlašteni pristup i manipulaciju podacima. Na slici 16., prikazana je stranica za prikaz podataka dnevnika rada i dekripciju.

```
@Aspect
@Component
public class LoggingAspect {

  private final Logger log = LoggerFactory.getLogger(this.getClass());

  @Autowired
  private HttpServletRequest request;
  @Autowired
  private HttpServletResponse response;
  @Autowired
  private LogServiceImpl logService;
  @Autowired
  private ActivityTypeServiceImpl activityTypeService;

  @Pointcut("within(@org.springframework.stereotype.Repository *)" +
          " || within(@org.springframework.stereotype.Service *)" +
          " || within(@org.springframework.web.bind.annotation.RestController*)")
  public void springBeanPointcut() {
  }

  @Pointcut("within(org.foi.diplomski.msakac.odmaralica.controller..*)")
  public void applicationPackagePointcut() {
  }

  @Pointcut("@annotation(org.springframework.web.bind.annotation.GetMapping)" +
          " || @annotation(org.springframework.web.bind.annotation.PutMapping)" +
          " || @annotation(org.springframework.web.bind.annotation.DeleteMapping)" +
          " || @annotation(org.springframework.web.bind.annotation.PostMapping)")
  public void requestMappingMethods() {
  }

  @Around("applicationPackagePointcut() && (springBeanPointcut() ||
requestMappingMethods())")
  public Object logAround(ProceedingJoinPoint joinPoint) throws Throwable {
    long startTime = System.currentTimeMillis();
    String className = joinPoint.getSignature().getDeclaringType().getSimpleName();
    String methodName = joinPoint.getSignature().getName();
    try {
      ResponseEntity<CreateResponseDTO<Object>> result =
(ResponseEntity<CreateResponseDTO<Object>>) joinPoint.proceed();
      logData(startTime, className, methodName, result);
      return result;
    } catch (IllegalArgumentException e) {
      log.error("Illegal argument: {} in {}.{}()",
Arrays.toString(joinPoint.getArgs()),
              className, methodName);
      throw e;
    }
  }
```



Slika 16: Dnevnik rada

6.6.7. Validacija korisničkog unosa

Validacija korisničkog unosa ključna je komponenta za osiguranje ispravnosti i sigurnosti web aplikacije. U React .jsx komponenatama, iskorišteni su automatski ugrađeni mehanizmi za validaciju kako bi se osigurao siguran korisnički unos. React omogućava ugradnju različitih tehnika validacije, uključujući provjeru formata unosa, duljinu lozinki, obavezna polja te provjeru prije slanja podataka na poslužitelja. Također, React osigurava da se korisnički unos automatski „čisti“ od potencijalno zlonamjernih skripti ili neispravnih znakove, čime se povećava sigurnost aplikacije. Jedan od načina da se zaobiđe automatsko čišćenje unosa je korištenje `dangerouslySetInnerHTML` no u tom slučaju ESLinter baca pogrešku i ne dopušta pokretanje klijenta.

6.7. Privatnost web aplikacije

Prilikom izrade web aplikacije, vođena je pozornost i na privatnost korisničkih podataka diljem funkcionalnih dijelova aplikacije. Korisnici web aplikacije pružaju minimalnu količinu podataka prilikom registracije koje je potrebna za potpuno korištenje funkcionalnost. Korisnici moraju unijeti svoje ime, prezime te email adresu. Ti podaci javno su vidljivi jedino ako korisnik

