Razvoj web aplikacija otvorenog koda u programskom jeziku Java

Bencek, Valentino

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:853892

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

VALENTINO BENCEK

Razvoj web aplikacija otvorenog koda u programskom jeziku Java

DIPLOMSKI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Valentino Bencek

Matični broj: 0016123610

Studij: *Baze podataka i baze znanja*

Razvoj web aplikacija otvorenog koda u programskom jeziku Java

DIPLOMSKI RAD

Mentor:

dr. sc. Matija Novak

Varaždin, rujan 2021.

Valentino Bencek

Izjava o izvornosti

Izjavljujem da je moj završni/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/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Tema rada su web aplikacije otvorenog koda. Rad daje uvod u web aplikacije koji obrađuje web servise, način rada web aplikacije, arhitekturu web aplikacije, sigurnost web aplikacije i tehnologije za razvoj web aplikacije. Uspoređuje web aplikacije s drugim vrstama aplikacija poput mobilnih i stolnih aplikacija. Glavni fokus rada je na razvoj web aplikacija. Dane su metodologije za razvoj kao i principi za razvoj web aplikacija. Opisana je razlika u razvoju kod aplikacija otvorenog koda i aplikacija zatvorenog koda. Opisani su izazovi s kojima se programer susreće prilikom razvoja aplikacije. Na kraju teorijskog poglavlja, pobrojene su licence te su tablično prikazana njihova svojstva. To poglavlje je izrazito važno kada govorimo o aplikacijama otvorenog koda. Praktični dio sastoji se od analize web aplikacija koja obuhvaća analizu dizajna i pozadine aplikacije. Najveći dio praktičnog djela sastoji se od izrade web aplikacije otvorenog koda u programom jeziku Java.

Ključne riječi: web aplikacija;Java;otvoreni kod;razvoj softvera;web programiranje;web tehnologije;softver;

Sadržaj

1. Uvod

Web aplikacija je softverski program koji je pohranjen na poslužitelju (engl. Serveru) i dostupan korisnicima preko web preglednika. Razlikuju se od tradicionalnih aplikacija po tome da ih ne pokreće operacijski sustav već se aplikacijama pristupa putem web preglednika i izvodi se na udaljenom poslužitelju. Glavna prednost web aplikacije nad stolnim (engl. desktop) aplikacijama je to što programeri ne moraju aplikaciju prilagođavati različitim operacijskim sustavima. Aplikacija koja se pokreće preko Chrome web preglednika radit će na oba operacijska sustava: Windows i Linux. Kada je potrebno ažurirati aplikaciju, programeri ne trebaju forsirati korisnike na ažuriranja već je samo potrebno ažurirati aplikaciju na poslužitelju te će svi korisnici koristiti jednaku verziju aplikacije. S korisničke strane, web aplikacije pružaju konzistentan izgled na svim platformama koji ovisi jedino o korištenom pregledniku. Također, omogućuju da se podaci spremaju na udaljeni poslužitelj, a ne lokalno, kako bi se njima moglo pristupiti s bilo koje lokacije ili uređaja eliminirajući pritom potrebu za ručnim prebacivanjem podataka. Naravno, postoje i neki nedostaci web aplikacija ako ih usporedimo sa stolnim aplikacijama. Web aplikacije većinom troše resurse (CPU, RAM) poslužitelja za obavljanje operacija te imaju limitirani pristup resursima korisničkog računala. Programi koji zahtijevaju veliku količinu resursa (Obrada videa, obrada slike, ..) bolje se realiziraju kao stolne aplikacije zbog lakšeg pristupa resursima osobnog računala. Izbor između web i stolnih aplikacija ovisi uvelike o korisniku. Većina kompanija nudi obje mogućnosti korisniku na odabir. Na primjer Microsoft ima stolne verzije svojih programa (Office, Excel,..) kao i verzije za preglednike.

Web aplikacija je optimizirana web stranica koja se sastoji od četiri glavne karakteristike preko kojih se lagao prepoznaje [1]:

1. Potreban je samo **jedan razvojni proces** kako bi radila na svim uređajima. Drugim riječima, web aplikacija se prilagođava svim operacijskim sustavima.
2. **Aplikaciju nije potrebno preuzeti**. Aplikacija se izvodi na poslužitelju i pristupa joj se preko web preglednika. Jedna „negativna“ strana toga je što zahtjeva pristup internetu.
3. **Aplikaciji se pristupa preko bilo kojeg Web preglednika**. Moguće je koristiti bilo koji web preglednik: Chrome, Safari, Opera, Brave i slično
4. **Pronalazi ih se preko web tražilica.** Kao i bilo koju web stranicu, web aplikaciju moguće je locirati preko tražilice upisom ključnih riječi.

Postoje dva osnovna **tipovi web aplikacija**. Aplikacije su klasificirane ovisno o svojoj funkcionalnosti i načinu prezentacije informacija[1]:

- **Statične web aplikacije** – osnovna vrsta web aplikacije koja je ujedno i najjednostavnija za implementirati. Takve aplikacije najčešće su kreiranje korištenjem HTML-a i CSS-a bez prave „backend“ potpore. Mogu sadržavati animacije, videozapise i slike. Ukoliko bi se neki sadržaj na takvoj web aplikaciji trebao promijeniti, potrebno je preuzeti cijeli HTML kod, izmijeniti ga te ponovo učitati na poslužitelj. Česti primjeri takvih aplikacija su razni portfelji ili digitalni životopisi neke osobe. Također, često prikazuju kontakt informacije koristeći statički sadržaj.
- **Dinamičke web aplikacije** – vrsta aplikacije koja je znatno kompleksnija od statičke web aplikacije. Dinamičke web aplikacije u svojoj pozadini imaju bazu podataka koja služi za pohranu svih podataka i informacije koje se prikazuju ili dohvaćaju preko preglednika. Omogućuju interaktivnu promjenu sadržaja preko korisničkog sučelja. To je čest rezultat sustava za upravljanje sadržajem (engl. Content Management System) ili skraćeno CMS. Takav sustav dolazi s dinamičnom web aplikacijom i omogućuje administratoru sustava kao i korisnicima da lagano ažuriraju i uređuju svoj sadržaj. Postoje mnogi jezici koji se mogu koristiti prilikom izrade web aplikacije. Za izradu modernijih i kompleksnijih web aplikacija može se koristi jezik Java. Ažuriranje sadržaja je lagano kod ovakve vrste aplikacija no zahtjeva dobro programiranu pozadinu koja često može biti kompleksna ako se radi o većim aplikacijama. Dobar primjer dinamičkih aplikacija su E-commerce aplikacije. Web aplikacija koja se ponaša kao trgovina i omogućuje korisnicima kupnju preko interneta. Razvoj ovakve vrste aplikacije zahtjeva dobro definiranu strukturu načina plaćanja. U ovo kategoriju spadaju još i portali – aplikacije s mnogo različitih sekcija te animirane web aplikacije**.**

2. Web aplikacije

Poglavlje daje uvod u web aplikacije te govori o pojmovima koji se koriste kroz cijeli rad. Započinje s web servisima kao jednom od glavnih komponenti web aplikacija.

Dalje je opisan način rada web aplikacije. Radi se o komunikaciji između klijenta i poslužitelja koja funkcionira tako da klijent prilikom neke radnje šalje zahtjev poslužitelju, poslužitelj prihvaća i obrađuje zahtjev te daje odgovor klijentu. Odgovor je dan u obliku HTML stranice koji preglednik razumije i zna kako prikazati klijentu. Jedan dio poglavlja posvećen je i arhitekturi web aplikacija. Arhitektura je vrlo bitan pojam kod aplikacija jer razlikuje loše aplikacije od onih dobrih u smislu razine skalabilnosti, modularnosti i sigurnosti. Nabrojene su vrste arhitektura web aplikacija: tri glavne podijele – aplikacija jedne stranice, mikroservisi i arhitektura bez poslužitelja i neke poznate arhitekture poput: klijent – poslužitelj i Model-View-Controller arhitektura koja je popularna prilikom izgradnje Java web aplikacija.

Velik dio ovog poglavlja posvećen je sigurnosti web aplikacija. Sigurnost je bez sumnje jedna od najbitnijih pojmova prilikom izrade bilo koje vrste aplikacija, a pogotovo prilikom izrade web aplikacija. Opisana je ranjivost web aplikacija te su pobrojeni i opisani napadi koje je moguće izvesti nad web aplikacijama. Radi se od dosta poznatim napadima poput: SQL injektiranja, Krađe sesije, Napada uskraćivanja usluge i slično. U skladu s time dane su i preporuke za izgradnju sigurne web aplikacije. Radi se o postupcima koje je potrebno primijeniti prilikom izgradnje same aplikacije. Spomenut je i OWASP kao jedna od najpoznatijih organizacija koja pruža smjernice za sigurnost ali i besplatne programe otvorenog koda koji služe za očuvanje sigurnosti aplikacija. Poseban dio poglavlja posvećen je sigurnosti aplikacija otvorenog koda kao i sigurnosti Java web aplikacija.

Opisane su i tehnologije koje se koriste za razvoj web aplikacija. Opisan je klijent i poslužitelj te programski jezici za razvoj klijenta kao i programski jezici za razvoj poslužitelja. Govori se i o web preglednicima, bitnim protokolima za komunikaciju između klijenta i poslužitelja, formatima zapisa podataka, API-ima te okvirima za razvoj aplikacija.

Zadnji dio ovog poglavlja odnosi se na usporedbu web aplikacija s drugim tipovima aplikacija. Točnije, uspoređuje se sa stolnim aplikacijama, mobilnim aplikacijama te web stranicama. Dolazi se do zaključka da svaka aplikacija ima svojih prednosti i nedostataka te svaka vrsta aplikacije ima svoju namjenu.

2.1. Web servisi

Web servisi su aplikacije ili izvori podataka kojima se pristupa putem standardnih web protokola (HTTP/HTTPS). Za razliku od web aplikacija, web servisi su dizajnirani da komuniciraju s drugim programima (aplikacijama) ,a ne direktno s korisnicima. Podaci koje web servis daje najčešće su u XML i JSON obliku. Aplikacija koja prima te podatke može ih lako pročitati i pretvoriti u oblik koji joj odgovara. Najpoznatiji protokol web servisa (SOAP) jednostavno dodaje zaglavlje svakoj XML poruci prije nego se pošalje preko HTTP-a. Web servisi koji su orijentirani na poslovanja koriste standard poznat pod nazivom UDDI (Universal Description, Discovery and Integration) standard. Takav standard formatira podatke u specifičnu vrstu XML-a poznatom pod kraticom WSDL (Web Service Description Language). Glavna svrha web servisa je pružiti API (Application Programming Interface), sučelje koje sadrži set funkcija preko kojih se pristupa podacima. Jedan od primjer Web servisa koji pruža API je „openweathermap“, aplikacija preko koje se mogu dobiti informacije o vremenu za određene koordinate.[2]

Svaki web servis trebao bi imati sljedeće karakteristike [2]:

- Dostupan je preko interneta ili preko privatne mreže
- Koristi standardizirani XML način razmjene poruka
- Nije vezan za operacijski sustav ili programski jezik
- Može se lagano razumjeti koristeći XML gramatiku
- Može se lagano otkriti koristeći mehanizme pretrage

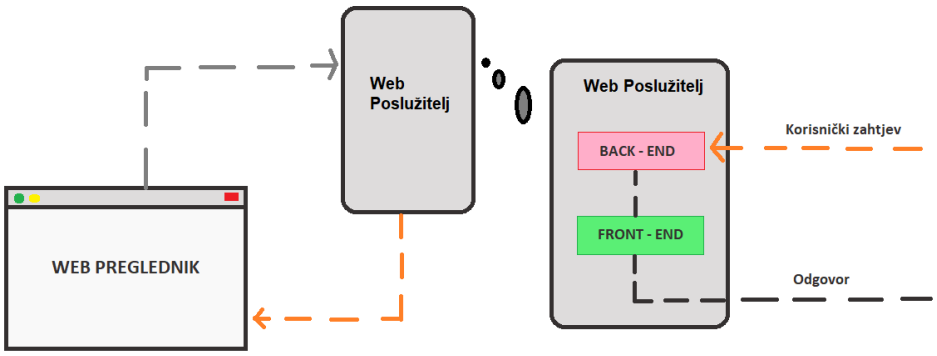
Web servis sastoji se od sljedećih komponenti [2]:

- XML strukturiranih poruka
- SOAP (Simple Object Access Protocol) – služi za prijenos poruka
- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Service Description Language) – opisnik za dostupnost usluge

2.2. Način rada web aplikacija

Sve web aplikacije nalaze se na nekom web poslužitelju, odnosno udaljenom računalu na kojem je instaliran potreban softver koji im omogućuje da poslužuju web aplikacije. Svako računalo može se pretvoriti u web poslužitelj pod uvjetom da ima potreban softver i dovoljno resursa za neometano posluživanje klijenata aplikacije.

Kada korisnik u web preglednik upiše URL (Uniform Resource Locator), zahtjev za prikaz sučelja web aplikacije poslan je na web poslužitelj. Web poslužitelj nakon primljenog zahtjeva prikupi sve potrebne informacije za web stranicu i dostavlja ih putem HTTP protokola. Zahtjeva korisnika zapravo se prosljeđuje u pozadinu (engl. Backend) aplikacije na obradu. Rezultat obrade šalje se korisniku u obliku poruku ili konkretnih podataka koje je korisnik zahtijevao. Donja slika prikazuje tu komunikaciju.

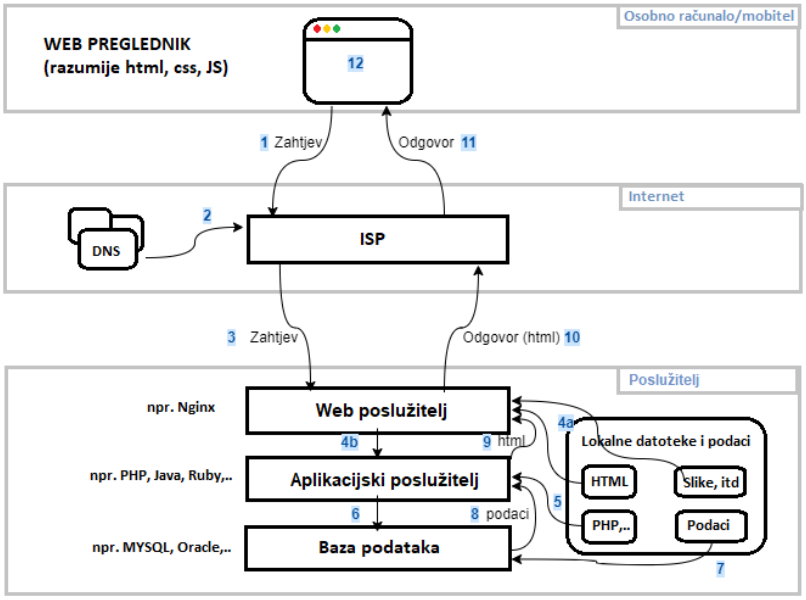


Slika 1: Komunikacija preglednika i poslužitelja

Gledajući donju sliku, redoslijed obrade zahtjeva je sljedeći [3]:

1. Klijent se sa svojim računalom spoji na mrežu i preko preglednika šalje URL zahtjev.
2. Računalo korisnika spaja se na javnu mreži preko ISPa i traži IP adresu za stranicu kojoj pokušava pristupiti.
3. Zahtjev se prenosi preko interneta do IP adrese koja odgovara URL-u.
4. Poslužitelj koji se nalazi na primateljskoj strani zahtjeva ima instaliran potreban softver i ako je u mogućnosti dohvaća datoteku koja je bila tražena zahtjevom i vraća ju nazad. Ako se traži HTML datoteka, poslužitelj ju nalazi i vraća ju natrag, a ako se radi o naprimjer .php datoteci poslužitelj zna kako baratati s njom te ju šalje aplikacijskom poslužitelju na obradu.
5. Aplikacijski poslužitelj dobiva HTTP zahtjev od web poslužitelja, nalazi datoteku s izvornim kodom za tu web adresu (naprimjer .php datoteka) i izvrši ju.
6. Program sadrži referencu na bazu podataka i šalje upit kako bi dobio potrebne podatke iz baze.
7. Poslužitelj baze podataka dohvaća potrebne datoteke iz upita.
8. Poslužitelj baze podataka šalje tražene podatke aplikacijskom poslužitelju

9. Aplikacijski server napravi završnu obradu podataka te ih šalje natrag web serveru kao HTTP odgovor.
10. Web poslužitelj šalje HTML tekst do tražene konekcije.
11. Podaci su poslani preko interneta na korisničko računala i konekcija se prekida.
12. Preglednik sada može pročitati dobiveni HTML kod i zna kako ga prikazati na ekran korisnika.



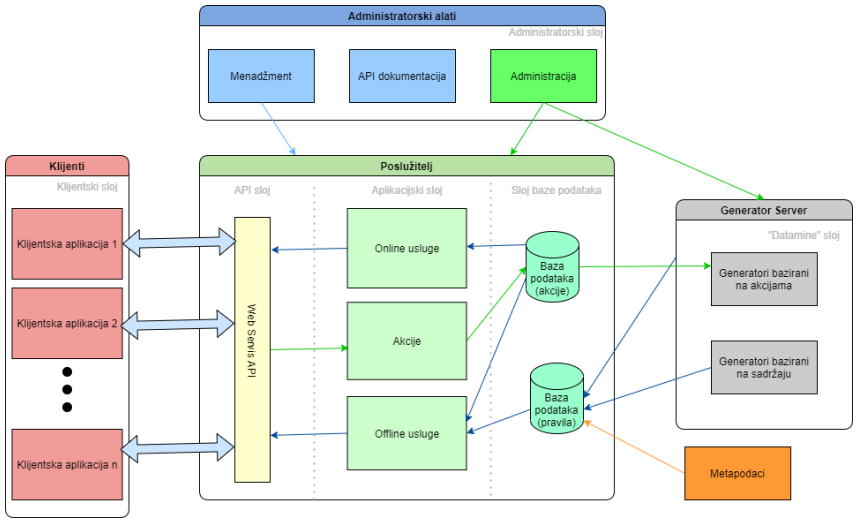
Slika 2: Detaljna obrada zahtjeva (Prema: [3])

2.3. Arhitektura web aplikacija

Arhitektura web aplikacija definira interakcije između aplikacija, sistema i baza podataka kako bi osigurala da više aplikacija mogu raditi zajedno. Aplikacija je dizajnirana da funkcionira efikasno te istovremeno ispunjava svoje specifične potrebe i ciljeve. Današnje web aplikacije zahtijevaju dobro definiranu arhitekturu zato što je sva komunikacija između programa, aplikacija ili uređaja bazirana na webu. Arhitektura osigurava da aplikacija bude skalabilna, efikasna, robusna te sigurna. Arhitektura neke web aplikacije zapravo je okvir (engl. Framework) koji se sastoji od svojih vanjskih i unutarnjih komponenata te veza i interakcija između komponenata aplikacija. Te komponente mogu biti korisnička sučelja, baze podataka ili neki sistemi. Primarni cilj arhitekture je osigurati da sve komponente aplikacije pravilno surađuju kako bi dale podlogu za izgradnju buduće aplikacije.

Donja slika prikazuje osnovnu arhitekturu neke web aplikacije. Prvi sloj odnosi se na klijente (engl. Clients). Aplikacija treba biti u mogućnosti prihvatiti veći broj klijenata. Klijenti

aplikaciju pokreću preko preglednika pristupajući joj preko URL-a. Sljedeća komponenta je poslužitelj koja sadrži više slojeva. Sloj koji pruža web servis (API), aplikacijskog sloja čija je glavna svrha vršiti obrade podataka i dohvaćanje podataka iz baze podataka. Podatke prosljeđuje i prima od sloja web servisa te od korisnika. Sloj baze podataka sastoji se od jedne ili više baze koja pohranjuje podatke u tablice. Na zahtjev aplikacijskog sloja, baza podataka izvršava upite i vraća daje podatke na obradu. Postoji i administratorski sloj koji vrši nadzor nad poslužitelj te sadrži potrebnu dokumentaciju aplikacije.



Slika 3: Osnova arhitektura web aplikacije (Prema: [4])

Postoje tri tipa arhitekture web aplikacija koji se koriste tijekom razvoja modernih web aplikacija [4]:

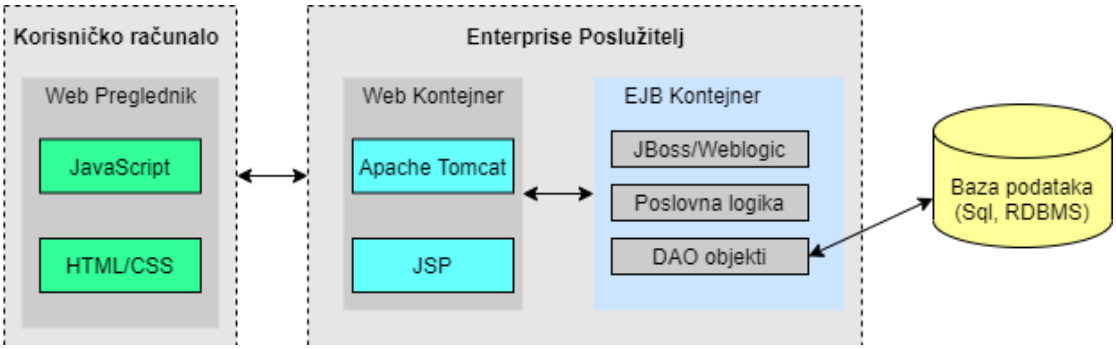
- **Aplikacija jedne stranice (engl. Single Page Application (SPA))** – Moderne i efikasne aplikacije dizajnirane su tako da zahtijevaju samo potrebne elemente sadržaja i informacija kako bi generirale intuitivno i interaktivno korisničko iskustvo (engl. User Experience). Ovakva vrsta aplikacija dinamičnija je u odnosu na ostale na način da korisniku prikazuje ažurirane informacije unutar trenutno učitane stranice bez potrebe da se učita potpuno nova stranica slanjem zahtijeva poslužitelju. Na taj način poboljšava se iskustvo korisnika jer web aplikacija počinje djelovati kao stolna aplikacija. Ovakav način dizajniranja web aplikacije oslanja se na AJAX (Asynchronous JavaScript and XML) pozive i nemoguće ga je izvesti bez AJAXa.
- **Mikroservis (engl. Microservice)** – Mikroservisi su mali i lagani servisi čija svrha je izvršiti jednu, specifičnu funkcionalnost. Ova arhitektura omogućuje programerima da povećaju svoju produktivnost i ubrzaju proces razvoja aplikacija. Pošto komponente u aplikaciji ne ovise jedna o drugoj direktno, ne

moraju biti razvijene u istom jeziku. Omogućuje programerima da rade s različitim tehnologijama koje i najbolje odgovaraju za realizaciju nekog problema.

- **Arhitektura bez poslužitelja (engl. Serverless Architecture)** – Aplikacija nema vlastit poslužitelj na kojem se izvodi, već poslužitelj pruža neka treća strana. Ovu arhitekturu koriste tvrtke koje nemaju vlastite poslužitelje ili ne žele održavati poslužitelje. Samim time ne moraju se brinuti o konfiguriranju poslužitelje te jačini poslužitelje već se mogu fokusirati na razvoj same aplikacije.

Java ima vlastitu web arhitekturu poznatu kao **Java web aplikacijska arhitektura.** Popularna je u poslovnom (engl. Enterprise) razvoju aplikacija zbog svoje prilagodljivosti. Može se koristiti za razvoj manjih, jednostavnih aplikacija ili većih, više-slojnih aplikacija. U oba slučaja pružit će aplikacije koje će biti na visokoj razini performansa, skalabilnosti te sigurnosti. Prednost ove arhitekture je mogućnost korištenja besplatnih Java alata i okvira za izradu različitih rješenja. [5]

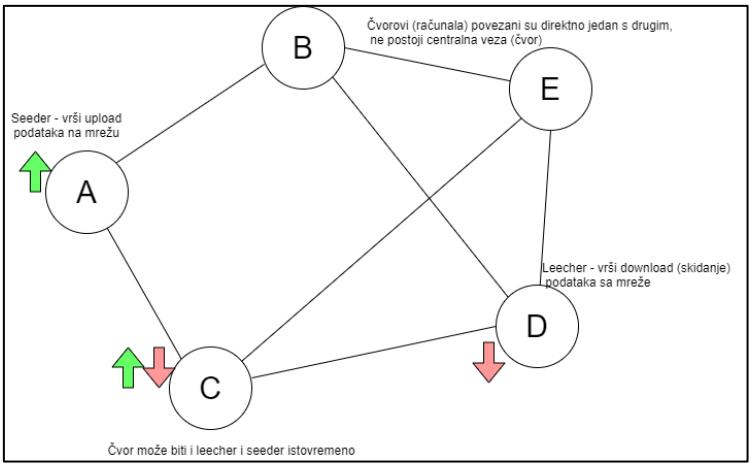
Donja slika prikazuje jedan primjer java web arhitekture. Strana klijenta i baza podataka ista je kao i kod ostalih arhitektura dok je poslužitelj poslovni. Poslužitelj sadrži web kontejner i EJB kontejner. Poslužitelj može biti Apache, GlassFish ili neki drugi koji može raditi sa JSP-om (Java Server Pages). Poslužitelj sadrži poslovnu logiku, web logiku kao i objekte potrebne za pristup podacima baze podataka.



Slika 4: Java arhitektura web aplikacije (Prema: [6])

Postoji još mnogo načina na koje se može konfigurirati arhitektura aplikacije. Jedan od jednostavnijih načina je **Klijent-poslužitelj** (engl. Client-server). Temelj te arhitekture je model zahtjev-odgovor. Klijent šalje zahtjev poslužitelju za informacijom i poslužitelj na to daje odgovor.

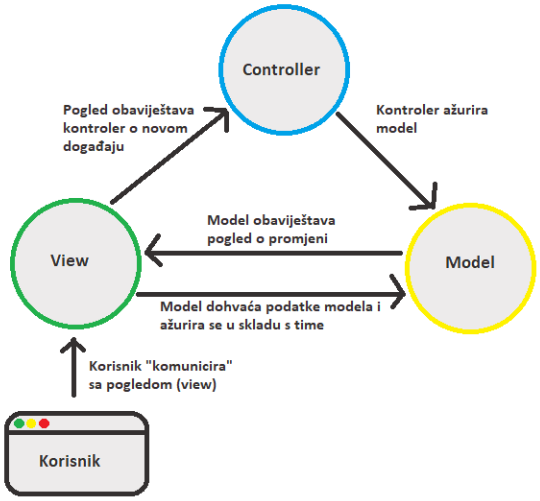
Peer-to-peer ili „od čvora do čvora“ je arhitektura u kojoj računala ili čvorovi mogu komunicirati jedan s drugim bez potrebe za centralnim poslužiteljem. Ukoliko jedan čvor u mreži zakaže i prestane raditi mreža ne „pada“ već nastavlja normalno s radom.