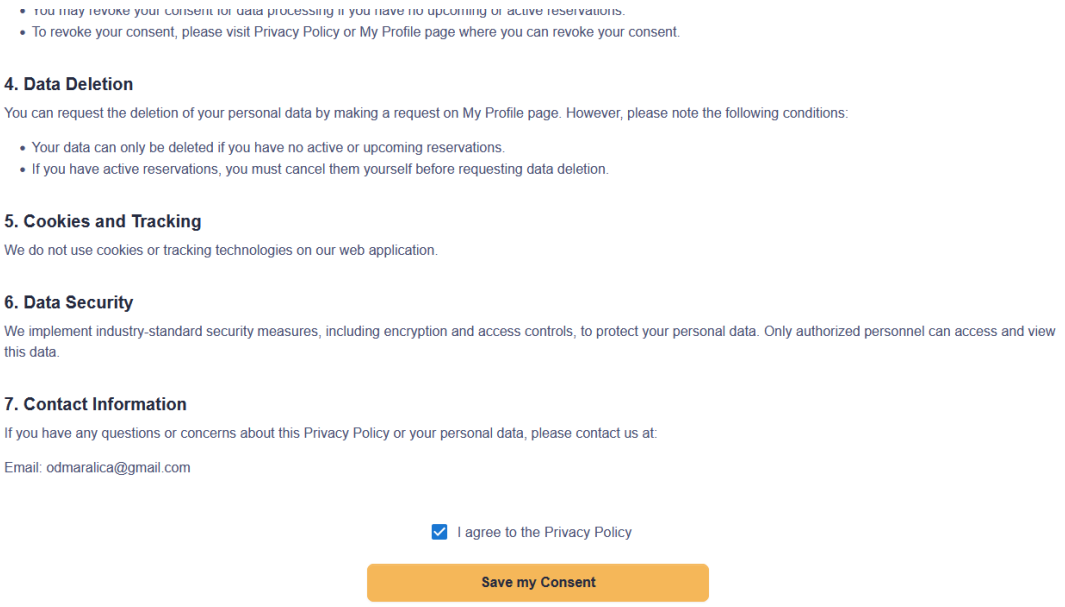
ocjenjuje objekte u kojima je stanovao. Podaci su vidljivi moderatorima, administratoru te iznajmljivaču kod kojega je korisnik iznajmio smještajnu jedinicu.

U web aplikaciju implementiran je uzorak politike privatnosti. Osim uzorka korisnik može zatražiti dozvolu za brisanjem korisničkih podataka.

6.7.1. Politika privatnosti

Svakom korisniku web aplikacije omogućen je pristup stranici politike privatnosti. Na toj stranici opisani su način sakupljanja, obrade i zaštite osobnih podataka unutar web aplikacije. Kao što je prethodno rečeno, ukoliko se korisnik želi registrirati potrebno je nekoliko osobnih podataka. Svaki zahtjev registriranog korisnika bilježi se u dnevnik rada zbog sigurnosnih razloga. Podaci su kriptirani te je pristup tim podacima strogo nadzoren. Korisnički podaci prikupljaju se zbog pružanja usluge rezerviranja smještajnih jedinica, poboljšanja platforme i servisa te očuvanju sigurnosti web aplikacije.

Nakon što se korisnik registrira, on nema mogućnost rezerviranja smještajnih jedinica tako dugo dok ne prihvati politiku privatnosti. Nakon što korisnik pokuša rezervirati smještaj, preusmjerava ga se na stranicu politike privatnosti. Politiku privatnosti potrebno je pročitati i prihvatiti kako bi korisnici imali znanja o načina rukovanja njihovim osobnim podacima unutar web aplikacije. Osim toga politika privatnosti ključna je za izgradnju povjerenja između korisnika i web aplikacije te kako bi se osigurala transparentnost o rukovanju podacima. Ukoliko je korisnik prihvatio politiku privatnosti, istu privolu može u bilo kojem trenutku povući, ali tek ukoliko korisnik nema nadolazećih ili aktivnih rezervacija. Ukoliko postoji nadolazeća rezervacija korisnik samostalno mora odgoditi isti te tada može povući privolu o politiki privatnosti. Ukoliko korisnik ima aktivnu rezervaciju, dužan je zadržati privolu do datuma kraja rezervacije. Nakon što korisnik povuće svoju privolu o politiki privatnosti, u bilo kojem trenutku može ponovno prihvatiti istu.



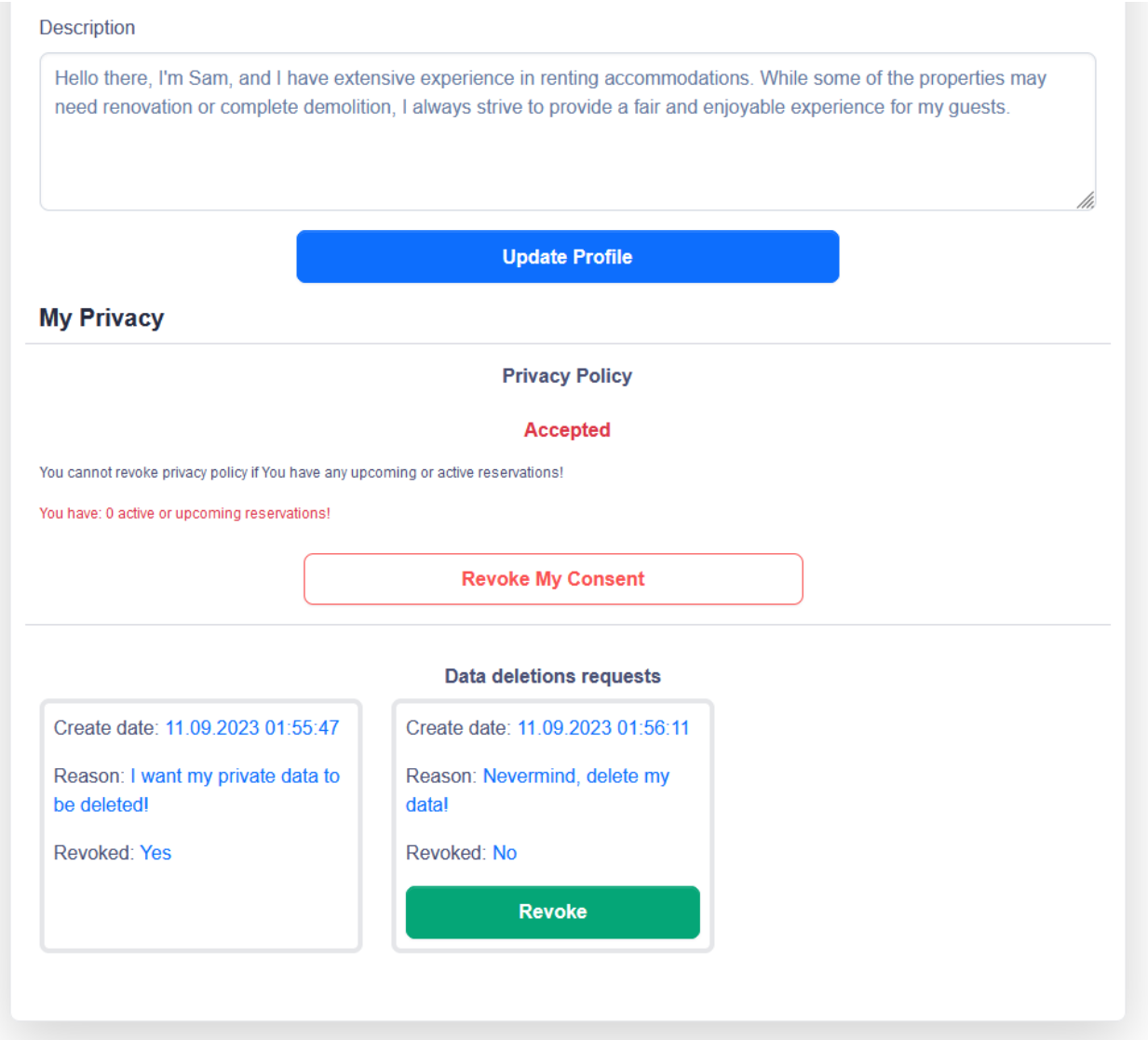
Slika 17: Prihvaćanje politike privatnosti