Slika 5: peer-to-peer arhitektura (Prema: [5])

Model-View-Controler (MVC) jedna je od najpoznatijih arhitektura u svijetu web aplikacija. Aplikacijska logika podijeljena je na tri dijela ovisno o funkcionalnosti koju ostvaruje[5]:

- Model – predstavlja logiku spremanja podataka u bazu podataka
- View – komponenta koja je vidljiva korisniku: neki ispis iz aplikacije ili korisničko sučelje
- Controller – komponenta koja predstavlja vezu ili sučelje između modela i pogleda (engl. View)



Slika 6: MVC arhitektura

2.4. Sigurnost web aplikacija

Velikoj većini web aplikacija mogu pristupiti svi korisnici interneta. Isto tako, ukoliko to web aplikacija dopušta, svatko može izraditi račun te se registrirati na neku web aplikaciju (Na primjer: Reddit, Facebook, eZaba i slično). Ti korisnici prilikom korištenja dijelit će svoje podatke s aplikacijom. Web aplikacija mora osigurati **povjerljivost**, **integritet** i **dostupnost** podataka korisnika. U svijetu informatike ta tri pojma predstavljaju **osnovni sigurnosti trokut** (CIA trokut). Povjerljivost (engl. Confidentiality) kao cilj ima osigurati pristup do podataka samo onim osobama koje trebaju imati pristup odnosno dozvolu za pristup tim podacima. Integritet (engl. Integrity) omogućuje promjenu pohranjenih podataka samo onim osobama koje su ovlaštene za tu promjenu. Zadnja komponenta trokuta je dostupnost (engl. Availability) koja omogućava korisnicima da u bilo kojem trenutku mogu pristupiti svojim podacima, potrebnim informacijama ili nekim servisima. Narušavanjem bilo koje od ove tri komponente narušava se sigurnost web aplikacija te se aplikacija koje se uvažavaju ove komponente smatraju ne sigurnima i treba ih izbjegavati. Porastom broja korisnika web aplikacija te vrijednosti podataka koje one pohranjuju, raste i broj napada na web aplikacija. Napad je svaka radnja koja ima za cilj narušiti sigurnost web aplikacija. Bila to krađa podataka, ometanje korisničkog pristupa do podataka ili rušenje same aplikacija.

Sigurnost web aplikacija predstavlja glavnu „komponentu“ bilo kojeg internetskog poslovanja. Pristup internetu s bilo koje lokacije omogućuje napadačima izvršavanje napada različitih veličina i kompleksnosti. Sigurnost web aplikacija bavi se sigurnošću web stranica, web aplikacija te web servisa poput API-a. To je proces zaštite web stranica i online servisa protiv različitih sigurnosnih prijetnji koje žele iskoristiti ranjivost programskog koda. Česte mete za napade su sustavi za upravljanje sadržajem (engl. Content management systems) na primjer WordPress, alati za upravljanje bazom podataka (engl. Database administration tools) poput phpMyAdmin te SaaS (Software as a service) aplikacije. [7]

Web aplikacije su lake mete za napade zbog [7]:

- Kompleksnosti koda – što je kod kompleksniji veća je vjerojatnost da će imati propuste zbog težine održavanja istog koda.
- Nagrade visoke vrijednosti – privatni podaci, bankovne informacije i slično.
- Jednostavnosti izvođenja napada – većina napada može se automatizirati gotovim skriptama i programima te napasti više meta istovremeno.

Gotovo je nemoguće zamisliti neko Internet poslovanje bez dobro razvijene web aplikacija. Evolucijom web aplikacija značajno se povećao broj firmi koje svoje poslovanje vrše

u online obliku. Takva poslovanja često su mete napada te ih je potrebno osigurati.. Postoje 3 važna razloga zašto je sigurnost web aplikacija bitna: [7]

1. **Sprječava gubitak osjetljivih podataka** – cyber kriminalci konstantno traže osjetljive podatke koje bi mogli ukrasti, mreže kojima bi mogli neovlašteno pristupiti i aplikacije koje bi mogli komprimirati. Bez valjana zaštite, organizacije se podliježu riziku gubitka važnih vlastiti kao i korisničkih podataka. Tako je ,naprimjer, u 2015. godini preko 10 milijuna web stranica napadnuto od strane hakera.
2. **Sigurnost je više od samog testiranja** – Većina sigurnosnih testova je automatizirano te efikasnost testova ovisi o osobi koja ih provodi. Bitno je napraviti reviziju sigurnosti prema postojećim normama ju provoditi.
3. **Osigurava očuvanje reputacije poslovanja i smanjuje poslovne gubitke** – gubitak podataka korisnika u većini slučajeva znači gubitak korisnika te povlači mnoge tužbe što znači da će organizacija početi gubiti stečenu reputaciju.

2.4.1. Sigurnost aplikacija otvorenog koda

Aplikacije otvorenog kod su dostupne svim. Mogu biti preuzete besplatno zajedno s pratećom dokumentacijom i vidljivim kodom. Aplikacije zatvorenog koda ili takozvani „proprietary software“ ima zaključan, čuvan, privatni kod koji je dostupan samo osobama zaposlenima u firmi koja razvija tu aplikaciju. Aplikacije otvorenog su s jedne strane sigurnije. Razlog tome je što postoji velika, razvijena zajednica ljudi koji doprinose razvoju projekta. Takve aplikacije razvijaju se jeftino, velike su fleksibilnosti i mogu se razviti u kratkom roku. Lagano se pronalaze sigurnosni rizici i otkrivaju greške koje se u kratkom roku ispravljaju. Takve aplikacije dopuštaju svima da naprave vlastiti pregled koda, preuzmu bilo koju verziju aplikacije ili čak da naprave vlastitu verziju aplikacije s većim ili manjim brojem funkcionalnosti. Ukoliko korisnik koristi aplikacije zatvorenog koda, mora vjerovati vlasniku aplikacije da aplikacije neće zlouporabiti njegove podatke, da će korištenje aplikacije biti sigurno i da će aplikacije izvršavati upravo one funkcionalnosti koje su dogovorene ugovorom.[8]

Unatoč mnogim prednostima aplikacija otvorenog koda, postoje neke mane koje se odnose na sigurnost. Zbog brzog razvoja novog softvera čest se dogodi da se ne naprave sigurnosne provjere ili se one teško održavaju. Također, velik broj ljudi u zajednici nemaju iskustva u radu sa sigurnošću aplikacija. Zato je bitno da se prije korištenja bilo kojeg softvera napravi istraživanje. Potrebno je pročitati komentare drugih ljudi koji su koristili isti softvere, pogledati aktivne probleme (engl. issues) koji nisu riješeni, pogledati koju vrstu enkripcije

softver koristi jer su neki algoritmi probijeni i nesigurni za koristiti i na kraju je potrebno samostalno napraviti odluku o korištenju softvera.

2.4.2.Ranjivosti web aplikacija

Ranjive web aplikacije često su podložne različitim napadima koji nastoje iskoristiti tu ranjivost. Ranjivost web aplikacije može biti u tome da nema dobar mehanizam autentikacije, ima otvorene portove preko kojih se napadač može spojiti na mrežu, izvodi neki servis koji je dostupan napadaču i slično. Napadi na web aplikacije mogu biti različite veličine. Mogu ciljati na neku bazu podataka kako bi dobili pristup i naštetili integritetu podataka ili mogu napasti cijelu mrežu odnosno poslužitelj na kojem se izvodi web aplikacija te time onemogućiti izvođenje same aplikacije.

Neki od poznatih napada su:[7]

- Cross site scripting (XSS)
- SQL injection
- Denial of service (DoS) napad
- Korupcija memorije(engl. Memory corruption)
- Buffer overflow
- Cross-site request forgery (CSRF)
- Krađa podataka (engl. Data breach)

2.4.3. Preporuke za sigurnu web aplikaciju

Uz Vatrozid, jednu od najpoznatijih metoda zaštite softvera, postoje još metoda za zaštitu web aplikacija. Navedeni procesi trebali bi biti provedeni prilikom izgradnje bilo koje web aplikacije [7]:

- **Prikupljanje informacija** – potrebno je ručno pregledati cijelo ukupnu aplikaciju. Identificirati moguće točke upada preko kojih bi napadač mogao „uči“ u aplikaciju te pregledati kod na strani klijenta kako ne bi sadržavao ne željene informacije koje bi napadač mogao iskoristiti.
- **Autorizacija** – aplikacija treba biti testirana za „path traversal“ napad. Prilikom takvog napada, napadač „isprobava“ različite URL putanje koje unosi u tražilicu ne bi li dobio pristup nekoj lokaciji koja je namijenjena samo za administratora.
- **Kriptografija** – potrebno je koristiti najpouzdanije algoritme za enkripciju osjetljivih podataka (lozinke, email adrese i slično)

- **Zaštita od uskraćivanja usluge** – aplikaciju je potrebno testirati protiv DoS napada provedbom protu–automatskih testiranja te zaštitom zaključavana računa korisnika.

2.4.3.1. OWASP

OWASP (Open Web Application Security Project) je online zajednica koja izrađuje besplatne alate, dokumentaciju i tehnologije koje pomažu korisnicima osigurati vlastite web stranice, web aplikacije i ostale resurse na internetu. Primarni fokus OWASP-a je u web sigurnosti, aplikacijskoj sigurnosti te procjeni ranjivosti.

Ustanovljeno je deset osnovnih principa prema kojima aplikacija „gradi“ svoju sigurnosti. Ti principi dobiveni su kombinacijom OWASP-a i knjige „Writing Secure Code“ napisane od strane Michael Howard-a i David LeBlanc-a. Principi su sljedeći[9]:

1. **Minimizirati područje koje može biti napadnuto** –Ograničavanjem mogućnosti koje se nude korisniku smanjuje se ranjivost.
2. **Postaviti sigurnosne standarde –** moraju postojati zadovoljivi standardi za postupanje s korisničkom registracijom, prijavom, kompleksnošću lozinke, e-mail verifikacije i slično.
3. **Princip najmanje privilegije –** korisnik mora imati najmanje moguć skup privilegija koje su mu potrebne za obavljanje nekog zadatka.
4. **Princip dubinske obrane (engl. The principle of Defence in depth) –** za ostvarivanje najveće razine sigurnosti najbolje je imati više sigurnosnih kontrola koja pristupaju riziku na različite načine kako bi osigurale aplikaciju.
5. **Fail securely –** pojam koji se koristi kada se neka transakcija ne izvrši. Transakcija mora biti izvršena u potpunosti ili potpuno odbačena.
6. **Uključivati usluge treće strane s oprezom**
7. **Odvajanje odgovornosti –** korisnike neke web aplikacije ne bi trebao biti i njen administrator jer bi sebi mogao dati besplatne proizvode ili informacije.
8. **Izbjegavati ostvarivanje sigurnosti skrivanjem –** ukoliko se aplikacija oslanja na skrivanje administratorskog URL-a kako bi bila sigurna, tada ona uopće nije sigurna.
9. **Sigurnost treba biti jednostavna i funkcionalna**
10. **Sigurnosne greške moraju biti popravljene ispravno** – Ukoliko se pojavi neke greška, ona mora biti ispravljena u korijenu.

2.4.3.2. Sigurnost Java web aplikacija

Postoji velik broj razvojnih okruženja koji podržavaju razvoj java web aplikacija (Eclipse, Netbeans, IntelliJ,..). Također, omogućuju i konfiguraciju poslužitelja na kojem će se aplikacija izvoditi. Ispravno konfiguriran poslužitelj čini aplikaciju sigurnijom.

Web aplikacija može imati korisnike s različitim ulogama. Dva osnovna tipa uloga su: Korisnik (engl. User) i Administrator. Administrator bi u pravilo trebao imati sva prava koja se odnose na upravljanje i postavljanje aplikacije. Korisnik aplikacije ne bi trebao vidjeti administratorske mogućnosti niti ih ne bi trebao moći koristiti. Iz tog razloga postoji više načina na koji se korisnik prijavljuje u aplikaciju [10]:

- **Osnovna prijava (engl. Basic Login)** – Koristi se osnovni način autentifikacije, prozor za korisničku prijavu pruža web preglednik. Potrebno je unijeti valjanu lozinku i korisničko ime. Za korištenje ovakvog načina prijave potrebno je podesiti poslužitelj. Moguće je koristiti Glassfish poslužitelj te ispravno podesiti web.xml datoteku s podacima uloga.
- **Prijavni obrazac (engl. Form login)** – za prijavu koristi se forma koju je moguće oblikovati. Konfiguracija je ista kao i kod osnovne forme za prijavu uz dodatak kreiranja stranica za prijavu i za greške. Radi se o JSP (Java Server page) i HTML stranicama.

Kako bi se programerima olakšala implementacija sigurnosti u njihove web aplikacije, kreirani su različiti okviri koji daju strukturu aplikacije. Neki od poznatijih su [11]:

- **JAAS (Java Authentication and Authorization Services)** – API za sigurnost koji se koristi za korisničku autentifikaciju i autorizaciju. Integriran je u JDK verziju 1.4.
- **Spring security** – može se lako prilagoditi potrebama aplikacije, koristiti se za rukovanje s prijavom i kontrolom pristupa. Lagan je za razumjeti i naučiti.
- **Apache Shiro** – obavlja autorizaciju, kriptiranje te upravljanje sesijom na svim vrstama java aplikacija, neovisno o njihovoj veličini. Okvir je neovisan o tehnologiji te radi na svim Java strukturama.
- **HDIV (HTTP Data Integrity Validator)** – daje dodatne funkcionalnosti aplikaciji poput sigurnosnih funkcija održavanja API-a, definiranje specifikacija okvira. Programerima pruža transparentnost bez dodavanje kompleksnosti u razvoju aplikacije.
- **OACC** – služi za kontrolu pristupa te djeluje kao API za upravljanje autentifikacijom i autorizacijom. Daje fleksibilan model sigurnosti.

2.5. Tehnologije za razvoj web aplikacija

Razvoj web aplikacija zahtjeva dobro poznavanje osnovnih tehnologija vezanih za razvoj kao i tehnologija koje olakšavaju sam razvoj, poput okvira (engl. Framework) na primjer. Poznavanje tehnologija i pravila razvoja omogućava programerima razvoj web aplikacije željene funkcionalnosti koja ostvaruje zadane ciljeve. Poznavanje jezika za razvoj klijentske strane (JavaScript, CSS, HTML) i barem jednog jezika za razvoj serverske strane (Java na primjer) minimalno je potrebno za razvoj jednostavnije web aplikacije. Vogel L. [24] navodi 9 bitnih tehnologija za sam razvoj, a to su:

- Web preglednici
- Jezici za razvoj klijentske strane – HTML i CSS
- Razvojni okviri (engl. Web Development Frameworks)
- Jezici za razvoj serverske strane – JS, Java, Python, PHP, Ruby,..
- Protokoli – na primjer HTTP
- API
- Formati podataka – Json, XML,CSV
- Klijent – strana klijenta (računalo, mobitel,..)
- Poslužitelj

Već je poznato da se web aplikacijama pristupa putem **web preglednika**. Oni prikazuju informacije i podatke dobivene od strane web aplikacija te se bave poslovima klijenta. Primaju unose od korisnika te šalju zahtjeve web aplikaciji. Uvelike se koriste i kod faze razvoja. Služe za testiranje same web aplikacije. Svaka web aplikacija trebala bi se izvoditi jednako na svim preglednicima. Svaki web preglednik treba biti u mogućnosti razumjeti osnovni jezik HTMLa. Naravno postoje web preglednici koji ne mogu interpretirati sve elemente HTMLa. Naprimjer Internet Explorer i Firefox nisu mogli razumjeti element za unos boja (color) u nekim svojim verzijama. Neki od poznatijih preglednika su: Chrome, Firefox, Safari, Opera i Internet Explorer.

Kako bi se sadržaj mogao prikazati korisnicima potrebno je koristiti jezike za programiranje klijentske strane. Ti jezici su **HTML i CSS**, te u većem broju slučaja i JavaScript. HTML je osnovni jezik kojeg bi svaki web programer trebao poznavati. Taj jezik preglednik može razumjeti i prikazati sadržaj koji je oblikovan putem HTMLa. CSS (Cascading Style Sheets) služi za oblikovanje sadržaja HTMLa. Ne služi za implementiranje logike već definira izgled same web stranice. Na kraju, JavaScript daje dinamiku web stranicama. Može se koristiti za dodavanje svojstva elementima HTMLa tijekom izvršavanja aplikacije te za

implementiranje logike na korisničkoj strani. JavaScript se također može koristiti i na poslužiteljskoj strani te nije nužno vezan za klijentsku stranu.

Okviri omogućuju programerima da izbjegnu rješavanje jednostavnih i ponavljajućih zadataka. Daju kostur rješenja te omogućuju programerima da strukturirano riješe probleme i implementiraju određene funkcionalnosti. Ugrađene biblioteke okvira smanjuju količinu koda koju programer mora napisati. Radi se o ponavljajućem kodu poput: validacija, spajanja na bazu, prijenos datoteka, upravljanja kolačićima, autorizacijama i slično. Okviri su često dobro dokumentirani pa se programer ni za taj dio ne treba brinuti. Jedna od velikih prednosti okvira je standardizacija i držanje konvencije. Okviri omogućuju lakše imenovanje klasa, varijabli, funkcija, tablica baza i ostalih aspekata aplikacije. Definiraju arhitekturu aplikacije te omogućuju programerima da se više fokusiraju na izradu funkcionalnosti, a manje na konfiguraciju aplikacije. Postoji velik broj dostupnih okvira, njihov odabir ovisi o tehnologijama koje programer planira koristiti kao i osobnoj preferenciji. Angular je okvir za razvoj dinamičnih web aplikacija. Ovaj okvir omogućuje kreiranje klijentske strane bez potrebe korištenja drugih okvira ili dodataka. Značajke koje Angular daje su: gotovi predlošci, MVC arhitektura, generiranje koda, razdvajanje koda i slično. Sljedeći poznati okvir je Ruby on Rails. Ovaj okvir koristi se s poslužiteljske strane i omogućuje brži i jednostavniji razvoj aplikacija. Omogućuje ponovo korištenje koda i pojednostavljuje sam proces programiranja. Od ostalih okvira može se spomenuti YII - okvir za razvoj web aplikacija otvorenog koda, Meteor JS koji je napisan u Node.js-u, Express.js koji je odličan za brzi razvoj aplikacija i API-a, Zend – okvir otvorenog koda baziran na PHP-u, Django – jedan od popularnijih okvira napisan u Pythonu koji prati MVC arhitekturu te na kraju Laravel – okvir koji je idealan za manje web stranice.[12]

Funkcionalnosti aplikacije realizirane su pomoću **programskih jezika.** Svaka obrada podataka, dohvat podataka ili neka operacija realizirana je putem nekog jezika. Rečeno je da se za rad klijentske strane koriste HTML i CSS. Za programiranje poslužiteljske strane mogu se koristiti mnogi jezici. Neki poznatiji su: Java, Python, Ruby, PHP ili JavaScript.

Za komunikaciju između klijenta i poslužitelja koriste se **protokoli**. Protokoli sadrže instrukcije o tome kako prenijeti informaciju s jednog računala na drugo udaljeno računalo. Točnije, protokoli služe za komunikaciju. Najpoznatiji je HTTP protokol. Zahvaljujući njemu, klijent može pristupiti web aplikaciji šaljući zahtjev. Odgovor koji dobiva je HTML stranica web aplikacije. Poznat je i protokol DDP (Datagram Delivery Protocol). On koristi Web Socket-e da stvori stabilnu konekciju između klijenta i poslužitelja. Rezultat toga je web stranica koja se ažurira u stvarnom vremenu bez potrebe osvježavanje preglednika. Važan je i protokol REST koji se većinom koristi kod API-a. Koristi standardne metode poput GET, POST i PUT kako bi razmijenio informacije između aplikacija.[5]

Web aplikacije često koriste funkcionalnosti koje pružaju neke druge aplikacije u obliku **API**-a. API omogućava programerima da koriste funkcionalnosti iz druge aplikacije bez poznavanja koda te jedino što im je potrebno je API ključ. Mnoge poznate web aplikacije pružaju API-e preko kojih dijele podatke. Neke od poznatijih su Facebook i Google.

Aplikacije tijekom svojeg životnog ciklusa obrađuju velik broj podataka. Iz tog razloga postoje standardizirani **formati zapisa podataka**. Razlikuju se tri najpoznatija formata zapisa podataka koje bi trebala prepoznati svaka web aplikacija: JSON, XML i CSV. Ti formati spremaju podatke na način da se kasnije mogu pročitati u bilo kojem programu koji zna raditi s tim formatima.

O **klijentu** i **poslužitelju** govorilo se dosta u prijašnjim poglavljima. Svaki korisnik aplikacije naziva se klijentom. On koristi računalo, mobitel ili neki drugi uređaj kako bi slao zahtjeve nekoj aplikaciji odnosno poslužitelju. Poslužitelj je udaljeno računalo koje je sposobno primiti i obraditi zahtjev klijenta te mu poslati odgovarajući odgovor.

Tehnologija koju je također važno spomenuti je odabrano **razvojno okruženje (IDE)** aplikacije. IDE olakšava razvoj aplikacije na način da vrši njeno kompajliranje, izvođenje, te stavljanje na poslužitelj. Može olakšati i pisanje koda na način da omogućuje korisniku da sam umetne jednostavni kod (getter i setter na primjer) te ubrzava pisanje samog koda. Neki od poznatijih razvojnih okruženja su Visual studio, Eclipse, Netbeans, JetBrains, IntelliJ i slično.

2.6. Prednosti i nedostaci web aplikacija

Prednosti i nedostatke moguće je navesti kod svakog softvera pa tako i kod web aplikacija. Velik broj prednosti samih web aplikacija moguće je prepoznati u prijašnjim poglavljima. U ovom poglavlju, prednosti i nedostaci navest će se putem usporedbe s ostalim vrstama aplikacija.

2.6.1. Usporedba web aplikacija i mobilnih aplikacija

Neka aplikacija može se pojaviti u više oblika. Česti je slučaj da web aplikacija ima i svoju mobilnu verziju. Mobilne aplikacije dizajnirane su za izvođenje na određenom operacijskom sustavu i koriste ugrađene funkcionalnosti mobilnih uređaja. Mobilne aplikacije su znatno skuplje za razviti od web aplikacija pogotovo ako je aplikaciju potrebno razviti za različite operacijske sustave : Android, iOS ili Windows. S druge strane, web aplikacije su neovisne o operacijskom sustavu i jeftinije su za razviti. Iako su jeftinije, nemaju mogućnosti koristiti funkcionalnosti uređaji kao ni resurse uređaja.

Prednosti web aplikacija nad mobilnim aplikacijama su sljedeće [13]:

- **Dostupnost** – web aplikaciji može pristupiti svaki uređaj koji ima pristup internetu i instaliran web preglednik bez obzira na operacijski sustav uređaja. S druge strane, mobilne aplikacije zahtijevaju instalaciju na uređaju te kompatibilnost uređaja s aplikacijom. Na primjer, mobilni uređaj s manjom verzijom android sustava ili s nedovoljno RAM memorije neće moći instalirati aplikaciju.
- **Ažuriranje aplikacije nije potrebno** – Mobilne aplikacije zahtijevaju konstantno ažuriranje kako bi pružile sve funkcionalnosti. To zahtijeva od korisnika da ručno preuzme svako novo ažuriranje. To nije slučaj s web aplikacijama. Programeri nakon što naprave izmijene koda mogu lagano objaviti novu verziju aplikacije na poslužitelj gdje korisnik ne mora vršiti preuzimanje aplikacije.
- **Manja cijena razvoja aplikacije** – jedna web aplikacija dovoljna je za pokretanje na svim operacijskim sustavima. Mobilne aplikacije zahtijevaju paralelan razvoj za sve operacijske sisteme.
- **Web aplikaciju može obaviti svatko** – za razliku od mobilnih aplikacija, web aplikaciju može objaviti svatko. Jedino što je potrebno je unikatna domena preko koje se pristupa aplikaciji. Mobilne aplikacije zahtijevaju odobrenje od strane aplikacijske trgovine (engl. App Store) kako bi bile objavljene.
- **Instalacija aplikacije nije potrebna** – mobilne aplikacije zahtijevaju instalaciju kako bi se mogle koristiti. Korisnik mora odvojiti dio svoje memorije za pohranu same aplikacije. To nije slučaj s web aplikacijama. One se mogu pokretati instalacije na uređaj.

Nedostaci web aplikacija u usporedbi s mobilnim aplikacijama su sljedeći [13]:

- **Limitirani pristup resursima i značajkama mobilnog uređaja** – web aplikacije često nemaju pristup nekim značajkama uređaja poput lokacije ili kamere. Mobilne aplikacije ima dostupan procesor kao i ram memoriju što ih čini bržima, također mogu koristiti značajke poput: lokacije, kamere, mikrofona bez gubitka performansi.
- **Stabilnosti** – ne optimizirane web aplikacije uzrokovat će rušenje web preglednika. Mobilne aplikacije mogu koristiti snagu procesora za izvršavanje težih zadataka.
- **Konstanta potrebna za Internet konekcijom** – web aplikacije zahtijevaju Internet za korištenje. Bez interneta ne može se pristupiti web aplikaciji. S druge

strane mobilne aplikacije (koje su vezane za Internet, na primjer: Reddit) mogu izvršavati dio svojih funkcionalnosti bez potrebe da budu povezane na Internet.

- **Ne postoji „App Store“ za web aplikacije** – većina korisnika naviknuta je tražiti aplikacije preko neke trgovine. Mobilne aplikacije lagano se nalaze na poznatim trgovinama dok s druge strane web aplikacije mogu se pronaći preko tražilice ako korisnik zna što traži.

2.6.2. Usporedba web aplikacija i stolnih aplikacija

Stolna aplikacija je vrsta aplikacije koja je instalirana na osobnom računalu. Aplikacija se može pokrenuti neovisno o drugim aplikacijama. Takve aplikacije zauzimaju mjesto na računalu i većina ne zahtijeva Internet konekciju. Neke poznatije stolne aplikacije su: Word, Excel, Powerpoint, Photoshop i slično. Veličina same aplikacije ovisi o broju funkcionalnosti i načinu pohrane podataka. Stolna aplikacije većinom podatke pohranjuju lokalnu, no neke omogućuju pohranu podataka u oblak pa tako reduciraju količinu podataka koje osobno računalo mora pohraniti. Prednosti i nedostaci web aplikacija nad stolnim aplikacijama slične su onima kao i kod mobilnih uređaja uz par iznimki.