6.7.2. Brisanje korisničkih podataka

Prema GDPR-u, korisnik ima „pravo na zaboravljanje“. To znači da korisnik u bilo kojem trenutku može zatražiti brisanje svih osobnih podataka. U web aplikaciji korisnik može posjetiti stranicu svojega profila te na dnu stranice postoji sekcija gdje korisnik može upisati razlog zbog kojeg traži svoje pravo na zaboravljanje. Nakon što korisnik upiše svoj razlog, pojavljuje se niz obavijesti za korisnika. Korisnik nema pravo na zaboravljanje ukoliko ima nadolazeće ili aktivne rezervacije. Razlog tome je što brisanje podataka prije ili prilikom boravka u rezerviranom smještaju može imati utjecaj na iznajmljivača i na samu funkcionalnost web aplikacije. S toga je korisnik prije slanja zahtjeva za brisanje podataka dužan otkazati rezervacije ili sačekati završetak aktivne rezervacije.

Nakon što korisnik pošalje zahtjev za brisanje podataka, moderator i admin dobivaju novi zahtjev koji moraju provjeriti i potvrditi. U međuvremenu, korisnik uvijek može povući svoj zahtjev za brisanjem podataka te u tom slučaju se zahtjev zatvara i na strani moderatora i administratora. Ukoliko korisnik ipak ne povuče zahtjev, te je njegov zahtjev odobren, brišu se korisnički podaci ime, prezime, email adresa, broj mobitela i opis te se korisnički račun deaktivira. Uz to se također brišu i sve recenzije sa ocjenjenih objekata. Iako se svi podaci brišu, korisnički račun i dalje ostaje u bazi podataka sa nasumično generiranim podacima. Razlog ne brisanje korisničkog računa je taj što bi svi zapisi diljem aplikacije koji su vezani uz tog korisnika, morali biti izbrisani. Odnosno ukoliko je korisnik imao deset završenih rezervacije, sve rezervacija bi se

obrisalo što bi utjecalo na poslovne podatke iznajmljivača. Na ovaj način iznajmljivač i dalje vidi sve prethodne rezervacije, ali više nema uvid u osobne podatke izbrisanog korisnika.



Slika 18: Zahtjev za brisanje korisničkih podataka

6.8. Izazovi i problemi u razvoju aplikacije

Prilikom razvoja aplikacije pojavili su se razni problemi i izazovi. U kontekstu sigurnosti, razvijanje aplikacije može lako započeti na pogrešan način. Kasnije, kada je potrebno doraditi određeni dio, javljaju se razni problemi, što može rezultirati gubicima velike količine vremena zbog nedostatka unaprijed razmišljanja.

Osim toga, naglašava se da sigurnost zahtijeva visoku razinu stručnosti, jer osim razumijevanja implementacije određene funkcionalnosti, također je bitno shvatiti kako se ta

funkcionalnost ili način implementacije može zloupotrijebiti kako bi se ostvarila neovlaštena kontrola nad podacima ili izvršile neovlaštene radnje nad podacima.

Razvoj aplikacije krenuo je prvo sa razvojem poslužiteljske aplikacije. Kreirani su sve komponente koje se standardno koriste u Springu. Nakon što je skoro cijela poslovna logika aplikacije bila implementirana, potrebno je bilo dodati autentifikaciju i autorizaciju. Ovaj pristup je bio prvi izazov i problem u razvoju jer se sigurnost nije uzimala prva u obzir. Sve krajnje točke bile su testirane pomoću Postman aplikacije no bez ikakvih autorizacijskih prava. Nakon što je implementirana autentifikacija pomoću i autorizacija pomoću uloga i JWT tokena, ponovno je bilo potrebno testirati sve krajnje točkih kojih je na koncu bilo više od osamdeset. No, prije samog testiranja prvo je potrebno bilo definirati tko sve može pristupati kojoj krajnoj točki i definirati autorizacijska prava za sva krajnje točke. Pošto nekim krajnjim točkama može pristupiti više vrsta korisničkih uloga, za te krajnje točke moralo se provesti još više testiranja. Rješenje ovog problema bilo je jednostavno, prvo implementirati sustave autentifikacije i autorizacije te zatim kreirati kontrolere i poslovnu logiku. U ovom slučaju radi se o manjem broju kontrolera, modela i krajnjih točaka. No, u slučaju kompleksnijeg i većeg Spring REST servisa, pogrešan pristup mogao bi prouzročiti ogromne sigurnosne rizike.

Pošto je dnevnik rada jedan od najvažnijih elemenata sigurnosti te reagiranja na sigurnosne incidente, potrebna je solidna implementacija, kontinuirani i kvalitetni nadzor cijelog promenta koji dolazi i odlazi sa poslužitelja. Prvi pristup implementaciji dnevnika rada, bio je na razini kontrolera. Ideja je bila kreirati komponentu koja će prije odgovora kontrolera klijentu, očitati podatke iz zahtijeva i odgovora te zatim kreirati redak u dnevnika rada sa tim podacima. Inicijalno, ovaj pristup izgledao je jednostavno za implementirati. Sama implementacija takvog dnevnika rada je veoma jednostavno, ali nikako skalabilna. Za svaku metodu kontrolera odnosno krajnju točku, potrebno bi bilo pozivati komponentu koju će zahtijev zapisivati u dnevnik rada. Iako u programskog kodu postoji apstraktni kontroler kojega koristi većina kontrolera za ostale kontrolere koji ne koristi apstraktni kontroler trebala bi se pozivati i postavljati komponenta zadužena za zapisivanje zahtjeva u dnevnik rada. Nakon uspješne implementacije, zbog konstatnog ponavljanja koda, potražen je bolji način za spremanje zahtjeva u dnevnik rada. Tada su pronađeni aspekti, koji su u prethodnim poglavljima detaljnije razrađeni. Implementacija dnevnika rada pomoću aspekata bila je potrebna jednom te je samo rješenje skalabilno, odnosno dodavanjem novih kontrolera nisu potrebne nikakve izmjene već se u dnenvik rada automatski zapisuju i zahtjevi prema njima.

Prilikom razvoja aplikacije na strani klijenta, za određene stranice potrebno je bilo raditi i do desetak različitih zahtjeva. Zbog loše testiranog sustava autorizacije, konstatno su bile potrebne izmjene na lancu sigurnosnih filtera na poslužitelju, jer su se unutar jednog autorizacijskog

pravila definirala pristupna prava jedne ili više uloga na više krajnjih točaka i sa više različitih HTTP metoda što je na koncu uzrokovalo međusobnim gaženjem autorizacijskih pravila. Zbog toga je ta implementacija potpuno maknuta te su implementirana autorizacijska pravila samo za jednu HTTP metodu te za jednu ili maksimalno tri krajnje točke ukoliko se radi o GET zahtjevima pošto se može dohvaćati lista resursa, jedan resurs ili traženje resursa.

Problem se također javio prilikom implementacije autorizacije putem Google OAuth2 davatelja. Prilikom uspješne autentifikacije i autorizacije, davatelj usluge je zahtjev slao na klijentsku aplikaciju umjesto na poslužiteljsku. Zbog neiskustva, riješenje je bilo traženo u kodu cijele aplikacije no nikad nije bilo pronađeno jer se je problem krio u pogrešnoj konfiguraciji Google klijenta. Naime, povratna krajnja točka (eng. callback) na koju Google nakon autorizacije šalje zahtjev bila je definirana sa pogrešnim portom što je uzrokovalo slanjem zahtijeva klijentu umjesto poslužitelju. Također je tijek rada i općenito sam proces iza OAuth2 autentifikacije i autorizacije poprilično kompliciran i težak za shvatiti i implementirati.

7. Zaključak

Sigurnost je jedan od ključnih faktora i prioriteta web aplikacija koji igra važnu ulogu tijekom njihovog cijelog životnog ciklusa. Ovaj rad detaljno istražuje ključne aspekte sigurnosti i privatnosti u kontekstu web aplikacija, te pruža najbolje prakse zaštite korisničkih podataka. Analizirane su metode, tehnike i alati koji unapređuju zaštitu od sigurnosnih prijetnji, s posebnim naglaskom na osiguravanje integriteta, dostupnosti i povjerljivosti u svim web okruženjima.