Prednosti web aplikacija nad stolnim aplikacijama su [14]:

- Web aplikaciju **nije potrebno instalirati lokalno** na računalo što rezultira uštedom prostora na tvrdom disku za razliku od stolnih aplikacija.
- **Automatska ažuriranja** – Web aplikacija automatski je ažurirana na poslužitelju bez da korisnik treba poduzeti neke radnje, stolne aplikacije zahtijevaju potvrdu korisnika prilikom preuzimanja ažuriranja ili čak ručno preuzimanje ažuriranja.
- **Neovisnost o operacijskom sustavu** – već je poznato da su web aplikacije neovisne o operacijskom sustavu, no stolne aplikacije još uvijek nisu. Iako postoji manji broj stolnih aplikacija koje mogu raditi na više operacijskih sustava, većina ih zahtjeva zaseban razvoj aplikacija za sve platforme. Tako prilikom preuzimanja stolnih aplikacije postoje opcije za različite operacijske sustava, te opcije zasebno se razvijaju što rezultira skupljim razvijem samih stolnih aplikacija.
- **Mobilni pristup aplikaciji** – web aplikaciji može se pristupiti s bilo koje lokacije i na bilo kojem uređaju. Stolne aplikacije vezane su za osobno računalo na kojem su instalirane i ne mogu biti prenosive. Podaci web aplikacije su također globalno dostupi, dok su podaci stolnih aplikacije u većini slučajeva pohranjeni lokalno.

- **Manji utrošak resursa računala** – većinu poslova web aplikacije obavlja poslužitelj što rezultira uštedom resursa računala. Stolne aplikacije mogu značajno opteretiti računalo ukoliko nisu dobro optimizirane. S druge strane ovo može biti i nedostatak jer dobro optimizirana stolna aplikacija je značajno brža.

Nedostaci web aplikacija o odnosu na stolne aplikacije su [14]:

- **Sigurnosne prijetnje** – sigurnost podataka kojima web aplikacija rukuje ovisi o razini sigurnosti koje vlasnik aplikacije provodi. Stolne aplikacije koje svoje podatke pohranjuju lokalno sigurnije su web aplikacija. One moraju koristiti kriptografiju za sigurno spremanje podatka korisnika.
- **Ovisnost o internetu** – stolne aplikacije većinom ne zahtijevaju pristup internetu, a aplikacije koje ga zahtijevaju omogućuju korištenje nekih svojih funkcionalnosti unatoč nedostatku interneta. Bez interneta pristupiti web aplikaciji je nemoguće.
- **Sporije su od stolnih aplikacije** – već je bilo više puta spomenuti da web aplikacije slabo troše resurse računala pa to rezultira i potencijalno lošijim performansama.

2.6.3. Usporedba web aplikacija i web mjesta

Web mjesto je skup globalno dostupnih web stranica koje su međusobno povezane unutarnjim vezama i nalaze se pod istom domenom. Razvija ih i održava jedna osoba, organizacija ili neko poslovanje. Web mjestu se pristupa putem interneta. Karakteristike koje definiraju web mjestu su:

- Sadrže kvalitetan i relevantan sadržaj
- Sadrži navigaciju koju je lagano razumjeti i kvalitetan web dizajn
- Može se lagano pronaći koristeći web tražilicu.

Web aplikacije razlikuju se od web mjesta. One moraju biti skalabilne, prilagoditi se svakoj platformi, modularne i testirane. Donja tablica prikazuje usporedbu web aplikacija i web mjesta po nekim glavnim parametrima. Prikazuje njihove sličnosti i razlike.

Tablica 1: Usporedba web aplikacije i web mjesta (Prema [14])

Parametar	*Web aplikacija*	*Web mjesta*
Korisnici	Krajnji korisnici koji vrše interakciju s aplikacijom.	Većinom sadrži statični sadržaj koji je javno dostupan svim posjetiteljima.

Korisnička interakcija	Uz čitanje sadržaja, korisnik može kreirati i upravljati vlastitim podacima.	Korisnik može vidjeti ili čitati sadržaj ali na njega ne može utjecati.
Autentikacija	Zahtijevaju prijavu u sustav kako bi korisnik mogao koristiti veliki spektar funkcionalnosti.	Prijava nije nužna, korisnik se može prijaviti u sustav kako bi dobivao obavijesti.
Zadaci i kompleksnost	Funkcionalnosti su znatno kompleksnije nego kod web stranica.	Web stranica prikazuje samo prikupljene podatke i informacije.
Vrsta softvera	Web aplikacija dio je web mjesta. Sama po sebi ne predstavlja web mjesto.	Web mjesto je kompletan produkt kojem se pristupa preko preglednika.

3. Razvoj web aplikacija otvorenog i zatvorenog koda

Ovo poglavlje fokusira se na aplikacije otvorenog koda. Aplikacije čiji je kod javno dostupan i čiji životni vijek ovisi o zajednici. Naravno, dana je i definicija aplikacija zatvorenog koda odnosno te aplikacije poznate su pod nazivom vlasnički softver (engl. Proprietary software).

Prva dva dijela ovog poglavlja posvećena su aplikacijama otvorenog i aplikacijama zatvorenog koda. Dane su definicije te su navedeni prednosti i nedostaci tih vrsta aplikacija. Navedeno je i par primjera stvarnih web aplikacija za svaku vrstu.

Poglavlje se uvelike fokusira i na razvoj web aplikacija. Navedene su i detaljno opisane faze razvoja odnosno koraci u razvoju web aplikacije. Spomenute su i osobe koje sudjeluju u razvoju. Pošto je tema ovog rada razvoj web aplikacije u programskom jeziku Java, dan je fokus i na tehnologije koje se koriste prilikom razvoja web aplikacije u Javi. Pa su tako opisani popularni API-i , JSP-ovi i JSF-ovi. Postoji velika razlika u razvoju web aplikacija otvorenog koda od razvoja aplikacija zatvorenog koda. Različiti su koraci, metodologije kao i odgovornosti. Sve je to opisano u zasebnom dijelu. Razvoj web aplikacija prate razni izazovi. Ti izazovi mogu poboljšati kvalitetu aplikacije ili ju smanjiti ovisno o načinu rješavanja

određenog izazova. Jedan dio poglavlja fokusira se na opis izazova ili problema koji se mogu javiti tijekom faza razvoja web aplikacije.

U zasebnom poglavlju navedeni su principi u razviju web aplikacije. Spominju se poznati principi poput: fokusiranja na korisnika, implementacije sigurnosti, korištenja alata, optimizacije ali i oni principi koji se javljaju prilikom razvoja: korištenje repozitorija i konzistentne arhitekture na primjer. Uvažavanje principa bitno je za stvaranje kvalitetnog softvera.

Kao zaključak ovog poglavlja, dana je usporedba aplikacija otvorenog koda i aplikacija zatvorenog koda. Radi se o usporedbi koja uvažava sva prijašnja poglavlja te daje jasnu sliku koje se točno razlike između te dvije vrste aplikacija.

3.1. Aplikacije otvorenog koda

Pojam otvorenog koda (engl. Open Source) opisuje način izrade programa na način da je program javno dostupan i može ga ažurirati i mijenjati bilo tko. Koristi se u kontekstu izrade softvera i opisuje specifičan pristup izradi aplikacija. Danas, taj je pojam dosta proširen i razvijen. Uključuje otvorene projekte, proizvode, otvorene razmjene, kolaborativan razvoj, brzi razvoj, transparentnost i razvoj orijentiran na zajednicu. Programi otvorenog koda su softver čiji izvorni kod bilo tko može vidjeti, modificirati ili poboljšati. Kod većine aplikacija, izvorni kod nedostupan je krajnjim korisnicima. Taj kod programeri mogu modificirati i promijeniti način na koji neki program ili aplikacija funkcionira. Programeri koji imaju pristup izvornom kodu mogu dodati nove funkcionalnosti te popraviti dijelove koda koji ne funkcioniraju na ispravan način. [15]

Programi otvorenog koda su programi koji su distribuirani zajedno s svojim izvornim kodom. Dostupni su za korištenje, izmjenu i distribuciju svima koji prihvate pravila korištenja tog programa. Ti programi najčešće uključuju licence koje omogućuju programerima da prilagode određeni program za korištenje ovisno o vlastitim potrebama te im daju kontrolu na distribucijom programa. Ideja o programima otvorenog koda javila se 1983. godine i njenim začetnikom smatra se Richard Stallman, programer koji je tada radio u MIT-u. Stallman je smatrao da bi softver trebao biti dostupan svim programerima kako bi ga se moglo modificirati po želji, s ciljem razumijevanja, učenja i poboljšanja koda. Stallman je počeo pisati vlastiti besplatni kod pod vlastitom licencom zvanom „GNU Public License“. Takav pristup brzo se proširio i nastala je takozvana Open Source Inicijativa (OSI) 1998. godine. Open Source Inicijativa (OSI) kreirana je sa svrhom promoviranja i zaštite programa otvorenog koda i njene zajednice. OSI djeluje kao centralni informativni repozitorij za softver otvorenog koda. Daje

pravila i smjernice za korištenje programa otvorenog koda te daje informacije o licencama koda, potporu kodu i zbližava zajednice ljudi koji razvijaju softver. [15]

OSI je neprofitna organizacija koja djeluje kao vodeća organizacija za softvere otvorenog koda. Njena definicija softvera otvorenog koda uključuje više kriterija koji se svode na [16]:

- Preraspodjelu softvera
- Dostupnost i cjelovitost izvornog koda
- Distribuciju i svojstva licenci
- Izvedena djela
- Antidiskriminacija

Softver otvorenog koda prilikom svojeg puštanja u javnost definira licence koje definiraju u kojoj mjeri je izvorni kod „legalan“ za korištenje krajnjim korisnicima. Postoje mnogo licenci o kojim se govori kasnije u poglavlju. Neki program smatra se programom otvorenog koda ako [15]:

- **Je dostupan s izvornim kodom koji je besplatan** – korisnici mogu pregledati kod koji čini program i mogu raditi izmjene nad njim.
- **Izvorni kod može se ponovo iskoristiti za izgradnju novog softvera** – svatko može uzeti izvorni kod i kreirati vlastiti program koristeći ga.

Otvoreni izvorni kod obično je pohranjen na nekom javnom repozitoriju i podijeljen je s javnošću. Najpoznatiji javni repozitorijI za dijeljenje otvorenog softvera su GitHub i GitLab. Javnom repozitoriju može pristupiti svatko te koristiti kod neovisno o drugim osobama kao i napraviti svoj doprinos dizajnu i funkcionalnosti samog projekta. Postavlja se pitanje da li je softver otvorenog koda slobodan od grešaka ili bug-ova. U većini slučajeva, nije. Na razvoju nekog softvera radi veći broj osoba te je kod konstantno modificiran i poboljšan. Kvaliteta samog projekta ovisi o znanju osoba koje rade na njemu. Otvoreni softver oko sebe okuplja ljude s različitim spektrom znanja te je gotovo neizbježno da će neki softver imati mane u sigurnosti, performansama te kvaliteti. S druge strane, zajednica ljudi brže pronalazi greške u kodu te su one ispravljene u kratkom roku. Bez obzira na vrstu softvera, greške će se uvijek pojaviti. Razlika između otvorenog softvera i komercijalnog softvera leži u tome tko je odgovoran za ispravljanje istih grešaka. Kod komercijalnog softvera, proizvođač mora ispraviti grešku dok kod otvorenog softvera grešku može ispraviti i sam korisnik.[15]

Aplikacije otvorenog koda većinom su dostupne besplatno. Također, ništa ne sprječava proizvođača nekog softvera da naplati softver korisniku te dozvoli redistribuciju aplikacije i izvornog koda. Kada se govori o softveru otvorenog koda tada se ne podrazumijeva uvijek da

je besplatan. Ono što korisnici cijene je sloboda. Sloboda u smisli vlastite modifikacije koda kako bi se neki program prilagodio potrebama korisnika. Korisnik tada može mijenjati izgled programa, dodati vlastite funkcionalnosti te poboljšati trenutne funkcionalnosti. Ukratko, besplatan softver i softver otvorenog koda dvije su različite stvari. Komercijalni softver može također biti i besplatan softver. Proizvođač tada daje vlastiti softver besplatno na korištenje, no korisnik ne dobiva izvorni kod nego samo izvršnu verziju programa. Otvoreni kod više se fokusira na slobodu prilagodbe softvera za vlastite potrebe, može biti besplatan, ali to nije nužno uvjet. [16]

Kroz ovo poglavlje moguće je prepoznati potencijal softvera otvorenog koda. Neke od glavnih prednosti koje definiraju takav softver su [16]:

- Njegova se kvaliteta može lako i uvelike poboljšati kada je izvorni kod javno dostupan te konstantno testiran i ispravljen.
- Nudi priliku za učenje programerima. Daje im mogućnost da primjene svoje vještine na najpopularnije programe koji su danas dostupni.
- Može biti znatno sigurniji od komercijalnog softvera jer su greške brzo pronađene i ispravljene.
- Budući da je u javnoj domeni i stalno podliježe ažuriranjima, male su šanse da će postati nedostupan ili brzo zastarjeti - što je veliki plus za dugoročne projekte.

Preferencija ljudi za korištenje otvorenog softvera znatno je porasla. Postoji mnogo razloga zašto se korisnici odlučuju za takav softver, a glavni razlozi su [17]:

- **Kontrola** – korisnici preferiraju programer otvorenog koda jer im nude veću kontrolu nad aplikacijom. Mogu pregledati kod kako bi se uvjerili da ne radi ono što ne bi trebao te mogu promijeniti dijelove koda ukoliko žele. Korisnici koji nisu programeri također imaju korist od takvog softvera jer ga mogu koristiti u bilo koje svrhe, a ne samo one za koje je proizvođač to predvidio.
- **Trening** – otvoreni kod pruža priliku ta učenjem. Omogućava programerima da pregledaju javni kod te nauče kako izraditi bolji softver. Programeri mogu razmjenjivati kod s drugim osobama te ga kritički vrednovati. Dobivanjem povratnih informacija. Programeri mogu naučiti kako napisati bolji kod koji je bez grešaka i koji je dobro optimiziran.
- **Sigurnost** – većina korisnika preferira programe otvorenog koda jer smatraju da su sigurniji i stabilniji od komercijalnog softvera. Zbog toga što je kod javno dostupan, omogućuje se svakom korisniku da pregleda kod te podijeli greške i propuste s drugim osobama koje tada mogu napraviti određene izmjene. Pošto

programeri ne moraju tražiti dopuštenje od originalnog autora kako bi izmijenili kod, oni mogu popraviti, ažurirati ili unaprijediti kod znatno brže nego što bi to učinili kod komercijalnog softvera.

- **Stabilnost** – pokazalo se da je razvoj softvera otvorenog koda bolje rješenje za duge projekte neko komercijalno rješenje. Korisnici takvog softvera mogu biti sigurni da će se softver konstantno unaprjeđivati i održavati jer ne ovisi o odluci kompanije koja bi mogla eventualno ugasiti (engl. terminate) usluge softvera nakon određenog vremenskog razvoja.
- **Zajednica** – otvoreni softver okuplja zajednicu ljudi oko sebe. Ta zajednica koristi taj softver, unaprjeđuje ga, održava ga te ga promovira. Zajednica se konstantno proširuje jer korisnici vide prednosti otvorenog razvoja koda nad komercijalnim razvojem.

Kao i programeri, korisnici također imaju velike koristi od softvera otvorenog koda. Većina takvog softvera he besplatna za korištenje. Najpoznatiji softver otvorenog koda vjerojatno je Linux. Za razliku od Windowsa, Linux može biti instaliran i distribuiran bezbroj puta bez ikakvih ograničenja. Ta značajka bitna je za poslužitelje. Ukoliko korisnik želi podesiti poslužitelj, može koristiti Linux bez da se treba brinuti o licenciranju i ograničenju broja instanci operacijskog sustava. Velika prednost je i fleksibilnost. Na primjer, prilikom prelaska na Windows 8, korisnici su bili nezadovoljni izgledom korisničkog sučelja. Windows operacijski sustav je zatvorenog koda te korisnici nisu bili u mogućnosti koristiti sučelje iz Windows sedmice i funkcionalnosti Windowsa 8. Korisnici Linuxa s druge strane mogu modificirati izgled korisničkog sučelja po želji. Naravno, to zahtijeva neko znanje iz programiranja, no na javnim repozitorijima postoji velik broj gotovih rješenja koje je samo potrebno integrirati u postojeći operacijski sustav. Otvoreni kod također omogućuje i velikim korporacijama izgradnju vlastitog softvera. Tako je, na primjer, Apple-ov operacijski sustav (iOS) izgrađen na temelju otvorenog koda.

Tehnologije otvorenog koda pomogle su uspostaviti veći dio interneta. Mnogi svakodnevno korišteni programi se temelje na tehnologijama otvorenog koda. Na primjer: Android operacijski sustav i Apple Operacijski sustav temelje se na kernelu (jezgri) i Unix / BSD tehnologijama otvorenog koda. Neki popularni softveri otvorenog koda koji se često koriste su [16]:

- Mozilla's Firefox web preglednik
- Thunderbird email klijent
- PHP skriptni jezik
- Python programski jezik

- Apache HTTP web poslužitelj

3.2. Aplikacije zatvorenog koda

Softver zatvorenog koda (engl. Closed Source) ili vlasnički softver (engl. Proprietary Software) je softver čije izvorni kod nije dostupan korisniku. On predstavlja suprotnost aplikacijama otvorenog koda. Takav softver u vlasništvu je neke kompanije ili individue te podliježe „copyright“ zakonima. Autor softvera ili kompanija u vlasništvu softvera ima kompletan nadzor nad njegovim razvojem. Model poslovanja proizvodnje vlasničkog softvera je jednostavan: prodati proizvod i zaraditi novce. U skladu s time, izvorni kod softvera je tajna. Izvorni kod otkriva načina na koji je softver razvijen. Otkrivanje takvog koda prouzrokovalo bi kopiranje proizvoda od strane konkurencije ili kopiranje nekih funkcionalnost softvera. Također omogućilo bi korisnicima da vide greške u kodu koje bi mogle narušiti sigurnost aplikacije. Vlasnički softver je softver koji je legalno vlasništvo neke osobe, izdavača, organizacije ili kompanije. To znači da oni imaju sva prava nad tim proizvodom što uključuje izdavanje proizvoda, razvoj proizvoda, prodaja proizvoda te unaprijeđenije proizvoda. Također imaju intelektualna prava nad izvornim kodom što znači da taj kod pripada njima i njegova „krađa“ može se smatrati kaznenim djelom. Za razliku od otvorenog softvera, samo vlasnici izvornog koda mogu vidjeti kod, praviti izmjene nad kodom, dodavati funkcionalnosti kodu te distribuirati kod.[18]

Veliki nedostatak softvera zatvorenog koda je limitiranost za korisnike. Vlasnik softvera može odlučiti i kojoj mjeri će njegov softver bili limitiran odnosno koliko će se on prilagođavati korisnicima te njihovim potrebama. Vlasnici mogu ograničiti korištenje softvera po želji koristeći uvjete korištenja (engl. Terms and Conditions) i pravila redistribucije softvera. Ograničeno može biti: Korištenje softvera, pregled i modifikacija koda, dijeljenje softvera kompatibilnost s drugim programima te korištenje softvera.[18]

Razvoj vlasničkog softvera popularno je kod velikih kompanija te kompanija koje razvijaju računalne igre. Privlači i programere koji žele komercijalizirati svoje proizvod. **Prednosti softvera zatvorenog koda** su sljedeće [19]:

- **Zarada** – velika prednost razvoja softvera je zarada. Gotovo je nemoguće zamisliti današnji svijet bez aplikacija koje nam olakšavaju svakodnevni život. Microsoft je poznat po tome što razvoja programe zatvorenog koda, suviše je reći da su jedni od najbogatijih kompanija u današnjem svijetu. Postoje besplatne alternative gotovo svim njihovim proizvodima, no jednostavno nisu dovoljno popularne za korištenje.

- **Stabilnost proizvoda** – softver otvorenog koda se konstantno razvija i evoluira, korisnici nemaju kontrolu nad načinom razvitka. Vlasnički softver predstavlja bolje rješenje za korisnike koje ne žele razvijati ili poboljšati programsko rješenje već žele imati sigurnost u korištenju.
- **Razvijen poslovni model** – razvoj vlasničkog softvera počinje s dobro definiranim poslovnim planom. Taj plan sadrži način razvoja softvera, buduća poboljšanja, izračune troškova i zarade i slično. Definiranje plana znatno se olakšava razvoj proizvoda jer programeri točno znaju što trebaju napraviti te koje su im dužnosti i vremenski rokovi.
- **Korisnička podrška** – velika prednost vlasničkog softvera je korisnička podrška. Ugovor obvezuje vlasnika softvera da korisniku pruži pomoć i podršku prilikom korištenja njegovog proizvoda. Također, u interesu vlasnika softvera je da pruži korisničku podršku jer samim time zadržava lojalnosti svojih korisnika i povećava svoju zaradu.
- **Lakoća korištenja –** cilj svake organizacije koja se bavi proizvodnjom vlasničkog softvera je olakšati njegovo korištenje. Korisnicima pružaju softver koji ima jednostavno i prepoznatljivo korisničko sučelje, navigaciju te funkcionalnosti kako bi unaprijedili njihovo korisničko iskustvo. Vlasnici se fokusiraju na grupu korisnika za koje je softver namijenjen za razliku od softvera otvorenog koda koji je namijenjen široj grupi korisnika.

Softver zatvorenog koda dolazi i s svojim **nedostacima**. One su sljedeće [19]:

- **Vijek trajanja softvera** – svaki proizvod ima svoj vijek trajanja pa tako i softver. Kada govorimo o softveru otvorenog koda tada smatramo da je njegov vijek trajanja dosta velik jer ga održava zajednica. S druge strane, softver zatvorenog koda ovisi o vlasniku ili organizaciji u čijem je vlasništvu. Vlasnik može u bilo kojem trenutku prekinuti potporu za proizvod te tako limitirati funkcionalnosti proizvoda ili ga kompletno ugasiti. To može predstavljati problem korisnicima koji temelje cijelo svoje poslovanje na softveru koji im pruža treća strana.
- **Glomazni softveri** – komercijalni proizvodi čest zauzimaju veliki prostor na računalo korisnika. Razlog tome je što oni instaliraju velik broj nepotrebnih programa na računalo. Ovo nije problem kod web aplikacija jer se one kompletno izvode na udaljenom poslužitelju.
- **Manja fleksibilnost** – vlasnički softver je gotov proizvod koji ima unaprijed zadanu namjenu. Korisnici taj softver ne mogu prilagođavati vlastitim

potrebama ili ga izmijeniti na bilo koji način osim onoga što je dozvolio vlasnik softvera.

- **Cijena** – u velikoj većini slučaja, vlasnički softver košta. Osim inicijalne cijene koštanja, vlasnik može mjesečno naplaćivati korištenje softvera ili čak naplatiti svako novo ažuriranje.
- **Licenciranje** – krajnji korisnik neće imati problema s licencama jer ono što plati i dobije. Velike kompanije, s druge strane, zahtijevaju veći broj kopija programa te licenciranje može biti problem. Obično se tada sklapaju posebni ugovori, gdje vlasnik nudi popust na količinu i definira posebne uvjete korištenja softvera.
- **Programska podrška** – kod aplikacija otvorenog koda, greška ili neki bug se ispravlja u kratkom roku jer je brzo uočen. S druge strane, programerima vlasničkog softvera može zatrebati i godina da poprave neku grešku. Korisnik mora čekati nova ažuriranja kako bi se neka greška ispravila ili dodala funkcionalnost koju on priželjkuje.
- **Sigurnost** – veliki problem kod aplikacija zatvorenog koda. Od programera se očekuje da naprave kvalitetno i sigurno rješenje u najkraćem mogućem roku. Pošto programeri bez ičije pomoći moraju pronalaziti sigurnosne rizike, to može predstavljati problem ako programerskom timu nedostaje ljudi ili jednostavno nema resursa za kvalitetan razvoj koda.
- **Privatnost** – Zbog nedostupnosti koda, korisnici ne znaju za što se njihovi podaci koriste te je su li sigurni. Tvrtka, podatke korisnika može koristit za ciljani marketing ili ih prodati trećoj strani.

3.3. Razvoj web aplikacija

Svaka web aplikacija, bilo da se radi o aplikaciji otvorenog koda ili aplikaciji zatvorenog koda, mora proći kroz ciklus razvoja web aplikacije. Sam proces razvoja aplikacije ovisi o poduzeću koje ju razvija. Postoje različite faze ili koraci razvoja aplikacije koji su već standardi prilikom razvoja aplikacije. Logično je da ne može biti razvijena bez osnovnih koraka koji uključuju dizajna, implementaciju te testiranje. Razvoj aplikacija otvorenog koda ima malo drugačiji pristup od razvoja onih zatvorenog koda što je opisano u zasebnom poglavlju.

Razvoj web aplikacije je proces programiranja na strani klijenta i programiranja na strani poslužitelja kako bi se napravila aplikacija koja je dostupna preko web preglednika. Programeri na početku kreiraju ideju o aplikaciji koja rješava određeni problem. Nakon razvitka ideje prelaze na dizajna aplikacije, izbor okvira i ostale tehnologije te implementaciju aplikacije. Nakon implementacije, aplikaciju je potrebno testirati kako bi se ona mogla javno objaviti na

poslužitelj i biti dostupna preko web preglednika. Web aplikacija pružaju velik broj sadržaja i usluga velikom broju korisnika. Uz samu važnost funkcionalnosti aplikacije, pouzdanost i kvaliteta izvršenja usluge postale su bitne. Kroz godine razvoja weba, zahtjevi i očekivanja od web aplikacija postali su sve veći i veći dok je proces razvoja, izvedbe i održavanja postao sve složeniji i teži za upravljati. Većina projekata nisu završena na vrijeme ili su prešla budžet upravo zbog lošeg upravljanja procesom razvoja. [20]

U današnjem svijetu, razvoj web aplikacija popularno je zanimanje. Tako, na primjer, postoji velim broj tvrtki koje se bave prodajom proizvoda ili usluga te žele biti prisutne na webu kako bi proširile svoje poslovanje na globalnu razinu i privukli veći broj korisnika. Te tvrtke žele jeftino i efikasno rješenje u obliku aplikacije kojoj mogu pristupiti korisnici različitih uređaja koji koriste različite operacijske sustave. Upravo zato se većina tvrtki odlučuje za korištenje web aplikacije u svojem poslovanju te je tržište za koje se razvijaju web aplikacije veliko. Razvoj internetskih tehnologija omogućio je da web aplikacije budu dobrih performansa, personalizirane i jednostavne za korištenje baš poput stolnih ili mobilnih aplikacija.