Objašnjeno je kako loša arhitektura može ozbiljno ugroziti sigurnost. Također, istraženi su česti napadi na web aplikacije koji mogu nanijeti veliku štetu ne samo krajnjim korisnicima, već i organizacijama koje stoje iza tih aplikacija. Definirana je privatnost osobnih podataka u skladu s regulatornim tijelima koja nadziru zaštitu tih podataka. Organizacije koje krše ova pravila podložne su ozbiljnim kaznama. Stoga je od suštinskog značaja razumjeti utjecaj sigurnosti i privatnosti na dizajn i arhitekturu web aplikacija te ih integrirati u razvojni proces od samog početka.

Postoji mnogo uzoraka koji mogu poboljšati sigurnost i privatnost web aplikacija. Uzorci sigurnosti i privatnosti nisu standardizirani, i ne primjenjuju ih svi arhitekti i razvojni timovi. Ipak, služe kako bi podržali i poboljšali sigurnost i privatnost web aplikacija. Stoga je preporučljivo razumjeti ove uzorke i pažljivo ih planirati prije početka razvoja.

Tijekom razvoja web aplikacije, korištene su moderne tehnologije koje automatski pridonose povećanju sigurnosti cijelog sustava. Java programski okvir Spring Boot nosi mnoge prednosti koje se jednostavno mogu implementirati i proširiti. Sustav autorizacije unutar Spring okvira je korisnički pristupačan i omogućava visoku razinu kontrole pristupa. No, kako bi se uspješno i efikasno implementirao, ovakav sustav treba primijeniti rano tijekom razvoja. Nedostatak iskustva kod razvojnog tima ili programera može dovesti do pojave sigurnosnih propusta koji mogu ugroziti cijeli sigurnosni sustav.

U zaključku, naglašena je ključna uloga sigurnosti i privatnosti u očuvanju korisnika i integriteta podataka u današnjem sve turbulentnijem digitalnom okruženju. Sigurnost i privatnost zahtijevaju kontinuiranu pažnju tijekom cijelog životnog ciklusa web aplikacije kako bi se osigurala zaštita korisnika i njihovih podataka.

Popis literature

[1] D. J. Brooks, “What is security: Definition through knowledge categorization” *Security Journal*, vol. 23, no. 3, pp. 225–239, 2010. DOI: 10.1057/sj.2008.18.

[2] Cisco, “What Is IT Security? https://www.cisco.com/c/en/us/products/security/what-is-it-security.html (pristupljeno 08.06.2023).

[3] Check Point, “What is Cyber Security?", 2022. https://www.checkpoint.com/cyber-hub/cyber-security/what-is-cybersecurity/ (pristupljeno 08.06.2023).

[4] M. Bay, “WHAT IS CYBERSECURITY? In search of an encompassing definition for the post-Snowden era”, French Journal for Media Research, 2016. ISSN: 2264-4733

[5] J. Stangarone, “5 big problems caused by bad application architecture,” *mrc*, 2012. https://www.mrc-productivity.com/blog/2012/02/5-big-problems-caused-by-bad-application-architecture/ (pristupljeno 22.06.2023).

[6] Microsoft, “Common web application architectures,”, 2023. https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures (pristupljeno 22.06.2023).

[7] M. Harwood, R. Price *Internet and Web Application Security,* 3. Izd., Burlington: Jones & Bartlett Learning, 2022

[8] WSAC, “Web Application Security Consortium.” http://www.webappsec.org/ (pristupljeno 27.06.2023).

[9] OWASP, “About the OWASP Foundation | OWASP Foundation.” https://owasp.org/about/ (pristupljeno 27.06.2023).

[10] ZAP, “Getting Started.” https://www.zaproxy.org/getting-started/ (pristupljeno 27.06.2023).