Razvoj web aplikacije dugotrajni je proces. Taj proces sadrži nekoliko koraka koji definiraju razvoj aplikacije. Vrijeme potrebno da se pojedini korak izvrši ovisi o veličini i složenosti problema kojeg aplikacija treba riješiti. Razvoj web aplikacije može se podijeliti na sljedeće korake: [20,21,22]

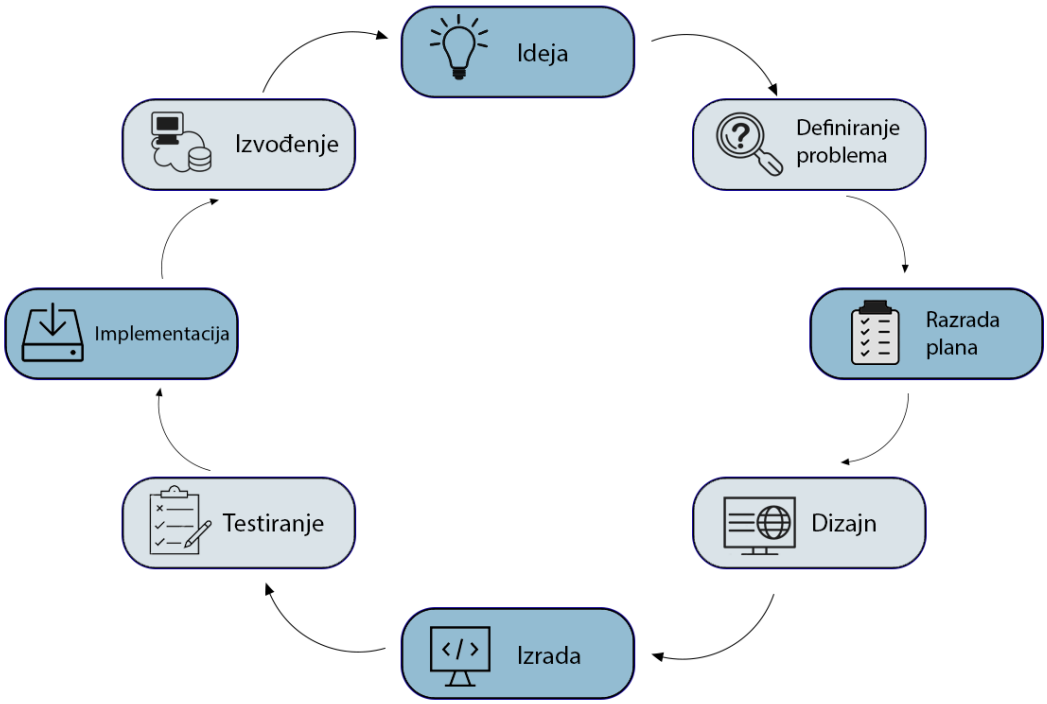
1. **Ideja** – polazna točka razvoja svake aplikacije. Ideja može biti nešto novo, nešto što će olakšati rješavanje nekog problema ili ideja može biti razvoj aplikacije koja je bolja, brža i jeftinija od aplikacije konkurencije. Najbolje aplikacije proizlaze iz traženja rješenja za probleme s kojima se ljudi susreću svakodnevno.
2. **Definiranje problema** – u ovoj fazi potrebno je dobro definirati problem kojeg aplikacija rješava te prikupiti sve relevantne informacije o problemu. Dobro definiran problem daje dobra put za pronalazak njegova rješenja. Potrebno je definirati **svrhu** i konačni **cilj** aplikacije. U ovom koraku potrebno je provesti **istraživanje tržišta.** Potrebno je identificirati i istražiti ciljanu publiku koja će koristiti aplikaciju. Ukoliko su zaposlenici tvrtke jedni od korisnika aplikacije, oni će pomagati prilikom izrade aplikacije. Potrebno je istražiti i konkurenciju, provjeriti da li postoji slična aplikacija na tržištu kako bi se osigurali da razvijena aplikacija bude bolja od one koja ja trenutno na tržištu.
3. **Razrada plana** – nakon definiranja problema, potrebno je razraditi plan i napraviti tijek rada (engl. workflow) za razvoj aplikacije. U ovoj fazi, programeri moraju definirati sljedeće:

- Što aplikacija morati raditi kako bi riješila definirani problem.
- Koje značajke i funkcionalnosti će aplikacija imati.
- Koji sve resursi su potrebni kako bi se aplikacija mogla razviti.

Aplikacija mora sadržavati jednu glavnu (engl. Core) funkcionalnost. Ta funkcionalnost rješavat će definiran problem te daje razlog korisnicima za korištenje aplikacije. Uvažiti se mora i set osnovnih funkcionalnosti koje pruža svaka aplikacija: manipuliranje korisničkih profila, upravljanje lozinkama, kontaktima, plaćanjima te upravljanje sadržajem aplikacije. Također, u ovom koraku, programeri moraju odabrat alate, platforme i ostale tehnologije koje će se koristiti za razvoj aplikacije. Alati moraju odgovarati tipu i opsegu projekta i njihova cijena korištenja mora ostati unutar dogovorenog budžeta. Definirane moraju biti i prekretnice (engl. Milestones) projekta kao i krajnji rok za završetak projekta.

4. **Dizajn web aplikacije** – u ovoj fazi izrađuje se skica (engl. Wireframe) i prototip izgleda korisničkog sučelja aplikacije kako bi se izgled rješenja prikazao ciljanim korisnicima. Programeri razrađuju dizajn koristeći interaktivne elemente kako bi omogućili ugodno korisničko iskustvo (engl. User Experience) za ciljanu publiku. Skica će biti prikazana potencijalnim korisnicima te će se zabilježiti njihova povratna informacija. Programeri će uvažiti povratne informacije te će prilagoditi dizajn prema tim informacijama. Postoji velik broj alata za izradu prototipa aplikacije koje programeri mogu koristiti, neki od poznatijih alata su: Sketch, InVision Studio, Adobe XD.
5. **Izrada web aplikacije** – uključuje izradu baze podataka, razvoj klijentske strane (engl. Frontend) i razvoj poslužiteljske strane (engl. Backend). Prilikom izrade baze podataka, programeri moraju odlučiti što će se spremati u tablice odnosne kakve će podatke baza bilježiti. Baza podataka čuva podatke korisnika te je bitno kvalitetno izraditi i definirati njene tablice, okidače, poglede te upite. Programeri zaduženi za razvoj klijentske strane uvažit će dogovoren dizajn aplikacije iz prijašnje strane te će napraviti korisničko sučelje aplikacije sukladno s time. Klijentsku stranu potrebno je povezati s bazom podataka preko poslužiteljske strane. Razvoj poslužiteljske strane temelji se na pozivanju pogleda i upita iz baze podatak, izrade načina obrade zahtjeva primljenih od korisnika te davanja odgovora korisniku. Programeri mogu kod aplikacije pisati od početka ili mogu koristiti razne okvire koji ima olakšavaju posao. Neki poznatiji okviri su: AngularJS, React JS, Django i slično. Za izradu rješenja koriste se tehnologije dogovorene u fazi planiranja.

6. **Testiranje aplikacije** – aplikacija se može testirati i u fazi njene izrade. Ovaj korak odnosi se na testiranje kompletne aplikacije koja će se lansirati na tržište ukoliko ispunjava zahtjeve testiranja. Testiranjem se osigurava kvaliteta web aplikacije. Bilo automatsko ili ručno, testiranje je konstantan proces koji se javlja u životnom ciklusu razvoja softvera. Testeri aplikacije rade testove za koji osiguravaju visoku razinu funkcionalnosti, upotrebljivosti, kompatibilnosti, sigurnosti i performansi aplikacije. Provode se rigorozni testovi kako bi se eliminirale greške i osiguralo da aplikacija radi besprijekorno. Testiranjem se također mogu utvrditi potrebna poboljšanja i nadogradnje koje se mogu implementirati u budućnosti.
7. **Implementacija aplikacije u poslovanje** – nakon što aplikacije prođe fazu testiranja, može se implementirati u poslovanje tvrtke ili se može lansirati klijentima. Potrebno je pružiti trening zaposlenicima i korisnicima kako bi mogli efektivno koristiti aplikaciju. Ukoliko se radi o razvoju vlastite aplikacije, potrebno je odlučiti kada će se ona plasirati na tržište.
8. **Izvođenje i održavanje aplikacije** – web aplikacija se mora izvoditi na poslužitelju. Potrebno je aplikaciju instalirati na odabrani poslužitelj koji je dovoljno dobar da se ona može neometano izvoditi. Također poslužitelj mora imati instaliran sav softver za podršku izvođenja aplikacije. Aplikacije mora biti održavana. Proizvođač aplikacije obvezuje se da će raditi potrebne testove kvalitete kako bi se uvjerio da je aplikacija na istoj razine performansi i sigurnosti na kojoj je bila kad je lansirana. Također obvezuje se da će ispravljati greške, dodavati nove funkcionalnosti na zahtjev korisnika te davati ažuriranja vezana za aplikaciju kao i tehničku podršku korisnicima.



Slika 7: Životni ciklus web aplikacije

Donja tablica sadrži kraći životni ciklus jedne aplikacije te navodi izlaze iz svake faze životnog ciklusa.

Tablica 2: Izlazi životnog ciklusa aplikacija (Prema : [23])

Faza životnog ciklusa	Izlaz
Konceptualizacija i dizajn	Funkcionalne i tehničke specifikacije + prijedlozi dizajna i specifikacija
Izrada aplikacije	Dokumentiran izvorni kod na repozitoriju
Implementacija	Dokumentacija aplikacije
Produkcija	Upute za izvođenje

Web aplikaciju može razviti jedna osoba ili u njenom razvoju može sudjelovati cijeli tim osoba koji rade za neku organizaciju. Ukoliko se radi o jednoj osobi ,tada ona ima sva zaduženja koja bi inače bila raspoređena u timu. U razvoju optimalnog projekta sudjeluju sljedeće osobe [23]:

- Projektni menadžer – upravlja projektnom tijekom njegova životnog ciklusa, zadužen za raspodjelu odgovornosti i komunikaciju između tehničkog i ne-tehničkog osoblja.
- Vlasnik projekta – osoba koja je zadužena da projekt zadovolji standarde organizacije te da bude korisnicima dostavljena u dogovorenom roku
- Programer koji razvija poslužiteljsku stranu (engl. Back-end Developer)
- Programer za klijentsku stranu (engl. Front-end Developer)
- Tester za osiguranje kvalitete
- Sistemski administrator

Donja tablica prikazuje sudjelovanje pojedinih osoba u različitim faza životnog ciklusa jedne aplikacije. Vidljivo je da većina osoba sudjeluje u svim fazama osim sistemskog administratora koji je zadužen za nadgledanje aplikacije tijekom njegov stvarnog izvođenja.

Tablica 3: Sudjelovanje osoba u životnom ciklusu aplikacije (Prema: [23])

Faze životnog ciklusa / osobe	**Projektni menadžer**	**Front-end programer**	**Back-end programer**	**QA tester**	**Sistemski administrator**
Tehnička i funkcionalna konceptualizacija	+	+	+	+	-
Izrada aplikacije	+	+	+	+	-
Testiranje aplikacije	+	+	+	+	+
Izvođenje aplikacije u određenoj okolini	+	+	+	+	+

3.3.1. Tehnologije za razvoj Java web aplikacija

Java ima jaku podršku za razvoj web aplikacija te je iz tog razloga često korištena za programiranje na strani poslužitelja. Tehnologije za razvoj Java web aplikacija dio su Java EE (Enterprise Edition) platforme uz dodatak mnogih Java SE (Standard Edition) klasa i paketa. Kako bi se te tehnologije mogle izvoditi na poslužitelju, poslužitelj mora imati instalirani kontejner ili web poslužitelj koji prepoznaje i pokreće klase koje su kreirane. [24,25]

Java web aplikacije ne izvode se direktno na poslužitelju već se izvode unutar web kontejnera na tom poslužitelju. Kontejner pruža okolinu za izvođenje (engl. Runtime Environment) za Java web aplikacije. Kontejner bi se mogao usporediti s JVM-om (Java Virtual Machine). JVM pruža okolinu za izvođenje lokalnim java aplikacijama, kontejner se također izvodi unutar JVM-a. Općenito, java definira dva kontejnera: Web kontejner i Java EE kontejner. Tipični web kontejneri u Java svijetu su Tomcat i Jetty. Web kontejner daje potporu za izvršenje Java servleta i JSP-a (JavaServer Page). Java EE kontejner pruža dodatne funkcionalnosti poput distribucije učitavanja poslužitelj. Većina modernih Java web okvira bazirana je na servletima. Popularni Java web okviri su GWT, JavaServer Faces, Struts i Spring Okvir. Ti okviri, za izvršavanje svojih funkcionalnosti, minimalno zahtijevaju postojanje web kontejnera.[24]

Poznato je da je Java web aplikacija kolekcija dinamičnih resursa poput Servleta, JSP-a, Java klasa i jar-ova te statičkih resursa poput HTML stranica i slika. Java web aplikacija upakirava se kao WAR (Web Archive) datoteka te se kao takva postavlja na poslužitelj. WAR datoteka je zip datoteka koja sadrži kompletan sadržaj neke web aplikacije. [24]

Kao što je rečeno, postoje mnoge Java tehnologije za razvoj Java web aplikacije. Dolje su navedene samo neke koje su najpoznatije. Također, ne moraju se koristiti sve tehnologije za razvoj pojedine aplikacije, ponekad je dovoljno koristiti samo JSP tehnologiju. Tehnologije se mogu i kombinirati zbog svoje kompatibilnosti, pa je tako moguće koristiti dvije ili više tehnologija zajedno kako bi se napravila željena aplikacija. Odabir tehnologije ovisi o namijeni aplikacije. Poanta tehnologija je da se olakša razvoj pojedine aplikacije, no ukoliko se ne koriste na ispravan način, mogu značajno otežati razvoj aplikacije. Neke poznate Java web tehnologije su sljedeće [25]:

- **Java Servlet API** – dopušta definiranje HTTP-specifičnih klasa. Sve klase nalaze se unutar paketa *javax.servlet*. Servlet klasa proširuje sposobnosti poslužitelja na kojim se nalazi aplikacija i koji komuniciraju na način zahtjeva-odgovora. Na primjer, servlet se može koristiti kako bi se dobio tekstualni unos iz online forme i zatim taj unos ispisao natrag na ekran koristeći HTML stranicu i format. Također, može se koristiti drugačiji servlet kako bi se taj zapis zapisao u datoteku ili bazu podataka. Servlet se izvodi na strani poslužitelja bez grafičkog sučelja ili HTML korisničkog sučelja. Java servlet proširenja omogućuju razvoj mnogih web aplikacija.
- **JavaServer Pages** – tehnologija koja pruža jednostavan i brz način stvaranja dinamičnog web sadržaja. JSP omogućuje brz razvoj web aplikacija koe su neovisne o poslužitelju ili platformi na kojoj se izvode. JSP omogućuje

uključivanje dijelova koda direktno u HTML dokument te tima omogućuje dinamičnu promjenu sadržaja HTML elemenata. Ukratko, JSP stranica je tekstualni dokument koji se sadrži od dva tipa teksta:

- Statičnih podataka koji se prikazuju u obliku HTMLa, WMLa (Wireless Markup Language) ili XMLa
- JSP elemenata koji definiraju dinamične dijelove sadržaja

Klase koje određuju JSP tehnologiju uključuju se u dokument preko *javax.servlet.js* paketa te *javax.el* paketa.

- **JavaServer Pages standardna biblioteka oznaka (JSTL)** – učahuruje temeljne funkcionalnosti koje su česte za mnoge aplikacije koje koriste JSP tehnologiju. Umjesto korištenja različitih oznaka od mnogobrojnih „dobavljača“ , koristi se jedan standardizirani set oznaka. Ta standardizacija omogućuje postavljanje vlastite aplikacije na JSP kontejner koji podržava JSTL te time daje bolju optimizaciju uključenih oznaka. Osnovni paket za uključivanje JSTL klasa je *javax.servlet.jsp.jstl* paket.
- **JavaServer Faces** – okvir korisničkog sučelja za izgradnju web aplikacija. Glavne komponente JSF tehnologije uključuju okvir za grafičko sučelje, fleksibilni model za renderiranje komponenata u različite „markup“ jezike i tehnologije te *RenderKit* za generiranje HTML stranica.
- **Java Message Service API** – način komuniciranje između komponenti softvera ili aplikacije. Sistem poruka može razmjenjivati poruke s različitim klijentima. Svaki klijent spaja se na agenta za razmjenu poruka koji omogućuje kreiranje, slanje, primanje te čitanje poruka. Za primjer sustava poruka može se uzeti neka aplikacija za autoindustriju. Aplikacija bi pratila koliko proizvoda ima trenutno u skladištu te kada bi skladište bilo gotovo prazno poslala bi poruku o narudžbi tvornici koja proizvodi aute. Tvornica za proizvodnju tada bi dalje poslala poruku tvornici za dijelove gdje bi naručila pojedine dijelove za auto. Tvornica za aute nakon primitka dijelova ažurira svoje skladište i proizvodi auto te šalje poruku skladištu s autima i tako u krug.
- **JavaMail API** - web aplikacije mogu koristiti JavaMail API za slanje poruka putem elektroničke pošte. Taj API sastoji se od dva dijela: sučelja na razini aplikacije koje komponente aplikacije koriste za slanje e-mailova te sučelje pružatelja usluge koji sadrži potrebne email protokole za slanje pošte.
- **Java API za XML procesiranje (JAXP)** – podržava procesiranje XML dokumenata koristeći DOM (Document Object Model), SAX (Simple API for

XML) i XSLT (Extensible Stylesheet Language Transformations). JAXP omogućuje aplikacijama parsiranje i transformaciju XML dokumenata neovisno o implementaciji XML načina procesiranja.

- **JDBC API** – omogućuje pozivanje SQL naredbi iz baze podataka u obliku metoda Java programskog jezika. Sastoji se od dva djela: sučelja na razini aplikacije koje mogu koristiti komponente aplikacije za pristup bazi podataka te sučelje pružatelja usluge koji dodaje JDBC driver Java EE platformi.
- **Java Persistence API** – Java tehnologija koja daje standardno rješenje za perzistentnost. Perzistentnost koristi objektno-relacijsko mapiranje kao most između objektno orijentiranog modela u relacijske baze podataka. Sastoji se od tri područja:
    - Java Persistence API
    - Jezik upita (engl. query language)
    - Objektno-relacijsko mapiranje
- **JNDI (Java Naming and Directory Interface)** – pruža funkcionalnosti za imenovanje i direktorije omogućujući aplikacijama da pristupaju različitim uslugama za imenovanje i direktorije. Daje aplikacijama metode za standardne operacije poput povezivanja atributa s objektima, pretragu objekata preko atributa i slično. Koristeći JNDI, web aplikacija može spremiti i primiti bilo koji tip imenovanog Java objekta što dopušta aplikacijama da surađuju s mnogim drugim aplikacijama i sistemima.

3.4. Sličnosti i razlike u razvoju aplikacija otvorenog i zatvorenog koda

Glavna razlika između aplikacija otvorenog koda i aplikacija zatvorenog koda je dostupnost izvornog koda. Samim time, razvoj takvih aplikacija znatno se razlikuje. Izvorni koda aplikacija zatvorenog koda mora se držati tajnom, tako se koriste različiti privatni repozitoriji kako bi se osigurao razvoj aplikacija u tajnosti. S druge strane, aplikacije otvorenog koda razvija zajednica. Izvorni kod dostupan je na pregled svima te svatko može napraviti doprinos razvitku aplikacije. Glavna značajka razvoja aplikacija otvorenog koda je paralelizam u razvoju. Faze razvoja aplikacije (planiranje, dizajn, implementacija,…) ne izvode se slijedno, već programeri vrše paralelan razvoj. To se većinom odnosi na faze dizajna, kodiranja i testiranja koje se u nekoj mjeri mogu izvoditi paralelno. Korisnici aplikacije imaju veliku ulogu u testiranju aplikacije gdje mogu prijaviti pronađenu grešku programerima ili ju mogu sami ispraviti.

Poznato je da je većina aplikacija otvorenog koda besplatna. Glavni problem s besplatnim softverom je nedostatak profesionalne korisničke podrške. Prilikom razmatranja o korištenju otvorenog softvera, korisnici su zabrinuti za sljedeće [26]:

- **Održavanje konzistentne softverske arhitekture** – korisnici žele biti uvjereni da se arhitektura aplikacije neće mijenjati na način da će previše odskakati od originala na koji su pristali kada su se odlučili koristiti ovaj softver.
- **Tehnička potpora i koordinacija prilikom instalacije** – prilikom instalacija aplikacije ili nekog drugog softvera, korisnici žele znati da li je softver automatiziran ili je potreba integracija s nekom drugom komponentom.
- **Upravljanje ažuriranjima i porast kompleksnosti koda** – korisnike također zabrinjava budućnost softvera. Žele znati hoće veličina budućih ažuriranja utjecati na kompleksnost koda. Upravljanje kompleksnošću koda bitna je stvar kod razvoja otvorenih aplikacija zbog paralelnog razvoja.

Aplikacije otvorenog koda razvija zajednica. Samim time, nemoguće je provesti isti način razvoja aplikacije kao kod razvoja aplikacija zatvorenog koda. Prva uočljiva razlika je **procesni model** koji se koristi prilikom razvoja. Aplikacije zatvorenog koda prate iterativni model razvoja. Kod takvog modela, softver prilikom razvoja prolazi kroz sve faze: planiranje, dizajna, implementacija, testiranje i slično. Prolazak je rekurzivan što znači da se moguće vratiti na prijašnju fazu. Aplikacije otvorenog koda koriste evolucijski model razvoja gdje softver nikad ne dostigne svoje završno stanje već se konstantno nadograđuje. Takav model više pogoduje paralelnom razvoju aplikacije što odgovara aplikacijama otvorenog koda. Također, aplikacije otvorenog koda nemaju jasno zadane ciljeve. Kod aplikacija zatvorenog koda točno se zna koji se ciljevi moraju ostvariti na kojem „checkpointu“. Sljedeća razlika u razvoju je **definiranje i specificiranje zahtjeva**. Time započinje razvoj aplikacija zatvorenog koda, no unatoč tome, zahtjevi ponekad znaju biti nejasni i loše definirani jer klijent za kojeg se razvija aplikacije nije siguran što sve želi od aplikacije. S druge strane, zahtjevi kod aplikacija otvorenog koda su jasniji jer su motivirani osobnim željama i potrebama pojedinog korisnika. Također, kod aplikacija zatvorenog koda ponekad nije moguće implementirati sve zahtjeve zbog limitiranog budžeta i vremena, dok s druge strane korisnici otvorenih aplikacija mogu sami definirati vlastite zahtjeve i sami ih implementirati. Arhitekti i projektni menadžeri odlučuju koje će zahtjeve uvažiti kod aplikacija zatvorenog koda. Kod aplikacija otvorenog koda, u originalnu verziju aplikacije, zahtjeve odobravaju glavni članovi koji rade na otvorenom projektu. Aplikacije zatvorenog kod sadrže dobro strukturiranu i jasnu **dokumentaciju**, s druge strane, otvorene aplikacije mogu, ali e moraju sadržavati dokumentaciju. Nedostatak dokumentacije može predstavljati problem prilikom korištenja aplikacije. U razvoju zatvorenih

aplikacija, arhitekt i projektni menadžer troše puno vremena na **analizu i dizajn** aplikacije, dok kod otvorenih aplikacija, dizajn se često radi kad i implementacija pošto svaki korisnik može personalizirati aplikaciju prema vlastitim potrebama. Veliki problem aplikacija otvorenog koda je održavanje konzistentne **arhitekture softvera**. Na otvorenim projektima radi velik broj ljudi i održavanje kvalitetne arhitekture je vrlo teško. To nije problem sa zatvorenim aplikacijama jer se održavanje konzistentne arhitekture forsira na svim fazama razvoja softvera. Sljedeća razlika leži u **implementaciji** rješenja problema za pojedine zahtjeve. Kod zatvorenih aplikacija postoji pravilo : jedna implementacija rješava problem jednog zahtjeva. S druge strane, kod aplikacija otvorenog koda, postoji više rješenja za pojedini problem. To ponekad predstavlja problema (aplikacija se grana na više verzija), a ponekad i dobro rješenje jer se tada može odabrani ona implementacija koja je optimizirana i bez grešaka u radu. Velika prednost otvorenog razvoja aplikacija je brzina implementacije. Zatvoreni projekti rijetko kad mogu dostići otvorene projekte kada se radi o brzini implementacije rješenja. Otvorenost **izvornog koda** daje mogućnosti korisnicima da pokrenu vlastiti razvoj (točnije nadogradnju) aplikacije te ju prilagode vlastitim potrebama. Takva sloboda razvoja ne postoji kod zatvorenih projekata. **Testiranje** aplikacije je također jedna od velikih prednosti otvorenih projekata. Korisnici često imaju ulogu testera te prijavljuju greške koje nađu u aplikaciji ili ih sami ispravljaju. S druge strane, zatvorene aplikacije oslanjaju se na vlastite programere i automatizirane sustave za pronalazak greški kako bi osigurali da je aplikacija optimizirana i bez grešaka. Rijetko kad se mogu pokriti svi mogući slučajevi izvođenja aplikacije korištenjem automatskog testiranja. Aplikacije otvorenog koda **isporučuju** se često jer ne ovise o faktorima koje sprječavaju aplikacije zatvorenog koda da budu puštene u opticaj. Zatvoreni projekti počinju se razvijati na zahtjev tržišta ili neke organizacije te moraju imati dobro definirane ugovore, uvjete korištenja i licence kako bi podlijegali zakonima tržišta. **Održavanje** aplikacija otvorenog koda lakše je zbog zajednice koja neku aplikaciju održava na životu, s druge strane, zatvorene aplikacije zahtijevaju financijsku potporu te potporu korisnika kako bi preživjele i kako se ne bi zauvijek ugasile. Jedno od glavnih pitanja koje se postavlja aplikacijama je: da li su one **sigurne**? Mnogi misle da su aplikacije zatvorenog koda sigurnije upravo zbog toga što je njihov izvorni kod skriven. To je često kriva informacija. Otvorenost koda omogućuje razvoj sigurne aplikacije jer se moguće ranjivosti aplikacije prepoznaju brzo te se brzo i ispravljaju. Aplikacije otvorenog koda razvijaju se brže, često su kvalitetnije od aplikacija zatvorenog koda te su vrlo vjerojatno besplatne. No to ne mora uvijek biti tako. **Licence** određuju koliko se aplikacije mogu modificirati od originala i kako se one mogu distribuirati, također one stavljaju i cijenu na neku aplikaciju.[26]

3.5. Principi razvoja web aplikacija

Postoje određeni principi koje je potrebno zadovoljiti tijekom razvoja web aplikacije kako si se stvorila kvalitetna web aplikacija. Ti principi zapravo su vodilje koje odvajaju kvalitetno aplikaciju od ne kvalitetne aplikacije. U ovom poglavlju napravljene su dvije podijele principa. Prva podjela odnosi se na osnovne principe koje mora zadovoljiti svaka kvalitetna web aplikacija. Radi se o principima o kojima se razmišlja prilikom planiranja aplikacija, dizajna aplikacije te izgradnje aplikacije. Druga podjela odnosi se na više „tehničke“ principe koji se javljaju prilikom izgradnje aplikacije.