[11] Dependency Trak “Introduction” https://dependencytrack.org/ (pristupljeno 27.06.2023).

[12] SAMM “What is OWASP SAMM?” https://owaspsamm.org/about// (pristupljeno 27.06.2023).

[13] OWASP, “OWASP Top 10:2021.” https://owasp.org/Top10/(pristupljeno 27.06.2023).

[14] A. Hoffman, "Web Application Security", 1.Izd., USA: O'Reilly, 2020.

[15] J.C.S.Santos, K. Tarrit, i M. Mirakhorli, "A Catalog of Security Architecture Weaknesses", IEEE International Conference on Software Architecture Workshops (ICSAW), 2017

DOI: 10.1109/icsaw.2017.25

[16] M. Shema, "Hacking Web Apps - Detecting and Preventing Web Application Security Problems", USA: Syngress, 2012.

[17] X. Wang, Y. Zheng, R. Zhang, P. Zhang, "Attack and defenses in user authentication systems: A survey", Journal of Network and Computer Applications, ScienceDirect, 2021

DOI: 10.1016/j.jnca.2021.103080

[18] J. Blatz, „CSRF: Attack and Defense“, McAfee, 2011

[19] A. Skendzic, B. Kovacic, E. Tijan, „General data protection regulation – Protection of personal data in an organisation“, IEEE, 2018.
DOI: 10.23919/MIPRO.2018.8400247

[20] Agencija za zaštitu osobnih podataka, “Najcesce postavljena pitanja”

https://azop.hr/najcesce-postavljena-pitanja/ (pristupljeno 09.07.2023).

[21] GDPR Informer, “Vodič kroz GPDR za početnike”, 2018

https://gdprinformer.com/hr/vodic-kroz-gdpr (pristupljeno 09.07.2023).

[22] GDPR.EU, "What is GDPR", 2023

https://gdpr.eu/what-is-gdpr/ (pristupljeno 10.07.2023).

[23] Linkedin, "How do you incorporate security and privacy principles into your application architecture", 2023.

https://www.linkedin.com/advice/0/how-do-you-incorporate-security-privacy
(pristupljeno 13.07.2023).

[24] ClickIT, "Web Application Architecture: The Latest Guide 2022", 2022.

https://www.clickittech.com/devops/web-application-architecture/#h-an-overview-of-web-application-architecture

(pristupljeno 13.07.2023)

[25] Hackr.io "What is Web Application Architecture? Components, Models and Types", 2022

https://hackr.io/blog/web-application-architecture-definition-models-types-and-more

(pristupljeno 13.07.2023).

[26] A. Masood "Cyber Security for Service Oriented Architectures in a Web 2.0 World: An Overview of SOA Vulenrabilities in Financial Services", IEEE, International Conference on Technologies for Homeland Security, 2013

DOI: 10.1109/ths.2013.6698966

[27] Invicti, "Monolithic vs microservices architecture: Which is better for security?", 2023

https://www.invicti.com/blog/web-security/monolithic-vs-microservices-architecture-which-is-better-for-security/

(pristupljeno 14.07.2023).

[28] O. Al-Debagy, P. Martinek, "Comparative Review of Microservices and Monolithic Architectures", IEEE, 18th International Symposium on Computational Intelligence and Informatics, 2018

DOI: 10.1109/cinti.2018.8928192

[29] Linkedin, "What are some common security risks and challenges in a microservices security architecture pattern", 2023.

https://www.linkedin.com/advice/0/what-some-common-security-risks-challenges (pristupljeno 19.07.2023).