Tehnologija konstantno napreduje, no principi ostaju isti. Potreba za aplikacijom koja je sigurna i prilagođena korisnika uvijek je prisutna. Dolje navedeni principi odnose se na razvoj modernih aplikacija, no zadovoljavanje tih principa uvijek je bio cilj razvoja svake aplikacije. Osnovni principi za razvoj web aplikacije su [27]:

- **Planirati razvoj aplikaciji i očekivati da će on potrajati** – proces kreiranja web aplikacije može biti dugotrajan ukoliko se radi o razvoju kompleksnije aplikacije koja ima mnoge funkcionalnosti.
- **Stvoriti konzistentnu aplikaciju** – kvalitetna aplikacija zadovoljava potrebe korisnika u sadašnjosti, ali i u budućnosti. Korisnici znaju kako se aplikacija ponaša i garantira im se da će njeno ponašanje ostati isto ili slično kako se ne bi morali ponovo educirati za njeno korištenje
- **Definirati skupinu korisnika** – prilikom razvoja aplikacije potrebno je identificirati ciljanu skupinu korisnika. Treba utvrditi tko će koristiti aplikaciju te ju takvom korisniku prilagoditi. Može se raditi o različitim tipovima korisnika. Ukoliko se korisnik neće snaći u aplikaciji, on ju neće ni koristiti.
- **Sigurnost mora biti dobro implementirana** - velik broj programera nije dobro upoznat s pojmom sigurnosti ili opće zanemaruju sigurnost kao takvu jer ne pruža nikakvu dodatnu funkcionalnost aplikaciji. Sigurnost je nešto na što programeri moraju obratiti pozornosti od početka razvoja aplikacija. Ona mora biti prisutna u svim aspektima razvoja i izvršavanja aplikacije.
- **Korištenje alata** – postoji velik broj razvojnih alata koje programeri mogu koristiti. Imaju pristup bibliotekama, okvirima i gotovim shemama. Mogu koristiti javne repozitorije, razvojna okruženja te poslužiteljska okruženja.
- **Aplikacija mora biti optimizirana i brza** – optimizacija aplikacije bitan je faktor koji se često zna ignorirati tijekom razvoja aplikacije. Programeri aplikaciju često optimiziraju po potrebi nakon njenog prvotnog lansiranja. Loša optimizacija

znači i manja brzina izvođenja aplikacije. Ukoliko je stranica sporija, korisnici neće previše čekati već će napustiti web aplikaciju i prijeći na konkurentnu aplikaciju.

- **Arhitektura aplikacija mora biti jasna i konzistentna** – arhitektura aplikacije je nešto što korisnici ne vide, a uvelike utječe na kvalitetu aplikacije. Potrebno je stvoriti konzistentnu arhitektura koja se provlači kroz sve faze životnog ciklusa razvoja aplikacije. Arhitektura nudi sigurnost, održavanje , fleksibilnosti i skalabilnost aplikacije te je neophodna za izgradnju aplikacije koja će trajati dugo. Arhitektura aplikacije daje modularnost kako bi se lako nadograđivala u budućnosti te koristila zajedno s drugim programima i aplikacijama.
- **Aplikacija se treba prilagoditi novim tehnologijama** – aplikacija ne bi trebala ovisiti o zastarjelim tehnologijama. Treba se prilagođavati novim uređajima, operacijskim sustavima, protokolima, okvirima, programima i slično. Razvoj aplikacije treba se temeljiti na novim i dokazanim tehnologijama kako bi aplikacija imala dug životni vijek i stekla povjerenje korisnika.

Razvoj web aplikacije popraćen je različitim principima koji pojednostavljuju proces izrade. Neki od najvažnijih principa su slijedeći [28]:

- **Korištenje zajedničkog repozitorija** – korištenje jedne baze koda (engl. Codebase) znatno olakšava proces razvoja aplikacije. Omogućuje programerima da zajednički razvijaju jednu aplikaciju korištenjem vlastitih računala. Najpoznatiji repozitorij za pohranu je Github. Kada neki programer razvije neku funkcionalnost potrebno je napraviti „push“ na Git. Ta funkcionalnost tada može odmah biti testirana od strane testnog tima. Nakon odobrenje, funkcionalnost se može spojiti s glavnim kodom odnosno glavnom granom koda.
- **Eksplicitno deklariranje i izoliranje ovisnost uključenih u aplikaciju** – postoji zlatno pravilo koje govori da je na početku razvoja aplikacije potrebno napravili listu svih vanjskih biblioteka i usluga koji će se koristiti. Te ovisnost trebale bi biti odvojene od koda i treba ih biti moguće uključiti i isključiti po potrebi. One trebaju imati minimalni utjecaj na kod. Upravljanje ovisnostima važno je jer osigurava sigurnost i stabilnost same aplikacije.
- **Odvajanje konfiguracije od aplikacijskog koda** – konfiguracija sadrži postavke potrebne za inicijalno pokretanja aplikacije te njezino izvođenje. Postavke konfiguracije često se mogu mijenjati te ih je potrebno pohraniti u zasebnu datoteku u formatu koja je čitljiva aplikaciji (na primjer: csv, xml,

json,..). konfiguraciju ne bi trebalo dijeliti s javnošću jer može sadržavati osjetljive podatke (lozinke, ključeve i slično). Konfiguracija može sadržavati postavke za konekciju s bazom podataka, korisničke podatke za spajanje s nekom uključenom uslugom, API ključeve i ostale podatke o kojima ovisi pokretanje i izvođenje aplikacije.

- **Korištene usluge moraju biti držane odvojeno** – aplikacije može koristit različite usluge treće strane, API-e ili komponente baze podataka. Te usluge moraju biti tretirane kao pridodani resursi.
- **Podatke pohraniti izvan web aplikacije** – podatke kojima barata aplikacija najbolje je pohraniti u neku relacijsku bazu podataka. Moguće je koristi vlastiti poslužitelj baze podataka ili iznajmiti poslužitelj baze od neke treće strane poput Googla ili Amazona.
- **Razdvajanje procesa aplikacije** – aplikacija će tijekom svog izvođenja koristiti veći broj procesa. Te procese potrebno je držati razdvojenima. Ukoliko neki proces troši previše resursa potrebno ga je razdvojiti na manje procese. Ti procesi neće međusobno dijeliti podatke i aplikacija će biti jednostavnija i stabilnija za korištenje. Time aplikacija postaje skalabilnija.

3.6. Izazovi u razvoju web aplikacija

Razviti aplikaciju visokih performansi koja ima korisnički prihvaćeno sučelje nije lagano. Razvoja kvalitetne web aplikacije zahtjeva vrijeme, tehničke i ljudske resurse te prevladavanje prepreka koje se nađu u procesu razvoja. Tijekom razvoja aplikacije, javljaju se mnogi izazovi. Ako se tim izazovima ne pristupa pravilno, oni mogu predstavljati probleme koji bi mogli smanjiti kvalitetu aplikacije. Izazovi se već pojavljuju u procesu planiranja. **Jasno definiranje ciljeva** predstavlja jedan od izazova. Potrebno je jasno definirati ciljeve i zahtjeve počevši od vizije. Također, ključni zahtjevi aplikacije na razini poduzeća različiti su od zahtjeva za neku igru ili uobičajenu aplikaciju. Prilikom planiranja, jasno moraju biti definirane sljedeće stvari: ciljana skupina korisnika, korisničko iskustvo, obavezne značajke dizajna i tehnički zahtjevi. U današnjem svijetu, postoji mnogo pouzdane tehnologije za razvoj web aplikacija. Tvrtke često imaju problem prilikom odabira pravilne tehnologije za razvoj određene aplikacije. **Izbor ispravne tehnologije** za razvoj često predstavlja izazov u svijetu web aplikacija. Tehnologija za razvoj web aplikacija predstavlja kombinaciju programskih jezika, okvira, poslužitelja, softvera i ostalih alata koje koriste programeri. Izbor tehnologije trebao bi ovisiti o kompleksnosti problema koji se rješava. Na primjer, nije potrebno koristi kompliciranе tehnologije ukoliko se razvija jednostavna aplikacija, isto tako nije potrebno koristiti tehnologiju

koja optimizira skaliranje ukoliko se radi o konzistentnom broju korisnika. Također, poželjno je koristiti popularnu tehnologiju jer tada će organizacija imati veći broj iskusni programera na odabir. Isto tako, korištena tehnologija treba biti dobro dokumentirana kako bi se neki problem što brže riješio. Slijedeći izazovi javljaju se u procesu implementacije rješenja, točnije u procesu dizajna, i procesu izrade funkcionalnosti. Veliki izazov predstavlja dobar „dizajn“ **korisničkog iskustva**. Korisničko iskustvo obuhvaća reakciju, percepciju i osjećaje koje korisnika izražava kad koristi aplikaciju. To je osjećaj lakoće i jednostavnosti korištenja kada se radi o dobrom dizajnu. Također, to može biti osjećaj frustracije ako korisnik vrši interakciju s lošim dizajnom iskustva. Prilikom dizajna sučelja, programeri moraju odlučiti kakav utisak žele da sučelje ostavi na korisnika. Kvalitetan dizajn korisničkog iskustva daje kvalitetniju aplikaciju koju će korisnici koristiti iz dana u dan te će ostvarivati ciljeve poduzeća. **Dizajn korisničkog sučelja** također predstavlja jedan od izazova. Sučelje sadrži sve vizualne elemente s kojima korisnik vrši interakciju u web aplikaciji. To je sve što korisnik vidi i sve što klikne kako bi se kretao kroz korisničko iskustvo. Dobra dizajna sučelja privlači korisnike. Cilj svakog dizajna jer napraviti korisničko iskustvo jednostavnim, pristupačnim i iskoristivim. To znači da će se koristiti samo ciljani, svrhoviti sadržaj s jasnim mogućnostima upravljanja sadržajem te dostupnosti informacija u svakom koraku korištenja sučelja. Programeri se susreću s izazovima kreiranja: jasne navigacije, jednostavne vizualizacije te tipografije koju je jednostavno pročitati. Tijekom faze razvoja funkcionalnosti pred programere se stavlja izazov **kreiranja aplikacija dobrih performansi i optimalne brzine**. Poznato je da korisnici ne vole aplikacije koje se učitavaju sporo. Spora aplikacija može značiti propast same aplikacije jer će je korisnici napustiti i neće se ponovo vraćati. Ukoliko se unaprijed zna da će aplikacija prikazivati mnogo sadržaja (na primjer slika ili videozapisa) tada treba potaknuti programere da izgrade optimiziraniju aplikaciju kako bi se osigurale bolje performanse. Aplikacija tada mora biti skalabilna kako bi se mogla prilagoditi rastu broja korisnika. Često je teško testirati aplikaciju koja mora posluživati velik broj korisnika jer je nemoguće predvidjeti koliko će zapravo korisnika koristiti tu aplikaciju. Ukoliko aplikacija nije skalabilna može doći do znatnog smanjenja performansi ili ček padanja poslužitelja aplikacije. Tako je **skalabilnost aplikacije** jedan od izazova prilikom programiranja. Jedan od najvažnijih izazova je **osigurati aplikaciju od sigurnosnih prijetnji**. Potrebno je mnogo toga uzeti u obzir kako bi aplikacija i njeni korisnici bili sigurni. Prvi korak je dobar odabir aplikacijske arhitekture. Treba osigurati da arhitektura ima dovoljno sigurnosnih usluga i opcija kako bi programeri mogli pravilno implementirati sigurnosne mjere u aplikaciju. Postoje SSL certifikati koji su globalni standard za sigurnosti i omogućuju kriptiranu komunikaciju između web preglednika i poslužitelja. Moguće ih je integrirati u aplikaciju kako bi poboljšali njenu sigurnost i osigurali da aplikacija

ne bude označena kao ne sigurna od strane web preglednika. Postoji velik broj tehnologija za sigurnost koje programeri moraju razmotriti kako bi sigurnosne prijetnje sveli na minimum.[29]

Provedeno je i istraživanje od UI Bakery Team [30] autora. Radi se o istraživanju najčešćih izazova koji se pojavljuju tijekom faza životnog ciklusa aplikacije. Izazovi se javljaju u fazi dizajna, izrade aplikacije ali i tijekom pružanja potpore aplikaciji. Najčešći izazovi tijekom faze dizajna aplikacije su [30]:

- **Nedovoljno vremena za dizajn** – najveći izazov ili problem tijekom faze dizajna. Čak se 72% testiranih formi izjasnilo da nemaju dovoljno vremena za razvitak kvalitetnog dizajna sučelja. Tvrtke većinom „recikliraju“ dizajn ili postojećih projekata uz manje izmjene.
- **Pojavljivanje grešaka tijekom kontrole dizajna** – kvalitetna implementacija dizajna jedan je od većih izazova. Teško je testirati sve aspekte dizajna pogotovo kada se radi o aplikaciji koja se treba izvoditi na različitim uređajima s različitim dimenzijama ekrana i koji koriste različite preglednike.
- **Nedostatak UX dizajnera** – IT poduzeća rijetko kad traže specijaliziranog UX dizajnera. Dizajn j većinom stvar dogovora te ga radi osoba koja je zadužena za front-end. Ta osoba većinom koristi provjerena, gotova rješenja.
- **Manjak interesa prema dizajnu** – prilikom razvoja aplikacije, glavni fokus je na razvoju funkcionalnosti, dok dizajn sučelja pada u drugi plan.
- **Nedostatak UI dizajnera** – isto kao i s UX dizajnerom, tvrtke rijetko kad odvajaju svoje resurse na zaposlenje UI dizajnera.

Istraživanje sadrži i rezultate za izazove koji se javljaju tijekom faze razvoja funkcionalnosti aplikacije. Ti izazovi su slijedeći [30]:

- **Nedostatak vremena za validaciju ideja** – na ovaj problem nailazi čak 86% ispitanih poduzeća. Kreirati profitabilnu aplikaciju iz prvog pokušaja gotovo je nemoguće. Potrebno je potrošiti dovoljno vremena za validaciju različitih ideja prije izgradnje aplikacije. Većina poduzeća ne može osigurati dovoljno vremena da kvalitetno razradi funkcionalnosti softvera i utvrdi na koji način će one pomoći korisnicima te koliko će korisnici morati platiti za to.
- **Problemi kod skaliranja rješenja** – poduzeća teško predviđaju koliko će korisnika koristiti aplikaciju te koliko će se ona nadograđivati i proširivati u budućnosti. Upravo zbog toga, rijetko se odlučuju na kreiranje skalabilnog rješenja od početka razvoja.
- **Dugo vrijeme plasiranja na tržište** – kaka tržište zahtjeva neku aplikaciju, ponekad prođe puno vremena prije nego što ona uopće bude lansirana.

Poduzeća često čekaju pre dugo da lansiraju određenu aplikaciju zbog manjka vremena ili resursa za razvoj.

- **Nedovoljno resursa za razvoj** – problem s kojim se poduzeća često susreću. Nedostatak radne snage ili tehničkih resursa onemogućuje razvoj željene aplikacije.
- **Problemi u komunikaciju između timova** – ne razumijevanje zahtijeva korisnika ili nesporazum između marketinških timova, timova za razvoj klijentske strane i poslužiteljske strane često predstavlja velik problem i dovodi do aplikacije koja je puno grešaka i netočnih rješenja.
- **Visoki troškovi razvoja** – poduzeća često idu u razvoj aplikacije ne znajući koliko će ih to zapravo koštati te hoće li uopće taj razvoj biti isplativ.

Izazovi ne prestaju ček i kad je razvoj aplikacije gotov. Oni se pojavljuju i kod faze održavanja aplikacije gdje poduzeće pruža korisničku potporu i izdaje ažuriranja za aplikaciju. Izazovi koji se javljaju su slijedeći [30]:

- **Kompleksan proces izdavanja ažuriranja** – kako bi aplikacija ostala konkurentna, poduzeće mora konstantno izdavati ažuriranja koja uvažavaju korisničke potrebe i potrebe tržišta. Ažuriranje aplikacije zahtjeva puno vremena i resursa ali je obvezno kako bi se zadržali korisnici aplikacije te stekli novi.
- **Težak proces prelaska na nove tehnologije** – ponekad, tehnologija postaje zastarjela te je potrebno aplikaciju prebaciti na novu tehnologiju. To zahtjeva mnogo vremena, resursa i novog znanja. Ukoliko aplikacija ne prijeđe na novu tehnologiju, riskira s greškama te sigurnosnim prijetnjama.

3.7. Usporedba web aplikacija otvorenog i zatvorenog koda

Spomenutu je već da je glavna razlika između aplikacija dostupnost izvornog koda. Čineći izvorni kod vidljiv zajednici, otvara se velik broj mogućnosti za razvoj aplikacije. Ne samo da neka aplikacija dobiva „besplatnu radnu snagu“ već dobiva i nove ideje, prijedloge te komentare. Prednosti razvoja aplikacije otvorenog koda su velike: brži i efikasniji razvoj, neovisnost o ciljevima firme te visoka razina sigurnosti samo se neke od mnogih prednosti. S druge strane, aplikacije zatvorenog koda većinom se razvijaju se za potrebe neke tvrtke ili na zahtjev tržišta. One općenito imaju manje slobode u razvoju jer su ograničene nekim definiranim zahtjevima klijenta i ciljevima organizacije. Sve u svemu, oba načina razvoja aplikacije imaju svojih prednosti i mana. Aplikacije zatvorenog koda ovise o tvrtki koja ih održava i čini dostupnima, dok aplikacije otvorenog koda ovisi o zajednici koja konstantno

ažurira aplikaciju te ju prilagođava novim tehnologijama i novim okolinama. Obje vrste aplikacija nalaze se na nekom repozitoriju. Razlika je samo što se aplikacije otvorenog koda nalaze na javnom repozitoriju dok se aplikacije zatvorenog koda nalaze na privatnim repozitorijima neke firme.

Glavna razlika leži u **cijeni**. Naravno, otvoreni softver ne mora nužno značiti i besplatni softver isto tako, zatvoreni softver ne mora imati neku inicijalnu cijenu. Razlika je u tome što je zatvoreni softver napravljen s ciljem neke zarade. Postoje različite pretplate, dodana plaćanja ili neke druge stavke koje čine zatvoreni softver skupljim za korištenje. Moguće je da zatvoreni softver ima neki besplatni plan, ali šansa je da je taj plan limitiran po funkcionalnostima ili traje samo kraći vremenski period. S druge strane, otvoreni softver većinom je besplatan za korištenje te dolazi s svim funkcionalnostima. Kod takvog softvera većinom se naplaćuju neke nadogradnje koje je napravila neka osoba te želi zaraditi za svoj trud. Inicijalni softver trebao bi uvijek biti besplatan. Aplikacije otvorenog koda pružaju **slobodu prilagodbe** softvera vlastitim potrebama. Daju mogućnost korisniku da promijeni dizajna korisničkog sučelja, dodaje nove funkcionalnosti, optimizira trenutne funkcionalnost i slično. Naravno, sloboda personalizacije ovisi o licenci s kojom dolazi softver. Neke licence dopuštaju potpunu izmjenu koda programa, dok druge licence limitiraju mogućnosti personalizacije. Uz sve to, korisnik mora posjedovati potrebno znanje za izmjenu koda programa. To može uključivati poznavanje određenih programskih jezika, arhitekture aplikacije ili operacijskog sustava na kojem se aplikacija izvodi. S druge strane, aplikacije zatvorenog koda imaju jako male mogućnost prilagodbe. Organizacija u vlasništvu softvera odlučuje o eventualnim promjenama u aplikaciji i te promjene će najvjerojatnije zahvatiti sve korisnike. Ukratko, pojedinac ne može utjecati na ažuriranja koja će tvrtka izdati. On može davati prijedloge za poboljšanja, ali bez poznavanja koda može slabo utjecati na izmjene. Organizacija odlučuje koje će greške ispraviti, koje će funkcionalnosti dodati te koja će poboljšanja napraviti na aplikaciji. Unatoč tome, aplikacije zatvorenog koda često ima bolje razvijeno **korisničko iskustvo** od otvorenih aplikacija. Prilagodba korisničkog sučelja ciljanim korisnicima često ostvaruje veći profit organizaciji. Korisnici žele koristiti aplikaciju koju razumiju i u kojoj se snalaze. Dobar dizajna korisničkog sučelja, jednostavna navigacija te razumljivost ključni su atributi aplikacija zatvorenog koda. Otvorene aplikacije rijetko kad imaju dobar dizajna korisničkog sučelja i razvijeno korisničko iskustvo jer se više fokusiraju na funkcionalnost nego na sam izgled. Naravno, ne mora nužno biti tako, dizajna dosta ovisi o ciljevima projekta i o osobama koje održavaju taj projekt pošto se trendovi u dizajna često mijenjaju i korisničko sučelje mora biti konstantno ažurirano kako bi održalo neku razinu estetiku. Aplikacije se razlikuju i po **korisničkoj, ali i tehničkoj podršci**. Kod aplikacija otvorenog koda, postoji standardna podrška kod licenciranih softvera. Na primjer, softver distribuiran od strane Red Hat-a i SUSE-a ima dovoljno podrške. Ostali otvoreni

softver, koji je slabo ili uopće nije licenciran nema korisničke podrške, no korisnici mogu koristiti razne forume kako bi riješili neki problem. Aplikacije zatvorenog koda dolaze s obveznom podrškom ovisno o sklopljenom ugovoru. Razina te podrške ovisi o takozvanom SLA (Service Level Agreement) ugovoru. Korisnička i tehnička podrška često je dio plana koji je korisnik kupio te kao takva mora biti osigurana. Velika razlika između vrsta aplikacija leži u **sigurnosti**. Izvorni kod otvorenih aplikacija dostupan je svima na pregled. Oko nekih otvorenih aplikacija okuplja se velik broj zajednica koje redovito vrše reviziju koda kako bi osigurali da u njemu nema grešaka i sigurnosnih rizika. Naravno, ranjivosti mogu postojati i može ih biti teže ukloniti ako se radi o kompleksnoj aplikaciji s mnogo funkcionalnosti. Kod aplikacija zatvorenog koda, vlasnik softvera zadužen je za pronalazak grešaka i ranjivosti u aplikaciji. Ukoliko se radi o manjoj firmi ili na aplikaciji radi manji broj ljudi, moguće je da će proći mjeseci bez da se neka greška primijeti i onda još mjesec da se ona ispravi. Vlasnik softvera se također obvezuje ugovorom da će u najkraćem roku ispraviti greške i osigurati da se one više ne pojave. **Stabilnost** aplikacije ovisi o zajednici koja ju održava (ako se radi o otvorenim aplikacijama) ili o vlasniku softvera (ako se radi o aplikacijama zatvorenog koda). Kod otvorenih aplikacija, vijek trajanja aplikacije ovisi o broju aktivnih korisnika i broju godina na tržištu. Zajednica koja održava takvu aplikaciju može odustati od nje ukoliko smatra da je više nije vrijedno održavati. Kod zatvorenih aplikacija, vlasnik softvera odlučuje o njenom vijeku trajanja. Ako smatra da aplikacija ne donosi dobru zaradu te samo troši resurse, izdavač ju može zauvijek ugasiti. O **popularnosti** vrsta aplikacija dalo bi se diskutirati. Kako tehnologija napreduje, otvorena rješenja su sve više popularnija i prihvaćena od strane zajednice. Neki projekti čak su i glavni na tržištu poput Linuxa i Apachea. Naravno, postoje grane industrije u kojima su vlasnički softveri još uvijek na visokoj razini popularnosti te se nalaze na tržištu dugi vijek godina. Radi se o Microsoftovim proizvodima kao i o industriji igara. **Interoperabilnost** s drugim softverom ovisi o razini održavanja aplikacije kao i o njenim ciljevima i standardima razvoja. Interoperabilnost je često na većoj razini kod otvorenog softvera nego kod zatvorenog. Aplikacije se razlikuju i po **mogućnosti dodavanja nove funkcionalnosti ili poboljšanja trenutne**. Kod aplikacija otvorenog koda, korisnik sam može razviti novu funkcionalnost ili poboljšati trenutnu, dok kod aplikacija zatvorenog koda, korisnik mora poslati zahtjev za razvoj nove funkcionalnosti te taj zahtjev mora biti odobren od strane vlasnika softvera. Zatvoreni softver imaju prednost nad otvorenim kad se radi o **prilagođenosti za produkcijsku okolinu**. On prolaze kroz različite rotacije testiranja te su manje šanse da će se softver ponašati nepredvidljivo prilikom izvršavanja u produkcijskoj okolini. Šanse su da otvoreni softver neće biti tehnički dobro dizajniran ili testiran na većoj skali za produkcijsku okolinu. Također, zatvoreni softver dolazi s **garancijom** te više odgovara tvrtkama sa sigurnosnim policama i osiguranjem odštete. Otvoreni softver ne dolazi s takvim jamstvom. [31]

4. Licence

Ovo poglavlje fokusira se na licence. Osim teorije o licencama, sadrži i opise vrsta licenci koje postoje de opisuje najčešće licence koje se koriste sa softverom otvorenog koda. Licence zapravo definiraju ograničenja nekog softvera. Iako se radi o softveru otvorenog koda, on ima i svojih ograničenja prilikom korištenja. Ta ograničenja postavlja vlasnik softvera ili zajednica prilikom izbor licence. Neke licence definiraju manja ograničenja dok postoje licence koje otvoreni softver približavaju komercijalnom softveru u smislu ograničenja koje postavljaju.

Mnoge organizacije koje razvijaju vlastite, zatvorene, aplikacije imaju svoje licence koje provode na većini svojih proizvoda. Ovo poglavlje ne bavi se takvim licencama, već se bavi ***licencama otvorenog koda***. Te licence dopuštaju softveru da bude slobodno korišten, modificiran i dijeljen. Kako bi neka licenca bila odobrena od strane OSI-a (Open Source Initiative) , mora proći kroz OSI postupak pregleda licence (engl. Open Source Initiative's license review process.) službena stranica za softver izvornog koda navodi sljedeće licence kao najpopularnije i najčešće korištene od strane zajednice [32]:

- Apache License 2.0
- BSD 3-Clause "New" or "Revised" license
- BSD 2-Clause "Simplified" or "FreeBSD" license
- GNU General Public License (GPL)
- GNU Library or "Lesser" General Public License (LGPL)
- MIT license
- Mozilla Public License 2.0
- Common Development and Distribution License
- Eclipse Public License version 2.0

Postoje još mnoge druge licence koje su odobrene od strane OSI zajednice ali spadaju u druge kategorije poput: licence za specijalnu namjenu, zamijenjene licence te umirovljene licence.

4.1. Licence otvorenog koda

Licence otvorenog koda su legalni ugovorni između autora softvera i njegovog korisnika te deklariraju da se neki softver može koristiti u komercijalne svrhe pod posebnim uvjetima. Licenca pretvara izvorni kod softvera u otvorenu, javno dostupnu, komponentu. Bez licence, softver nije koristan drugima bez obzira da li je on objavljen na nekom javnom repozitoriju.

Svaka licenca otvorenog koda navodi što korisnici smiju raditi s softverom, njihove dužnosti i radnje koje ne smiju raditi prema pravilima navedenim u ugovoru sa odredbama i obvezama. Danas, postoji mnogo licenci od kojim se za licenciranje softvera mora odabrati jedna. Te licence variraju po svojoj kompleksnosti i zahtjevima te je na vlasniku softvera da odluči koja licenca najbolje odgovara njegovim potrebama. [33]

Postoji velik broj ljudi pa čak i programera koji misle da otvoreni softver znači besplatan softver koji je moguće koristiti, kopirati, modificirati i distribuirati bez ikakvih ograničenja. To je naravno, netočno. Pojmovi otvorenog softvera i javne domene često se miješaju. Softver koji je javne domene ili takozvani „shareware“ moguće je besplatno koristiti bez specijalnih dozvola ili licenciranja. Kao što je bilo navedeno, softver otvorenog koda dolazi licencom te nije nužno besplatan. Suprotno od zatvorenog softvera gdje vlasnik zabranjuje pristup, kopiranje ili modifikaciju izvornog koda, softver otvorenog koda dopušta upotrebu, ponovo korištenje, dijeljenje, modifikaciju i distribuciju koda drugim programima ili aplikacijama. Također, isto kao kod licenciranja softvera zatvorenog koda, otvoreni softver također podliježe različitim legalnim uvjetima i ograničenjima ovisno o tipu otvorene licence koja se koristi. Stoga je važno ostati u skladu s uvjetima licenciranja softvera otvorenog koda korištenjem ispravnih i odobrenih otvorenih licenci. [33]

Licence za softver otvorenog koda određuju kako drugi, osim vlasnika, mogu koristiti, mijenjati ili distribuirati izvorni softverski kod. Daju drugim korisnicima dopuštenja i prava da ponovo koriste kod u razne svrhe poput ponovnog korištenja koda u novoj aplikaciji ili uključivanje koda u druge projekte. Jedna od glavnih prednosti otvorenog koda je njegova javna dostupnost. Ta dostupnost olakšava pronalazak i rješavanje problema te bolje razumijevanje kako nešto funkcionira kad dokumentacija nedostaje ili je ona netočna. Ovisno o vrsti licence koja se koristi, korisniku je dopušteno modificirati izvorni kod kako bi ga prilagodio vlastiti potrebama ili riješio neku grešku u programu. Licenca će odrediti da li je takva radnja moguća i pod kojim uvjetima. Na primjer, licenca može obvezati korisnika da svaku modifikaciju koda mora učiniti javno dostupnom.[33]

Postoje dvije glavne vrste otvorenih licenci: **Copyleft** licence i **Permissive** licence. Svaka licenca obrađena je u zasebnom poglavlju zajedno s licencama koje spadaju u tu vrstu. Licence se razlikuju po zahtjevima i ograničenjima koje stavljaju na korisnike.

4.1.1. Copyleft licence

Copyright ili u prijevodu „autorsko pravo“ je zakon koji zabranjuje osobama da koriste, modificiraju ili dijele nečiji rad bez dozvole od originalnog vlasnika odnosno autora rada. Pojam autorskog prava koristiti u različitim industrijama: filmskoj, glazbenoj pa čak i softverskoj

industriji. Kada neki autor izdaje program pod Copyleft licencom, tada oni „podnose“ zahtjev za Autorsko pravo (engl. Copyright) nad projektom i izdaju izjavu da drugi ljudi imaju pravo koristiti, modificirati i dijeliti projekt pod uvjetom da se pridržavaju pravila koje daje ta licenca. Glavno pravilo kojeg se moraju pridržavati je: ako koriste neku komponentu koja ima ovakvu vrstu otvorene licence, tada moraju svoj kod učiniti otvorenim kako bi ga drugi također mogli koristiti. [33]

Najpopularnije Copyleft licence, poredane od najrestriktivnije prema najblažoj, su sljedeće:

- GNU General Public License (GPL)
- Affero GPL (AGPL)
- Lesser General Public License (LGPL)
- Eclipse Public License (EPL)
- Mozilla Public License (MPL)

GNU General Public License (GPL) je licenca koja dolazi u više verzija i jedna je od najpopularnijih licenci. Trenutno najnovija verzija je GPL-3 verzija.. Licenca čuva autorska prava i pogodna je za komercijalnu, patentiranu ili privatnu upotrebu. Svaki softver koji koristi kod po GPL licencom mora distribuirati sav svom izvorni kod pod istom licencom. Ovakvo ograničenje čini ovu licencu popularnom u zajednici koja razvija otvoreni softver. Ukratko, licenca dopušta kopiranje, distribuciju i modifikaciju softvera uz uvjet da osoba mora bilježiti datume promjena u izvornom kodu. Svaka modifikacija softvera licenciranog ovom licencom mora biti dostupna pod istom licencom zajedno s popratnim uputama u izgradnji i instalaciji modifikacije. [33,34]

Affero GPL (AGPL) licenca dosta je slična GPL licenci te dodaje samo jednu klauzulu koja je dosta važna određenim tipovima softvera. Pošto se GPL licenca „aktivira“ prilikom distribucije softvera, postoji rupa u licenci koju mogu iskoristiti softveri koji su dostupni samo preko mreže odnosno, nisu distribuirani u tom smislu riječi. AGPL licenca rješava tu rupu na način da dodaje klauzulu koja se odnosi na interakciju s udaljenom mrežom te aktivira GPL licencu za bilo koji softver koji se koristi preko interneta. [34]

Lesser General Public License (LGPL) licenca daje iste uvjete kao i AGPL i GPL licence , no namijenjena je za manje projekte ili biblioteke (engl. Library). Glavna značajka licence leži u tome da manji projekti ili objekti kojima se pristupa putem većih licenciranih softvera ne zahtijevaju distribuciju. Također, modificirani izvor ne treba biti distribuiran pod istim uvjetima kao i veći projekt.[34]

Eclipse Public License (EPL) je licenca otvorenog koda razvijena od strane Eclipse zajednice. Licenca je slična GPL licenci te omogućava povezivanje koda pod tom licencom s zatvorenim softverom. EPL se često koristi za poslovni softver. S ovom licencom, softver razvijen korištenjem EPLa može se kombinirati s elementima zatvorenog softvera. [55]

Mozilla Public License (MPL) najmanje je restriktivna licenca otvorenog koda koja spada u kategoriju Copyleft licenci. Licenca dopušta modificiranje i korištenje otvorenog koda u softveru zatvorenog koda ako se taj kod drži u zasebnim datotekama i pravilno je licenciran i distribuiran. [34]

4.1.2. Permissive Licence

Permissive ili takozvana dopuštajuća licenca otvorenog koda suprotna je Copyleft licenci i garantira slobodu korištenja, modifikacije i redistribucije te dopušta izdavanje vlastitih izvedenica iz originalnog softvera. Ovakva vrsta licence stavlja minimalna ograničenja na načine korištenja softverskih komponenti od strane javnosti. Permissive licenca otvorenog koda dopušta različite načine korištenja, modifikacije i distribucije softvera otvorenog koda, dopuštajući njegovu upotrebu pri razvitku vlastitog softvera i ne zahtijevajući pritom ništa za uzvrat. Ukratko, za razliku od Copyleft licence, ova licenca ne zahtijeva javnu objavu vlastitog softvera ukoliko se koristio tuđi, javni kod. [32]

Najpopularnije Permissive licence otvorenog koda su:

- Apache License
- MIT License
- Berkeley Source Distribution (BSD) License
- Unlicense

Apache License je licenca koja dopušta upotrebu, modifikaciju i distribuciju koda te samo zahtjeva navođenje autora originalnog koda u distribuirani kod ili kao obavijest u softveru. Također, projekti izvedeni iz otvorenog koda, veći projekti ili modifikacije imaju dopuštenje da nose različite uvjete licence kad se dijele i ne trebaju pružati izvorni kod javnosti. Apache licence daju potporu patentima.[34]

MIT License jedna je od najkorištenijih licenci otvorenog koda na svijetu jer je kratka i razumljiva. Dopušta svakome da radi što god želi s originalnim izvornim kodom, tako dugo dok su originalna autorska prava i uvjeti licence uključeni u distribuirani izvorni kod ili softver.[34]

Berkeley Source Distribution (BSD) License je licenca koja je podijeljena u više vrsta. BSD licenca s dvije klauzule slična je MIT licenci, dok BSD licence s tri i četiri klauzule definiraju veći broj ograničenja i zahtjeva za ponovo korištenje. [34]

Unlicence najmanje je restriktivna licenca otvorenog koda jer za cilj ima učiniti otvoreni kod dostupnim javnosti. Nema nikakve uvjete, te se izvedeni softver može distribuirati bez izvornog koda i pod različitim uvjetima. [34]

4.1.3. Usporedba licenci otvorenog koda

Donja slika prikazuje glavne uvjete i prava pojedinih licenci. Daje do znanja što koja licenca dopušta, zabranjuje ili obvezuje. Detaljnija objašnjenja licenci mogu se pronaći na službenoj web stranici: https://tldrlegal.com (prema [34]).

	Uvjeti/Licence	COPYLEFT					PERMISSIVE			
		GPL	AGPL	LGPL	EPL	MPL	Apache	MIT	BSD	Unlicence
DOPUŠTA	Komercijalnu upotrebu softvera	X	X	X	X	X	X	X	X	X
	Modifikaciju	X	X	X	X	X	X	X	X	X
	Distribuciju	X	X	X	X	X	X	X	X	X
	Stavljanje jamstva na softver	X	X	X		X	X		X	
	Stavljanje patenta na doprinos u kodu softvera	X		X	X	X	X			
	Pod – licenciranje (engl. Sublicense)				X	X	X	X	X	X
	Privatno korištenje softvera				X		X	X		X
	Ponovo licenciranje softvera									X
ZABRANJUJE	Pod-licenciranje	X	X	X						
	Držanje vlasnika softvera odgovornim za štetu	X	X	X	X	X	X	X	X	X
	Korištenje tuđih zaštitnih znakova (engl. Trademarks)				X	X	X		X	
OBVEZUJE	Uključivanje originalnog softvera prilikom distribucije vlastitog	X		X	X	X				
	Navođenje signifikantnih promjena u odnosu na originalni softver	X	X							
	Licenciranje vlastitog softvera uvjetovanom licencom	X								
	Uključivanje punog teksta licence u vlastiti softver	X	X	X	X	X	X	X	X	
	Uključivanje autorskih prava u vlastiti softver	X	X	X	X	X	X	X	X	
	Uključivanje instalacijskih instrukcija	X	X	X	X					
	Učiniti izvorni kod dostupnim na najmanje 3 godine									
	Pružati jamstvo korisnicima za vlastite modifikacije na softveru				X					
	Učiniti vlastiti izvorni kod javno dostupnim		X	X	X	X				
	Naglašavanje bitnih promjena u odnosu na originalni kod						X			
	Uključivanje potrebnih bilješki (engl. NOTICE file)						X			

Slika 8: Uvjeti licenci otvorenog koda (Prema: [34])

4.1.4. Postoci korištenja licenci i razina ograničenja

Donja tablica sadrži danas najpopularnije licence otvorenog koda. Dan je postotak njihovog korištenja u praksi te razina ograničenja koje sadrže. Kao što je bilo spomenuto u prijašnjim poglavljima, Copyleft licence definiraju više ograničenja nego Permissive licence koja daju više slobode.

Tablica 4: Postoci korištenja licenci i ograničenja (Prema: [35])

Pozicija	Licenca	Postotak korištenja	Ograničenja
1	MIT License	32%	Mala
2	GNU General Public License (GPL) 2.0	18%	Velika
3	Apache License 2.0	14%	Mala
4	GNU General Public License (GPL) 3.0	7%	Velika
5	BSD License 2.0 (3-clause, New or Revised)	6%	Mala
6	ISC License	5%	Mala
7	Artistic License (Perl)	4%	Srednja
8	GNU Lesser General Public License (LGPL) 2.1	4%	Velika
9	GNU Lesser General Public License (LGPL) 3.0	2%	Velika
10	Eclipse Public License (EPL)	1%	Srednja
11	Microsoft Public License	1%	Srednja
12	Simplified BSD License (BSD)	1%	Mala
13	Code Project Open License 1.02	1%	Mala
14	Mozilla Public License (MPL) 1.1	<1%	Srednja
15	GNU Affero General Public License 3.0 or later	<1%	Mala

5. Analiza web aplikacija

Analiza postojećih web aplikacija daje uvod u implementaciju vlastite web aplikacije. Analizom se prikupljaju ideje i analizira trenutno stanje sličnih web aplikacija. Kao primjer web aplikacije otvorenog koda, izrađena je web aplikacija za davanje osvrta (recenzija) na knjige. U ovom poglavlju vrši se analiza nad jednom web aplikacijom koja je izrađena u svrhu davanje osvrta na knjigu i nad jednom web aplikacijom koja daje osvrte na filmove te ima dostupan izvorni kod na repozitoriju. Proučavanjem navedenih aplikacija zadobivena je ideja za dizajn i razvoj vlastite web aplikacije otvorenog koda.

Aplikacije koje su analizirane korištene su s aspekta običnog korisnika. Prva stvar koju treba odrediti je svrha aplikacije. Naravno, već je bilo napomenuto da se radi o aplikacijama za davanje osvrta, no postavlja se pitanje da li je to jedina svrha aplikacije te da li ima neke definirane ciljeve poput: zarade, edukacije i slično. Kao korisnik, postoji ograničen skup funkcionalnosti kojima se može pristupiti, tako da radnjama koje obavljaju moderatori i administratori, pristup nije omogućen. Aplikacija za osvrt na filmove ima dostupan izvorni kod, tako da se samom analizom koda mogu utvrditi mogućnosti aplikacije, no radi se o aplikaciji manjeg obujma. Više informacija o samim aplikacijama, kao i pojedinačna analiza, nalazi se u donjim poglavljima.

Veliki fokus stavljen je na strukturu aplikacije. Radi se o načinu navigacije kroz aplikaciju te lokacijama pojedinog pogleda. Tu se dobivaju odgovori na pitanja poput: Kako je podijeljena aplikacija? Gdje se nalazi pogled s informacijama o pojedinoj knjizi ili filmu? Gdje su osvrti na tu knjigu/film? Kako je realizirano dodavanje novih osvrta na knjigu/film? Kako se može pratiti pojedini knjiga/film? Gdje se mogu vidjeti dani osvrti na knjigu/film?. Nadalje, praćene su i mogućnosti koje ima korisnik. Na koji način on može sve utjecati na sadržaj same web aplikacije, koje su njegove odgovornosti i zadaće.

Manji fokus stavljen je na dizajn same aplikacije kao i na dizajn korisničkog iskustva. Dizajna je jedan od aspekata aplikacije koji sam želio sam implementirati. Naravno, određene ideje su dobivene iz analize poput: pozicije elemenata, načina navigacije kroz aplikaciju (kroz glavnu navigaciju, ali i sporedne navigacije poput pritiska na sliku knjige i slično), logike dodavanja novih osvrta, izgled i pozicija određenih formi i slično.

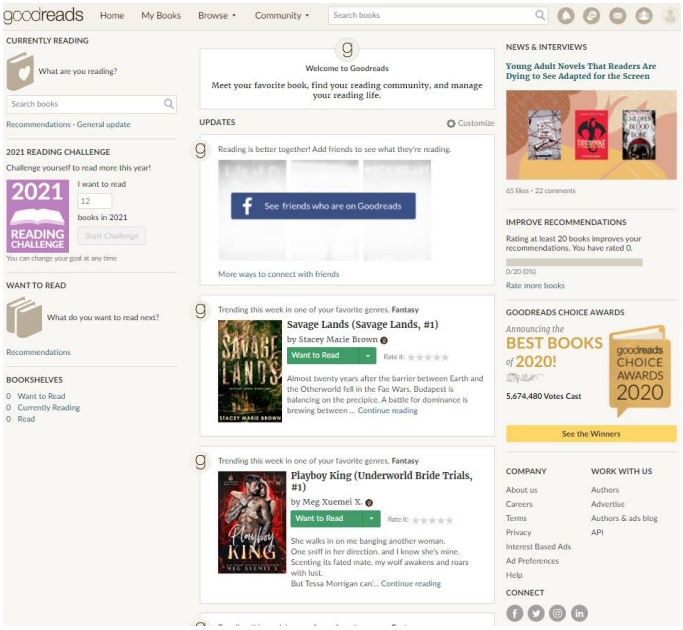
Na kraju, dan je i vlastiti osvrt na analizirane stranice. Navedene su prednosti i mane, te pobrojeni elementi koje bih promijenio te elementi koje bih mogao iskoristiti prilikom implementacije vlastite web aplikacije na tu temu.

5.1. Goodreads

Web aplikacija služi za davanje osvrta na pročitane knjige. Službeno se nalazi na domeni: https://www.goodreads.com/ gdje se njenim funkcionalnostima može pristupiti i bez kreiranja računa. Aplikacija nema javno dostupan izvorni kod što je čini **aplikacijom zatvorenog koda.** Ova aplikacija je odabrana za analizu jer je najpoznatija **web aplikacija za davanje osvrta na knjige.** Omogućuje pregled knjiga koje su čitali vaši prijatelji, praćenje knjiga koje korisnik želi čitati, dobivanje preporuka knjiga prema integriranom sustavu za preporuku. Također, omogućuje dobivanje preporuka knjiga prema preporukama od strane zajednice. Iako aplikacija omogućuje davanje osvrta na knjige zajedno s ocjenama (1-5), to nije njena glavna funkcionalnost. Vlasnik aplikacije tvrdi kako je glavni razlog izgradnje ove aplikacije dijeljene knjiga i mišljenja o knjigama s prijateljima.

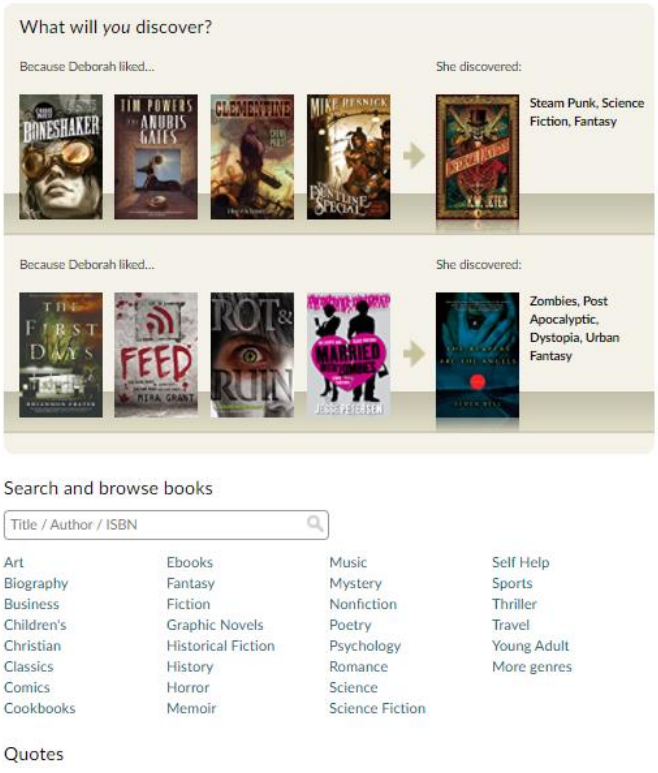
Analizu započinjem na početnoj stranici. Otvaram dva paralelna web preglednika: jedan preglednik gdje nisam prijavljen (uloga gosta) i jedan preglednik gdje sam kreirao račun i gdje sam prijavljen (uloga korisnika). Prijava je riješena na regularan način gdje aplikacija zahtjeva vaše ime, prezime, email adresu te izabranu lozinku. Prijava je dosta jednostavna jer samo zahtjeva potvrdu email pošte u nekom vremenskom roku, nema potvrde lozinke, odabira korisničkog imena ili captcha potvrde. Prijavljeni korisnik ima znatno drugačiji izgled početne stranice od ne prijavljenog korisnika.

Donja slika prikazuje početnu stranicu koja se renderira za prijavljenog korisnika. Odmah je vidljivo da izgledom podsjeća na neku društvenu stranicu. Razlog tome je upravo taj pristup „dijeljenja knjiga s prijateljima“. Ima jednostavnu navigaciju u zaglavlju, pretragu te elemente društvene stranice poput notifikacija, poruka, pregleda objava s knjigama i slično. Što se tiče samog prikaza knjiga, koristi se dosta jednostavni dizajn: prikazuje se naslovnica knjige, naziv knjige, autor, kratki opis te mogućnost dodavanje knjige u vlastitu kolekciju. Vidljivo je kako je sama aplikacija dosta razvijena uz višestruke mogućnosti pretrage prema knjigama, autorima, korisnicima i slično. Odabirom opcije navigaciji pod nazivom „My books“ otvara novi pogled koji sadrži kolekciju korisničkih knjiga. Taj pogled prikazuje tablični prikaz svih knjiga s stupcima: naslovnica, naslov knjige, autor, prosječna ocjena, datumi te stupcima koji prikazuju vlasti osvrt i ocjenu. Nadalje, aplikacija ima svoj integrirani sustav za preporuke koji zahtjeva dodavanje od 20 knjiga u svoju kolekciju kako bi mogao biti korišten. Taj sustav koristi algoritme ML- a (Machine Learning) čije baza sadrži do 20 bilijuna redaka za učenje.



Slika 9: Početna stranica korisnika- Goodreads [Izvor: goodreads.com]

Iako gost dijeli određeni broj funkcionalnosti s prijavljenim korisnikom, njegov početni pogled je znatno drugačiji. Korisnički pogled početne stranice prikazuje donja slika. Pogled sadrži nekoliko preporučenih knjiga izabranih od strane aplikacije te mogućnost pretraživanja prema nekoj ključnoj riječi ili žanru. Jedan od nedostataka koji sam odmah ovdje uočio je nedostatak klasične navigacije koja se pojavi tek kada je neka knjiga izabrana.



Slika 10: Početna stranica gosta - Goodreads [Izvor: goodreads.com]

Također postoje zasebni pogledi za knjige koje su tek izašle te knjige koje su izabrane kao najbolje knjige za tu godinu. Odabirom na neku knjigu otvara se zaseban pogled koji sadrži detaljne informacije o knjizi. Tamo se omogućuje korisniku da pročita osvrte na knjigu te da doda vlastiti osvrt. Korisnik može također postaviti neko pitanje vezano za knjigu. Može knjigu dodati i u vlastitu kolekciju.

Slika 11: Pregled detalja knjige - Goodreads [Izvor: goodreads.com]

Karakteristike i funkcionalnosti koje smatram pozitivnima i dobro implementiranima i koje bih mogao iskoristiti u vlastitoj aplikaciji su sljedeće:

- Početna stranica koju vidi gost – pogled je dosta jednostavan i kroz minimalistički dizajn ostvaruje vlastite ciljeve: pretragu i prikaz knjiga
- Pogled koji prikazuje detalje knjige – Pogled ima jednostavan i lijep dizajn te prikazuje sve što je potrebno za pojedinačnu knjigu. Lagano se mogu pročitati i pronaći osvrti na knjigu
- Način dodjele osvrta – Osvrti se dodaju na pogledu detalja knjige te zahtijevaju samo dodavanje ocjene (1-5) te opcionalnog kratkog opisa
- Navigacije – navigacija je dobro strukturirana i vodi do željenih pogleda

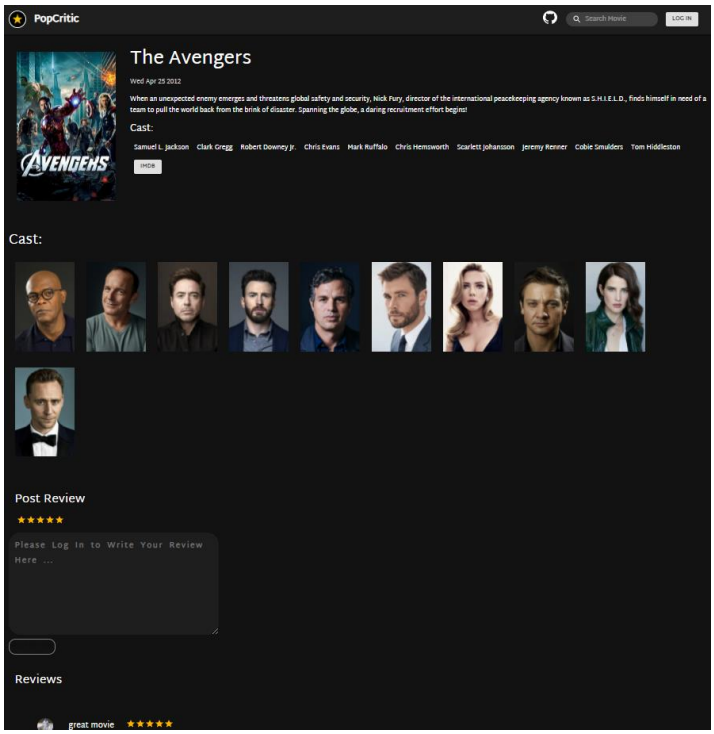
Funkcionalnosti koje ne bih iskoristio su: početni pogled korisnika jer moja aplikacija ne bi išla u smjeru neke društvene mreže već bi se više fokusirali na same osvrte. Također, ne bih knjige dijelio po različitim pogledima već bi ih držao na jednom pogledu uz mogućnosti filtriranja.