[30] A. P. Rajan, "Serverless Architecture - A Revolution in Cloud Computing", Tenth International Conference on Advanced Computing, 2018

DOI: 10.1109/ICoAC44903.2018.8939081

[31] N. Mateus-Coelho, M. Cruz-Cunha, "Serverless Service Architectures and Security Minimals", 10th International Symposium on Digital Forensics and Security, 2022

DOI: 10.1109/ISDFS55398.2022.9800779

[32] AWS, "What is SOA(Service-Oriented Architecture)?"

https://aws.amazon.com/what-is/service-oriented-architecture/ (pristupljeno 28.07.2023)

[33] A. Masood, "Cyber Security for Service Oriented Architectures in a Web 2.0 World: An Overview of SOA Vulenrabilities in Financial Services", IEEE International Conference on Technologies for Homeland Security, 2013

DOI: doi:10.1109/ths.2013.6698966

[34] C. Dougherty, K. Sayre, R. C. Seacord, D. Svoboda, K. Togashi, "Secure Design Patterns", Software Engineering Institute, 2009

DOI: 10.1184/R1/6583640.v1

[35] J. Siljee, "Privacy Transparency Patterns", Proceedings of the 20th European Conference on Pattern Languages of Programs, 2015

DOI: 10.1145/2855321.2855374

[36] P. C. Clements, R. Kazman, M. H. Klein, "Evaluating Software Architectures: Methods and Case Studies", Addison-Wesley Professional, 2001.

[37] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns - Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1995.

[38] E. Fernandez-Buglioni, "Security Patterns In Practice - Designing Software Architectures Using Software Patterns", Wiley Series, 2013.

[39] R. Wasserman, B .H. C. Cheng, "Security Patterns", Department of Computer Science and Engineering - Michigan State University, 2003.

[40] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, P. Sommerlad, "Security Patterns - Integrating Security and Systems Engineering", John Wiley & Sons, 2006.

[41] UC Berkley School of Information, "Privacy Patterns"

https://privacypatterns.org/ (pristupljeno 03.08.2023).

[42] Javatpoint, „Spring Boot Architecture“

https://www.javatpoint.com/spring-boot-architecture (pristupljeno 04.09.2023).

[43] Spring, „Aspect Oriented Programming with Spring“

https://docs.spring.io/spring-framework/docs/4.3.15.RELEASE/spring-framework-reference/html/aop.html (pristupljeno 09.09.2023).

Popis slika

Slika 1: Odnos između računalne, informacijske i IT sigurnosti (Izvor: vlastita izrada, prema: [4]) 9
Slika 2: Tijek rada perzistentog XSS napada (Izvor: vlastita izrada, prema: [14]) 18
Slika 3: Tijek rada reflektiranog XSS napada (Izvor: vlastita izrada, prema: [14]) 19
Slika 4: Primjer CSRF napad (Izvor: vlastita izrada, prema: [18]) 23
Slika 5: Sekvencijski dijagram uzorka SAP (Izvor: vlastita izrada, prema: [40]) 40
Slika 6: Prikaz sustava sa korištenim SAP uzorkom (Izvor: vlastita izrada, prema: [40]) 40
Slika 7: Struktura uzorka Secure Logger (Izvor: vlastita izrada, prema: [34]) 41
Slika 8: Dijagram klasa uzorka Secure Adapter (Izvor: vlastita izrada, prema: [38]) 43
Slika 9: Struktura uzorka Input Validation (Izvor: vlastita izrada, prema: [34]) 44
Slika 10: Dijagram klasa uzorka Session-Based Role-Based Access Control 45
Slika 11: Stranica pretrazivanje web aplikacije 52
Slika 12: ERA model 53
Slika 13. Struktura projekta na strani poslužitelja 57
Slika 14: Sekvencijski dijagram sustava JWT autentifikacije 66
Slika 15: Stranica za prijavu 70
Slika 16: Dnevnik rada 76
Slika 17: Prihvaćanje politike privatnosti 78
Slika 18: Zahtjev za brisanje korisničkih podataka 79

Popis tablica

Tablica 1: OWASP 10 web aplikacijskih sigurnosnih rizika [13] 14
Tablica 2: Sigurnosne slabosti web aplikacije prema sigurnosnim aspektima [15] 15

Prilozi

Uz rad se prilažu sljedeći prilozi:

Aplikacija na strani klijenta: https://github.com/msakac/odmaralica_frontend

Aplikacija na strani poslužitelja: https://github.com/msakac/odmaralica