5.2. PopCritic

PopCritic je **web aplikacija otvorenog koda** napisana pomoću ReactJS-a. Aplikacija se trenutno izvodi na domeni: https://popcritic.web.app/ te je njen izvorni kod dostupan na javnom repozitoriju githubu: https://github.com/theabbie/PopCritic. Projekt je službeno zaštićen MIT licencom. Osim spomenutog ReactJS-a, u projektu su još korištene i tehnologije: NodeJS i PostgreSQL za razvoj baze podataka. Na projektu radi manji broj ljudi te postoji mogućnost uključenja na projekt uz prijašnji kontakt jednog od člana tima. Aplikacija primarno služi za dodavanje osvrta na filmove te pregled istih.

Donja slika prikazuje pogled detalja jednog filma. Odmah je vidljivo da pogled nije toliko detaljizirani kao kod prijašnje analizirane knjige. No, ova aplikacija je posebice zanimljiva jer ne koristi nijedne vlastite podatke već se filmovi, podaci o filmovima, glumci te sve slike dohvaćaju preko API poziva. Iako je dizajn dosta jednostavan, aplikacija ispunjava svoju svrhu i predstavlja dobro polazište za implementaciju glomaznije aplikacije.

Putem analize ove aplikacije dobio sam ideju za implementaciju funkcionalnosti na vlastitoj aplikaciji gdje bi se podaci također dohvaćali putem API poziva uz mogućnost vlastite izmjene istih radi manjih korekcija.



Slika 12: Detalji filma - PopCritic [Izvor: popcritic.web.app]

6. Izrada vlastite web aplikacije

Posljednje veliko poglavlje ovog rada odnosi se na osmišljanje i implementaciju vlastite web aplikacije otvorenog koda. Za izradu aplikacije iskorišteno je znanje naučeno tijekom fakultetskog obrazovanje kao i nova znanja naučena prilikom izrade ovog rada. Aplikacija se trenutno nalazi na javnom repozitoriju Github-a[1]. Tamo je dostupan izvorni kod aplikacije licenciran pod GPLv3 licencom. Licenca je odabrana iz razloga što dopušta korištenje izvornog koda, ali uz uvjet da korisnik mora učiniti i njegov kod otvorenim. Na repozitoriju se nalaze i upute za postavljanje aplikacije kao i upute za generiranje postavljanje baze podataka te skripte za generiranje tablica i testnih podataka.

Web aplikacija zamišljena je kao mala knjižnica u kojoj korisnik može pretraživati knjige te davati osvrte na pročitane knjige. Naravno, aplikacija ima veći broj funkcionalnosti koje se nadopunjavaju i vežu se na davanje osvrta (recenzija) na knjige. Web aplikacija podijeljena je u dvije zasebne aplikacije: administratorsku i korisničku. Opis domene aplikacije, njene funkcionalnosti kao i sama struktura aplikacije, opisani su u zasebnom poglavlju.

Kroz teorijski dio ovog rada, bile su navedene tehnologije za razvoj web aplikacija. Spominjala su se razvojna okruženja, različiti okviri i biblioteke koje zajednički tvore programski proizvod. Bez tih tehnologija, izrada ove aplikacije bila bi otežana i gotovo nemoguća. Posebno se obraćala pažnja na odabir tehnologije koje su otvorenog koda. Korištene tehnologije nalaze se pod otvorenom licencom te su pogodne za korištenje. Više o tehnologijama korištenim kod izrade aplikacije može se pročitati u zasebnom poglavlju.

Za potrebne testiranja, koristi se skup podataka s knjigama. Koristi se i API za dohvat podataka o knjigama. Sve te podatke treba negdje pohraniti. Za to se koristi JavaDB baza podataka. Baza sadrži tablice koje čuvaju podatke i iz kojih kasnije nastaje model podataka. Detalji implementacije baze mogu se pročitati u zasebnom poglavlju.

Na izradu aplikacije utrošeno je 2 mjeseca rada gdje se svaki dan radilo po 4 sata. Najveći dio tog vremena potrošeno je na dizajn sučelja i realizaciju kontrolera koji dohvaća podatke iz baze te ih šalje na formu i obrnuto. Aplikacija je rađena po uzorku MVC arhitekture. Prikaz same aplikacije, detalji implementacije te navigacije kroz aplikaciju, nalaze se u zasebnom poglavlju: Implementacija i izgled korisničkog sučelja.

[1] URL do repozitorija: https://github.com/vbencek1/review_java_app

Posljednje poglavlje rezervirano je za davanje vlastitog osvrta na kreiranu aplikaciju. Ocjenjuje se postotak završenosti aplikacije ovisno o predviđenim funkcionalnostima kao i sam dizajn i implementirane funkcionalnosti.

6.1. Domena, funkcionalnosti i struktura web aplikacije

Ideja aplikacije je pružiti potporu korisnicima za evidentiranje pročitanih (ili onih koje žele pročitati) knjiga te im dati mogućnost za ocjenjivanje istih. **Ciljana skupina** korisnika za koje je aplikacija izrađena su osobe koje čitaju knjige, osobe koje žele proučiti knjigu prije kupnje te osobe koje žele informirati javnost o kvaliteti neke knjige. Dakle, glavna **svrha** aplikacije je omogućiti korisnicima davanje osvrta na knjige.

Web aplikacija sastoji se od dvije samostojeće aplikacije koje su podijeljene po tipu korisnika. Korisničku aplikaciju koriste obični korisnici – čitatelji knjiga. Administratorsku aplikaciju koriste moderatori i administratori čija uloga je nadzirati i dodavati sadržaj web aplikaciji. Svaka aplikacija zahtjeva vlastite korisničke račune, korisnička aplikacija može se koristiti i bez prijava ali uz limitirane mogućnosti.

Korisnička aplikacija ne zahtijeva prijavu. Gost ima limitirane mogućnosti. Može pristupiti početnoj stranici gdje mu se prikazuju preporučene knjige, popularne knjige te posljednje knjige s recenzijama. Preporučene knjige generiraju se prema unaprijed zadanom algoritmu bez praćenja rada korisnika. Uglavnom se preporučuju knjige koje imaju dobru ocjenu. Gost može pregledavati i detalje knjige. Tamo može vidjeti osnovne informacije o knjizi, recenzije koje su dane na tu knjigu kao i osnovnu statistiku za tu knjigu. Također, može i samostalno pretraživati knjige po kriterijima poput: ISBN-a, naziva knjige ili autora, minimalne ocjene, izdavača, godine izdanja i slično. Uz te radnje, gosti ima i mogućnost registracije kao i mogućnost obnove lozinke uz slanje korisničkog imena i adrese e-pošte. Također ima i mogućnost prijave u aplikaciju. Aplikacija ima dvojezičnu potporu pa korisnik može birati između hrvatskog i engleskog jezika.

Prijavom u aplikaciju gost postaje aktivni korisnik te mu se „otključavaju“ sve mogućnosti korisničke aplikacije. Početna stranica i stranica s pretraživanjem knjiga ista je kao i kod gosta s istim mogućnostima. Preporučene knjige imaju malo bolje prilagođeni algoritam za generiranje prikazanih knjiga. Odabirom jedne knjige prikazuju mu se detalji. Ovdje registrirani korisnik može dodati knjigu u vlastitu kolekciju te može ostaviti osvrt na knjigu. Prilikom davanje osvrta, korisnik uz odabir ocjene i pisanja teksta, može odlučiti hoće li taj osvrt biti javan ili privatni. Privatni osvrt vidljiv je samo osobi koja ga je objavila. Korisnik može spremati knjige u vlastitu kolekciju. Tamo te knjige može pretraživati i uklanjati po želji. Na

zasebnom pogledu, korisnik može pregledavati vlastite osvrte na knjige. Uz mogućnosti filtriranja, ima i mogućnost izmjene osvrta kao i mogućnost uklanjanja istog. Ukoliko korisnik želi dodate recenziju na knjigu koja ne postoji u bazi, tada mora slati zahtjev za dodavanjem knjige. Može poslati samo ISBN ili može poslati naziv knjige uz detalji opis. Slanjem ISBN-a, aplikacija automatski provjerava da li taj ISBN već postoji. Ukoliko ne postoji, šalje se zahtjev moderatorima koji dalje vrše određene radnje kako bi dodali knjigu u bazu. Korisnik može i pregledati vlastiti profil. Tamo se nalaze njegovi osobni podaci koje može ažurirati. Ažuriranje podataka zahtjeva unos lozinke zbog sigurnosnih mjera. Promjena emaila, kao i kod registracije, zahtjeva unos potvrdnog koda koji je poslan na e-poštu korisnika.

Administratorska aplikacija zahtjeva prijavu. Tu postoje dva tipa korisnika: Moderator i Administrator. Moderator ima malo manji broj funkcionalnosti od administratora. Aplikacija također ima dvojezičnu potvrdu. Nakon prijave, korisnika se preusmjerava na stranicu s popisom knjiga. Tamo moderator može filtrirati sve postojeće knjige. Odabirom neke od knjiga, moderator može vidjeti detalje te knjige ili promijeniti detalje te knjige. Također ima mogućnosti brisanja knjige kao i dodavanje nove knjige upisom podataka. Može pregledavati i sve recenzije koje su grupirane po knjigama. Recenzije može i uklanjati ukoliko to smatra potrebnim. Zatim može pregledavati zahtjeve za dodavanjem knjiga od korisnika. Ovo je jedna od najkompleksnijih funkcionalnosti ove aplikacije. Ukoliko je korisnik poslao zahtjev s ISBN—om, moderator može kreirati knjigu slanjem ISBN-a OpenLibrary API-u koji dohvaća i automatski puni podatke za tu knjigu. Detalji o načinu funkcioniranje takvog odavanja knjige opisani su u zasebnom poglavlju o tehnologijama. Može kreirati i novi knjigu preko zahtjeva s nazivom knjige. Tada mora sam puniti podatke koji nedostaju. Zahtjeve može i brisati. Nadalje, moderator vidi i popis korisnika aplikacije. Korisnike može blokirati ili odblokirati ovisno o statusu korisnika. Moderator može vidjeti i statistiku aplikacije. Na tom pogledu nalazi se osnovna statistika o ocjenama knjiga, broju osvrta, aktivnim korisnicima i slično. Moderator može i pregledavati vlastite informacije te ih može ažurirati.

Administrator ima iste mogućnosti kao i moderator uz dvije dodatne funkcionalnosti: pregled dnevnika rada i pregled moderatora. Na pogledu za dodavanje moderatora, administrator može filtrirati moderatore te im može blokirati/odblokirati račune. Može također dodati novog moderatora u sustav. Nadalje, može pregledavati i dnevnik rada. Dnevnik rada može biti filtriran po moderatorima kao i po običnim korisnicima.

Funkcionalnosti aplikacije podijeljene su prema tipu korisnika i vrsti aplikacije. Korisnik može biti tipa: Gost (Ne registrirani korisnik), Registrirani korisnik, Moderator i Administrator. Vrsta aplikacije je: Korisnička i Administratorska.

Funkcionalnosti korisničke aplikacije, grupirane prema tipu korisnika su sljedeće:

Gost (Ne registrirani/prijavljeni korisnik):

- **Prijava u aplikaciju** – Neki korisnik smatra se gostom tako dugo se ne prijavi u aplikaciji. Kako bi se mogao prijaviti, mora imati kreirani račun, odnosno valjano korisničko ime i lozinku. Također, korisnik se ne može prijaviti u aplikaciju ukoliko je njegov korisnički račun blokiran.
- **Registracija** – Gost može postati pravim korisnikom putem registracije. Za registraciju mu je potreban valjan email, korisničko ime koje još nije iskorišteno te lozinka. Nakon uspješne validacije podataka, korisniku se na račun e-pošte dostavlja potvrdni kod. Korisnik završava svoju registraciju unosom valjanog potvrdnog koda.
- **Oporavak lozinke** – Ukoliko je zaboravio vlastitu lozinku, gost može zatražiti da mu se nova lozinka dostavi na adresu e-pošte. Za oporavak lozinke, potrebno je unijeti valjano korisničko ime i e-mail.
- **Promjena jezika** – aplikacija trenutno podržava dva jezika: engleski i hrvatski. Korisnik može samostalno odabrati željeni jezik.
- **Pregled preporučenih, popularnih i nedavno ocijenjenih knjiga** – Početna stranica aplikacije sastavljena je od različitih elemenata koji dijele knjige na više načina. Korisniku se preporučuju knjige koje su popularne i dobro ocijenjene. Popularne knjige su one s najvećim brojem recenzije. Nedavno ocijenjene knjige su one koje su nedavno dobile osvrt.
- **Pregled detalja knjige** – Uključuje prikaz naslovnice knjige, detaljnih informacija o knjizi kao i prikaz osvrta na knjigu s detaljima osvrta i prikaz statistike za odabranu knjigu.
- **Pretraga knjiga** – Korisnik može pretraživati knjige prema unesenim parametrima. Dostupni parametri su: ISBN, naziv knjige ili autora, godina izdanja, izdavač, minimalna ocjena. Također može odabrati opciju filtriranja prema nazivu, broju osvrta te prema najvišoj ocjeni.

Običan korisnik (Prijavljeni korisnik):

- **Može sve što i gost aplikacije**
- **Odjava iz aplikacije** – Korisnik se može odjaviti iz aplikacije te ostati u aplikacija kao gost.

- **Dodavanje/uklanjanje knjige iz vlastite kolekcije** – Prilikom pregleda detalja knjige, korisnik može „označiti“ knjigu i tako ju dodati u vlastitu kolekciju, na isti način korisnik može ukloniti knjigu iz kolekcije.
- **Dodavanje/ažuriranje/brisanje osvrta na knjigu** – Korisnik može dodati vlastiti osvrt na knjigu putem pogleda za detalje. Korisnik modificira svoj osvrt na pogledu za pregled svih svojih recenzija.
- **Pregled knjiga iz vlastite kolekcije** – Pregled knjiga iz vlastite kolekcije. Može ih filtrirati na isti način kao i sve knjige.
- **Pregled vlastitih osvrta na knjige** – Na zasebnom pogledu korisnik može vidjeti sve dane osvrte te ih filtrirati prema: nazivu knjige, ocjeni i statusu.
- **Slanje zahtjeva za dodavanje knjige** – Korisnik može kreirati zahtjev slanjem ISBN-a koji je dužine od točno 10 znakova. Može ga kreirati i slanjem naziva knjige te informacija. Moderator pregledava te zahtjeve.
- **Pregled/ažuriranje vlastitog profila** – Korisnik može mijenjati sve informacije osim korisničkog imena. Promjena e-maila zahtjeva unos potvrdnog koda dobivenog na adresu e-pošte. Sve promjene zahtijevaju unos lozinke.

Funkcionalnosti administracijske aplikacije, grupirane prema tipu korisnika su sljedeće:

Moderator:

- **Pregled knjiga** – Pregled svih knjiga iz baze
- **Pregled/ažuriranje/brisanje detalja knjige** – Odabirom postojeće knjige iz popisa, moderator može do pregledati postojeće informacije, ažurirati ih ili kompletno ukloniti knjigu iz baze.
- **Unos nove knjige** – Moderator dodaje novu knjigu unosom potrebnih informacije. Može dodati i sliku naslovnice knjige. Ukoliko nije dodana slika, sustav će sam pokušati dodati sliku naslovnice preko ISBN-a koja se dobiva putem OpenLibrary API-a
- **Pregled i brisanje osvrta na knjige** – Moderatoru se prikazuju sve knjige koje imaju recenziju. Moderator te recenzije pregledava i uklanja po potrebi. Uklanjanje rezultira slanje obavijesti putem e-maila korisniku koji je dodao tu recenziju.
- **Pregled zahtjeva i dodavanje nove knjige putem zahtjeva** – Pregled zahtjeva za dodavanjem knjige od korisnika. Moderator može i dodati knjigu putem tog zahtjeva. Tada se šalje REST poziv OpenLibrary API-u koji puni informacije o knjizi preko ISBN-a. Moderator može te informacije dodatno

modificirati. Dodavanje knjige preko zahtjeva rezultira slanje obavijest putem e-pošte svim korisnicima koji su poslali zahtjev s tim ISBN-om. Knjiga se može dodati putem zahtjeva s naslovom knjige na isti način.

- **Pregled i blokiranje korisnika** – Pregled svih korisnika aplikacije uz mogućnost blokiranja i aktiviranja računa. Blokirani korisnici ne mogu se prijaviti u aplikaciju.
- **Pregled statistike aplikacije** – Pogled sadrži nekoliko grafova koji služe kao okvirni prikaz stanja podataka u aplikaciji. Sadrži informacije o knjigama i korisnicima.
- **Pregled i ažuriranje vlastitih informacija –** Ažurirati se može samo ime, prezime, email i lozinka.

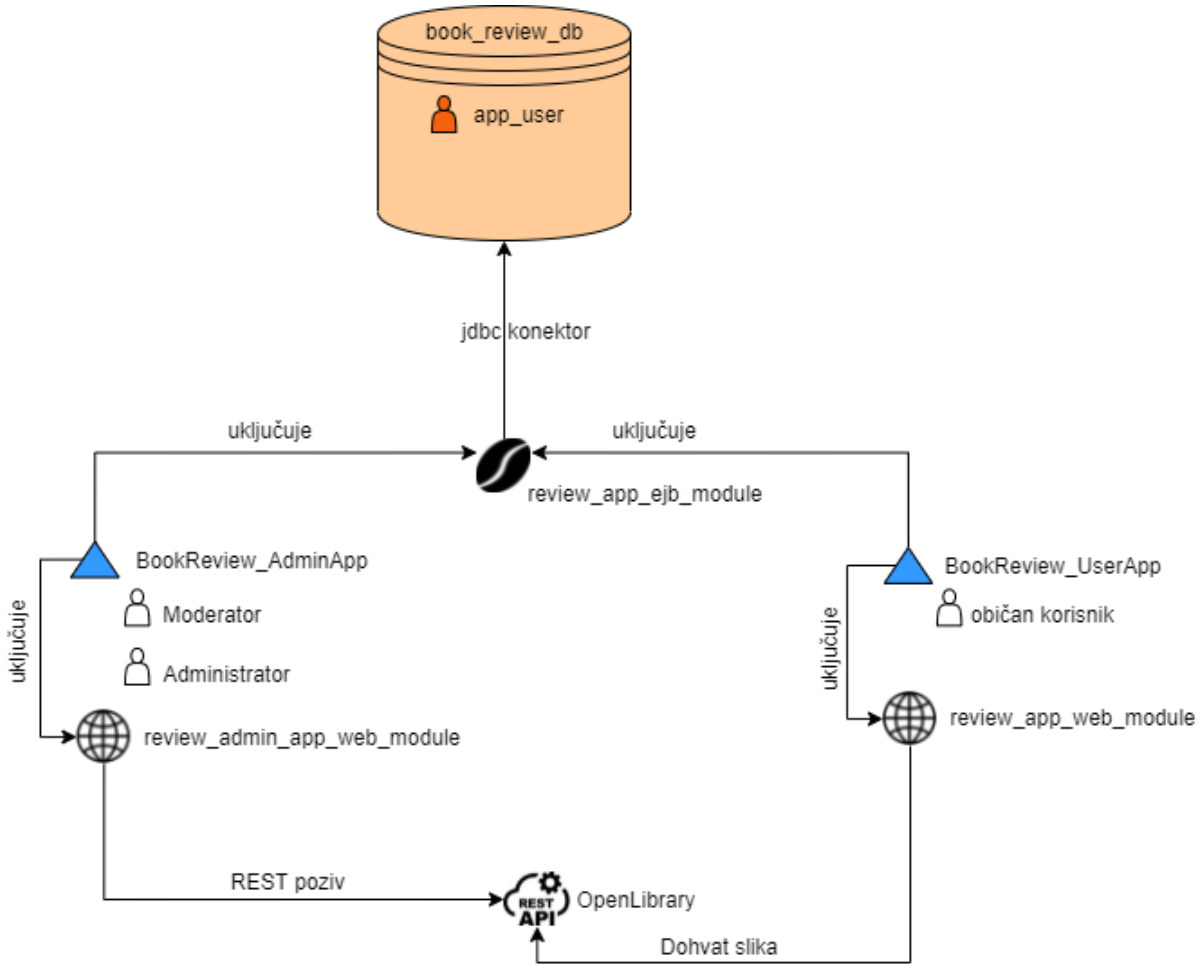
Administrator:

- **Može sve što i moderator**
- **Pregled i blokiranje moderatora** – Pregled svih moderatora aplikacije uz mogućnost blokiranja i aktiviranja računa. Blokirani moderatori ne mogu se prijaviti u aplikaciju.
- **Dodavanje novog moderatora i ažuriranje informacija** – Mogućnost kreiranja novog moderatora ili ažuriranja postojećeg.
- **Pregled dnevnika rada** – Daje uvid u radnje nad aplikacijom te zahtjevima koji su bili poslani.

Već je prijašnje spomenuto da je web aplikacija podijeljena na dvije aplikacija: korisničku i administratorsku. Te aplikacija građene su po modulima. Postoji ukupno tri modula i dvije aplikacija. Struktura web aplikacije je sljedeća:

- review_app_ejb_module (EJB modul) – Služi za dohvat podataka iz baze podataka te unos, ažuriranje i brisanje podataka iz iste baze. Sadrži modele podataka koji odgovaraju tablicama i njihovi atributima iz baze podataka. Sadrži sučelja (fasade) preko kojih se pristupa do metoda za generiranja upita koji dohvaćaju podatke iz baze. Za generiranje upita koristi se Criteria API tehnologija.
- review_app_web_module (Web modul, upakiran u WAR) – Sadrži korisničko sučelje i kontroler za preusmjeravanje podataka između pogleda i EJB modula. Koristi metode definirane sučeljem u EJB modulu kako bi dobio podatke iz baze. Te podatke oblikuje po potrebni i koristi ih u HTML dokumentu koji web preglednik razumije. Ovaj modul specifično se koristi kod korisničke aplikacije.

- review_admin_app_web_module (Web modul, upakiran u WAR) – Radi isto što i gornji modul samo što se ovaj koristi eksplicitno u administratorskoj aplikaciji.
- BookReview_AdminApp (poslovna aplikacija) – Pakira EJB modul i administratorski war modul u jednu aplikaciju.
- BookReview_UserApp (poslovna aplikacija) - Pakira EJB modul i korisnički WAR modul u jednu aplikaciju.



Slika 13: Arhitektura web aplikacije

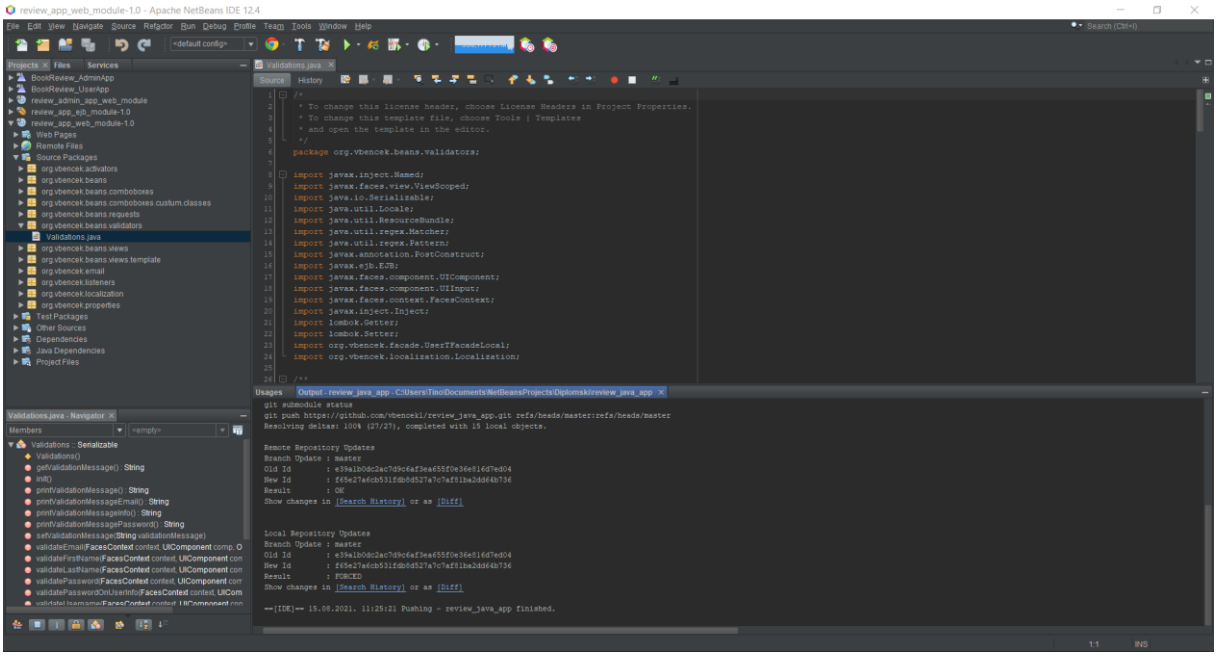
Kratko objašnjenje slike: Obje aplikacije koriste zajednički EJB modul za dohvat podataka iz baze. Taj modul konfiguriran je na način da pristupa podacima baze koristeći JDBC konektor koji se spaja na bazu *book_review_db* i koristi *db usera* pod nazivom *app_user.* Taj modul u sebi sadrži niz metoda koja generiraju SQL upit i šalju ga na bazu. Tim metodama pristupa se preko fasada (sučelja). EJB modul arhitekturi daje Model podataka. Svaka aplikacija ima i svoj Web modul. Taj modul daje kontrolere i poglede. Koristeći metode pružene od strane EJB modula, generira pogled za korisnika. Aplikacija koristi i vanjski API pod nazivom OpenLibrary. Administracijski web modul generira i šalje REST poziv za dohvat podataka o pojedinoj knjizi. Dohvaća se podaci o autoru, ali trenutno samo njegov naziv.

Korisnički web modul koristi OpenLibrary API opciju za dohvat slika naslovnica pojedinih knjiga. Te slike se mogu direktno prikazivati putem <img> elementa u HTML jeziku. Ovdje je riječ o MVC arhitekturi gdje EJB modul predstavlja M (Model) a Web modul predstavlja VC (View i Controller). Detalji o tehnologiji nalaze se u sljedećem poglavlju.

6.2. Korištene tehnologije

Provjerene i pouzdane tehnologije uvelike olakšavaju proces izrade aplikacija. Prilikom izrade ove aplikacije, težilo se korištenju tehnologija otvorenog koda.

Izabrano razvojno okruženje je **Netbeans** verzije 12.4. Netbeans je razvojno okruženje otvorenog koda čiji izvorni kod je u vlasništvu Apache softverske zajednice. Licenciran je otvorenom Apache licencom što ga čini besplatnim za korištenje u komercijalne i ne komercijalne svrhe. Pruža potporu za jezike: Java, css, HTML, JavaScript, php i još mnoge druge što ga čini dobrim razvojnim okruženjem za ovaj projekt.



Slika 14: Netbeans razvojno okruženje

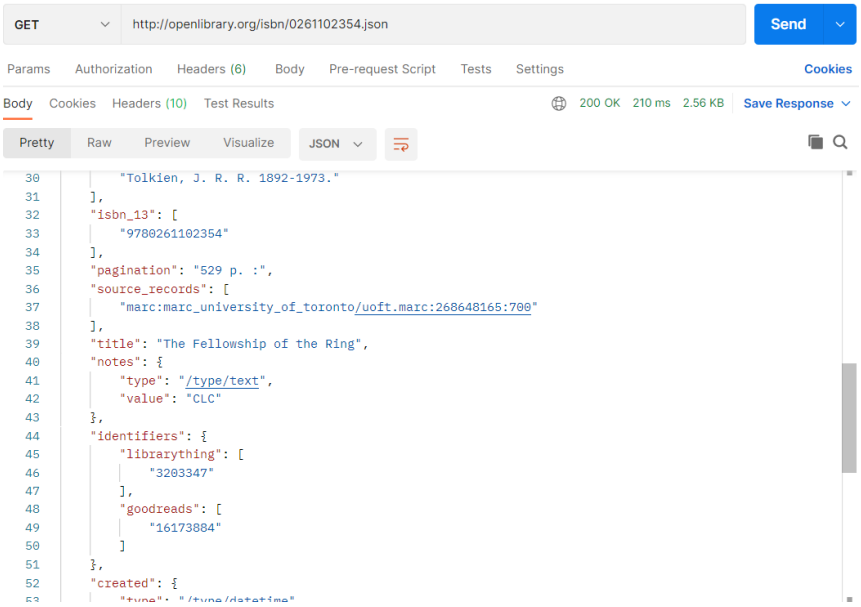
Programski jezik korišten za izradu aplikacije je Java. Osim Jave, korišteni su još i HTML, css i JavaScript te SQL jezik na bazi. Zbog promjene politike Oracle-a koji distribuira verzije Jave gdje Java više nije u potpunosti besplatna, koristio se JDK (Java development kit) verzije 8 dostupan na stranicama OpenLogic-a. Dakle, korištena verzija jave je **OpenJDK 8** dostupan na stranici: https://www.openlogic.com/openjdk-downloads.

Za potrebe testiranja i spajanja aplikacije na bazu, koristio se **JDBC derby** konektor. On omogućava spajanje na **JavaDB** bazu koja je u ovom slučaju lokalno podignuta.

Aplikacije se izvodi na **Glassfish 5.1** poslužitelju koji je također podignut lokalno. Glassfish je aplikacijski poslužitelj otvorenog koda koji je trenutno u vlasništvu Eclipse zajednice. Trenutno je licenciran preko dvije licence: EPL i GPL. Pošto Glassfish „koristi" Jakarta EE platformu, daje podršku za JSF, JSP servlete i dr. što omogućuje kreiranje poslovnih aplikacija koje su skalabilne i portabilne.

Za potrebe izrade korisničkog sučelja koriste se dva okvira: **JSF 2.3** i **Primefaces 10.0**. Oba okvira su otvorenog koda i koriste MVC arhitekturu. JSF licenciran je pod Apache licencom dok su Primefaces licencirani putem MIT licence. Ti okviri služe za izradu korisničkog sučelja korištenjem ponovo iskoristivih komponenti koje je moguće povezati s Java kodom preko „pozadinskog zrna" (engl. Backing bean). Samim time omogućuju kreiranje dinamičke web aplikacije s interaktivnim sadržajem. Za korištenje okvira unutar Maven aplikacije, potrebno je dodati ovisnosti na okvire. Za Primefaces je potrebno također uključiti i repozitorij koji se nalazi na url-u: http://repository.primefaces.org/ preko kojeg Primefaces uzima potrebne komponente.

Glavni servis korišten u aplikacije je **OpenLibrary API**. Servis je spomenut već par pura kroz rad, te je već poznato koja mu je svrha. API je otvorenog koda te se cijeli izvorni koda nalazi na repozitoriju: https://github.com/internetarchive/openlibrary . Za potrebe ovog rada, s OpenLibrary-a preuzimaju se podaci o knjigama, autorima te slikama naslovnice knjiga. Slanjem REST poziva preko GET metode dobiva se JSON format zapisa koji se kasnije oblikuje u zapis čitljiv aplikaciji odnosno u pojedini model podatka koji aplikacija koristi. Za dohvaćanje informacija o knjizi, potrebno je poslati GET zahtjev s ISBN-om knjige na putanju „isbn" s nastavkom „.json". Primjer takvog poziva nalazi se na donjoj slici. Za dohvaćanje autora, šalje se njegova identifikacija na putanju „authors". Naslovnica knjige može se dohvati na više načina. Jedan način je da se na putanju http://covers.openlibrary.org/b/isbn/{isbn}-{veličina}.jpg pošalje valjan ISBN te veličina naslovnice koja može biti: S, M ili L.



Slika 15: Odgovor OL API-a korištenjem Postmana

6.2.1. Biblioteke

Manje poglavlje u kojem su pobrojene bitne vanjske biblioteke koje su korištene prilikom izrade ovog rada. Svakoj biblioteci pridijeljena je svrha i korisnost u kodu.

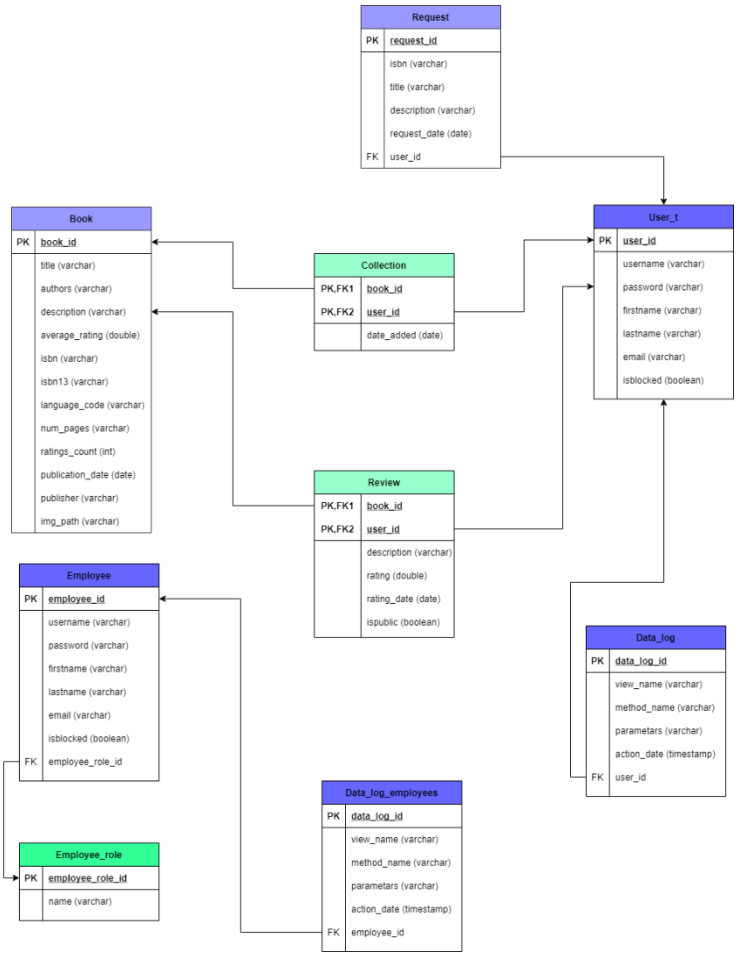
- **javax.mail-api 1.6.2** (javax.mail) – primarno služi za slanje e-pošte na navedenu adresu. Koristi se na više mjesta u kodu: za slanje potvrdnog koda prilikom registracije i zaboravljene lozinke te za slanje obavijest korisniku prilikom određenih radnji administratora/moderatora. Zahtjeva postavljanje vlastitog smtp poslužitelja, no može se koristiti i google-ov račun s njihovim poslužiteljom (smtp.gmail.com:587). Biblioteka je licencirana pod CDDL i GPL v2 licencom.
- **spring-security-crypto** (org.springframework.security) – služi za implementaciju sigurnosti aplikacije. Za potrebe izrađene aplikacije koristi se samo **PbkdfPasswordEncoder** kao način enkripcije lozinke koja se sprema u bazu. Na taj način izvorni tekst lozinke je nepoznat, odnosno zna ga samo korisnik. Biblioteka koristi Apache 2.0 licencu.
- **jackson-databind** (com.fasterxml.jackson.core) – primarno služi za manipulaciju JSON objektima. Projekt je inače otvorenog koda i nalazi se na javnom repozitoriju. Biblioteka je važan dio aplikacije jer omogućuje dohvaćanje podataka iz JSON-a koji se dobiva kao rezultat REST poziva. Također, koristi se i u drugim dijelovima aplikacije gdje je potrebno „čitati“ i kreirati JSON objekt.

- **Openpdf** (com.github.librepdf) – služi za generiranje pdf dokumenta radi čega se i koristi u aplikaciji. Projekt je otvorenog koda i također se nalazi na javnom repozitoriju. Omogućuje generiranje PDF-a preko Java koda što predstavlja idealno rješenje za ovaj projekt. Projekt je licenciran pod MPL i LGPL licencom.

Osim navedenih biblioteka, korištene su još i mnoge druge koje nisu navedene jer su dobro poznate. Radi se o bibliotekama za pružanje web servisa, za kreiranje klase REST servisa, spajanja na bazu i slično.

6.3. Baza podataka

Tablice baze podataka i veze između tih tablica prikazane su ERA modelom (donja slika). Za svaku tablicu kreiranja je SQL skripta koja generira tablice i puni ih testnim podacima. Baza trenutno sadrži 9 tablica od kojih je devet glavnih tablica, jedna pomoćna i dvije tablice služe za ostvarenje veze više na više. Upute za rekonstrukciju baze podataka nalaze se u mapi „Database“ na repozitoriju ove aplikacije.



Slika 16: ERA model

Glavna tablica modela je „Book“. Ona sadrži informacije o pojedinoj knjizi. U tablicu „User_t“ pohranjuju se podaci o običnom korisniku (korisnička aplikacija), dok se s druge strane, u tablicu „Employee“ pohranjuju podaci o moderatorima i administratorima (administratorska aplikacija). Tablica „Employee_role“ sadrži uloge za administratorsku aplikaciju. Tablica „Collection“ ostvaruje vezu više na više i povezuje knjigu i korisnika. Služi za pamćenje knjiga koje je korisnik dodao u svoju kolekciju. Postoji i tablica „Review“ koja također ostvaruje vezu više na više i povezuje knjigu i korisnika. Služi za spremanje osvrta koje je pojedini korisnik dao na knjigu. Tablica „Request“ sadrži kreiranje zahtjeve za dodavanje knjiga od strane korisnika. Na kraju, postoje i tablice za implementaciju dnevnika rada. To su tablice: „Data_log“ i „Data_user_log“ za korisnike obje vrste aplikacija.

6.4. Aplikacijska logika i izgled korisničkog sučelja

Poglavlje se bavi opisom razvijene web aplikacije. Prikazani su pogledi te logika koja stoji iza tih pogleda. Ta logika uključuje: pozadinska zrna (engl. Backing bean) koja spajaju frontend s backend-om, modele podataka – preslikani iz tablica baze podataka te fasade i implementacije metoda za dohvat podataka iz baze. Radi preglednosti poglavlje je dodatno podijeljeno na tri djela: korisnička aplikacija, administratorska aplikacija te web servis. Prikazane su i opisane glavne značajke navedenih programskih proizvoda.

6.4.1. Korisnička aplikacija

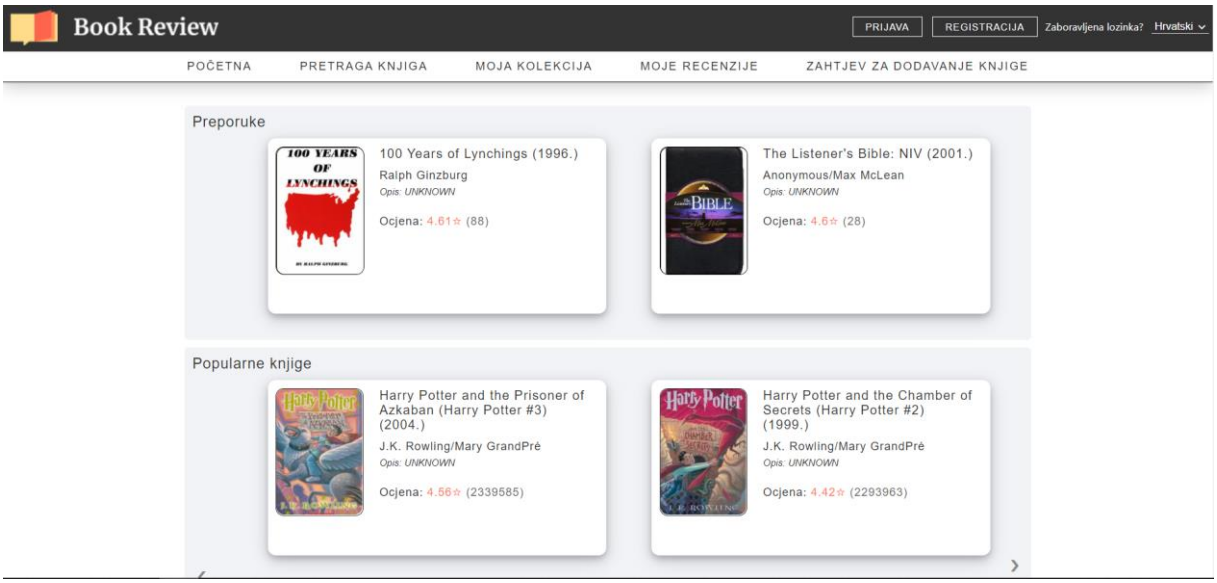
Aplikacija sadrži elemente koji se ponavljaju na svakoj stranici. Ti elementi su: zaglavlje(engl. header) i podnožje (engl. Footer). Zaglavlje sadrži logo, prijavu, odjavu, registraciju, promjenu jezika te navigaciju. Također sadrži informaciju o prijavljenom korisniku. Radi smanjenja posla i ponove iskoristivosti koda napravljen je predložak (engl. Template) koji sadrži te elemente. Svaka stranica kreirana je iz tog predloška. Donja slika prikazuje samo <head> element tog predloška koji uključuje css i JS datoteke.

```
<h:head>
      <meta http-equiv="Content-Type" content="text/html;
      charset=UTF-8" />
      <h:outputStylesheet library="css" name="cssTemplate.css"/>
      <h:outputStylesheet library="css" name="cssPages.css"/>
      <script
      src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.
      min.js"></script>
      <script type="text/javascript"
      src="resources/js/TemplateFunctions.js"></script>
```

```
        <script type="text/javascript"
        src="resources/js/IndexJS.js"></script>
         <title><ui:insert name="title">Title</ui:insert></title>
         <link rel="shortcut icon"
        href="resources/images/title/title_logo_41Q_icon.ico"/>
    </h:head>
```

6.4.1.1. Početna stranica, registracija, zaboravljena lozinka i lokalizacija

Početna stranica prikazuje knjige grupirane u tri elementa. Prvi elementi prikazuje preporučene knjige. Srednji sloj dohvaća podatke iz baze na način da zahtjeva dohvat knjiga s minimalnom ocjenom četiri. Za prijavljenog korisnika još se dodatno filtrira po broju osvrta. Dugi element prikazuje popularne knjige. Ovdje se dohvaćaju šest knjiga koje imaju najveći broj osvrta. Zadnji element odnosi se na posljednje ocijenjene knjige. Korisnik tamo vidi knjige koje su nedavno dobile osvrte. Klikom na bilo koju od knjiga otvara se detaljni prikaz te knjige.



Slika 17: Početna stranica korisničke aplikacije

Elementi stranice definirani su u xhtml datoteci. Radi se o JSF elementima koji se povezuju s backend-om pomoću zrna. Svaka stranica ima svoje zrno na kojeg se referencira pomoću imena. U slučaju početne stranice, zrno je imenovano s *viewHome* i notirano je s *@ViewScoped* anotacijom. Postoji veći broj anotacija koje se mogu koristiti za zrna. Kod ove aplikacije koriste se 4 vrste anotacija: *Request*, *View*, *Session* i *Application*. Svaka anotacija ima svoju svrhu. *RequestScope* pamti podatke za vrijeme obavljanja jednog zahtjeva, *ViewScope* pamti podatke za vrijeme trajanja jednog pogleda (do osvježavanja stranice), *SessionScope*, kao što sama riječ govori, pamti podatke za vrijeme trajanja jedne sesije a *ApplicationScope* sadrži podatke koji su jednako za sve korisnike. Zrna sadrže metode za dohvat podataka preko EJB-a koji pristupa bazi. Zrna zatim povezuju java objekte s

elementima HTML-a. Donja metoda dohvaća listu knjiga preko fasade EJB-a – *BookFacade*. Funkcija *findBooksByCriteria* konstruira SQL upit kojeg postavlja na bazu.

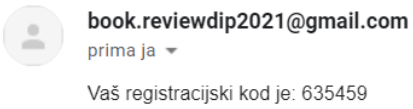
```
@EJB(beanName = "BookFacade")
BookFacadeLocal bookFacade;
…
public List<Book> getPopularbooks(int startingFrom, int totalNumber)
{
        return bookFacade.findBooksByCriteria("", "", 0, "", 0,
"reviews", startingFrom, totalNumber);
    }
```

Donji programski isječak prikazuje način na HTML dobiva podatke od zrna. Koristi se *html repeater* za generiranje elementa koji sadrže knjige. Taj element dobiva listu knjiga od zrna. Varijabla *row* zapravo je tipa *Book* iz modela i time omogućuje dohvaćanje atributa knjige.

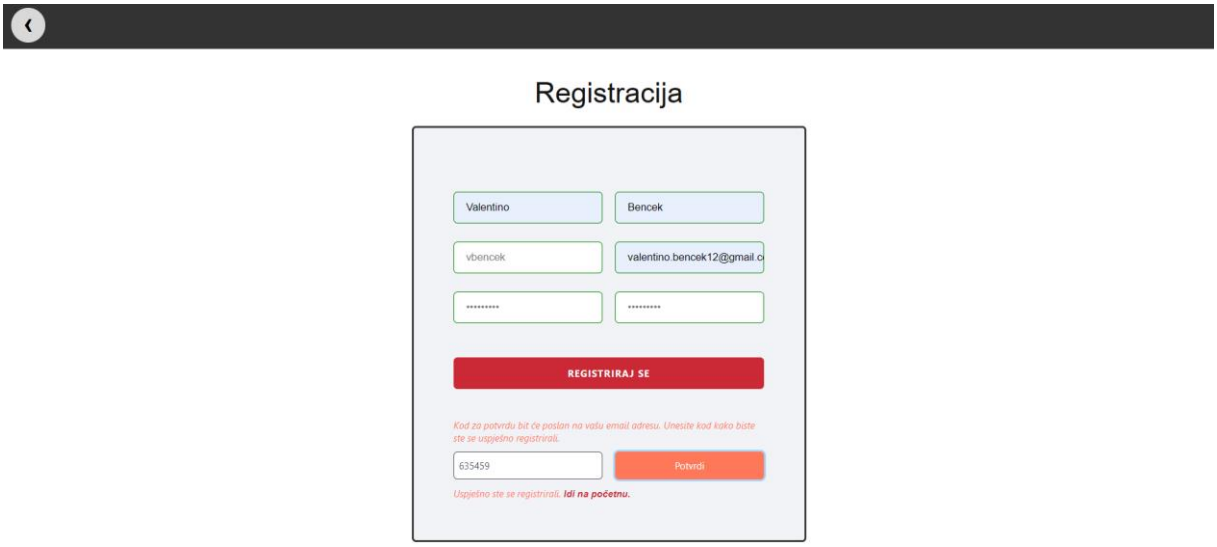
```
<ui:repeat value="#{viewHome.getPopularbooks(4, 2)}" var="row">
<h:outputLink value="#{viewHome.redirectToBookDetails(row.bookId)}">
<div class="div-book-container">
<div class="div-book-image-container">
<img src="#{viewHome.setIMG(row)}"
alt="#{viewHome.makeTextShorter(row.title, 100)}"/>
</div>
<h:outputLabel class="div-book-header"
value="#{viewHome.makeTextShorter(row.title, 100)}
(#{viewHome.convertDateToYear(row.publicationDate)}.)"/>


<h:outputLabel class="div-book-details"
value="#{row.authors}"/>


<h:outputText class="div-book-description"
value="#{l['viewHome.book.description']}: #{row.description}"/>
<div class="div-book-grade-text">
<h:outputLabel class="div-book-grade-text-gradeText"
value="#{l['viewHome.book.rating']}:"/>
<h:outputLabel class="div-book-grade-text-gradeValue" value="
#{genFunctions.roundDouble(row.averageRating)}&#9734; "/>
<h:outputLabel class="div-book-grade-text-NOratings"
value="(#{row.ratingsCount})"/>
</div></div></h:outputLink></ui:repeat>
```

Registracija je jedna od kompleksnijih formi aplikacije. Zahtjeva kreiranje vlastite validacije, slanje emaila te provjeru potvrdnog koda. Forma funkcionira na slijedeći način: korisnik prvo mora ispravno popuniti sva polja. Postoje vlastiti implementirani Java validatori

koji osiguravaju da su polja ispravno popunjena (duljina, duplicirana korisnička imena, format e-maila i slično). Nakon ispravnog popunjena polja, korisnik na uneseni email prima potvrdni kod kojeg unaša u aplikaciju. Registracija je uspješna ukoliko je kod ispravan.



Slika 18: Potvrdni email



Slika 19: Korisnička registracija

Donji kod prikazuje dva primjera vlastito – implementiranih validatora. Prva funkcija provjerava postojanje korisničkog imena u bazi. Također provjerava da li je uneseni String ispravne duljine. Druga metoda odnosi se na provjeru ispravnosti e-maila. E-mail mora biti unikatni (kao i korisničko ime) te mora biti odgovarajućeg formata.

```
public void validateUsername(FacesContext context, UIComponent comp,
Object value) {
        msgUsername="";
        String validString = (String) value;
        if (validString.length() < 4 || validString.length() > 16) {
            System.out.println("validations: invalid username");
            ((UIInput) comp).setValid(false);
            msgUsername += "*" + res.getString
("validations.validateUsername") + "\n";
        }else if(userTFacade.findUserByUsername(validString)!=null){
            System.out.println("validations: username already
exists");
```

```
            ((UIInput) comp).setValid(false);
            msgUsername = "*" +
res.getString("validations.validateUsernameExists") + "\n";
        }
    }
public void validateEmail(FacesContext context, UIComponent comp,
Object value) {
        msgEmail="";
        String validString = (String) value;
        Pattern pattern = Pattern.compile("^[A-Z0-9._%+-]+@[A-Z0-9.-
]+\\.[A-Z]{2,6}$", Pattern.CASE_INSENSITIVE);
        Matcher match = pattern.matcher(validString);
        boolean checkValid = match.matches();
        if (!checkValid) {
            System.out.println("validations: invalid Email");
            ((UIInput) comp).setValid(false);
            msgEmail = "*" +
res.getString("validations.validateEmail") + "\n";
        }else if(userTFacade.checkIfEmailExist(validString)){
            System.out.println("validations: email already exists");
            ((UIInput) comp).setValid(false);
            msgEmail = "*" +
res.getString("validations.validateEmailExists") + "\n";
        }
    }
```

Nakon uspješne validacije, korisnik se kreira i unosi u bazu podataka koristeći EJB fasadu. Sam unos obavlja EJB.

```
@EJB(beanName = "UserTFacade")
UserTFacadeLocal userTFacade;
…
private void RegisterUser() {
        UserT user = new UserT();
        String hashPassword = pbkdf2PasswordEncoder.encode(password);
        user.setPassword(hashPassword);
        user.setUsername(username);
        user.setFirstname(firstName);
        user.setLastname(lastName);
        user.setEmail(email);
        user.setIsblocked(false);
        userTFacade.create(user);
        loginUser(user);
```

```
    }
```

Donja metoda šalje poruku na adresu e-pošte. Koristi postavke iz konfiguracijske datoteke za postavljanje sesije i korisnika koji šalje poruku. Taj korisnik zapravo je aplikacija.

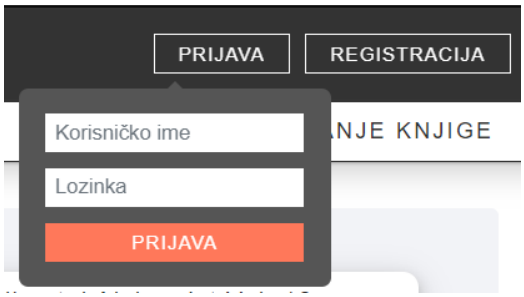
```
public void sendMessage(String emailTo, String subject, String text) {
        setProperties();
        try {
            Message message = new MimeMessage(session);
            message.setFrom(new InternetAddress(username));
            message.setRecipients(Message.RecipientType.TO,
                    InternetAddress.parse(emailTo));
            message.setSubject(subject);
            message.setText(text);
            Transport.send(message);
            System.out.println("Email Sender: Message sent!");
        } catch (MessagingException e) {
            throw new RuntimeException(e);
        }
    }
```

Postoji i mogućnost obnove lozinke ukoliko ju korisnik zaboravi. Obnova lozinke zahtjeva poznavanje vlastitog korisničkog imena i adrese e-pošte. Kad su uvjeti zadovoljeni, korisnik prima novo generiranu lozinku na e-mail. Lozinka je nasumično generirana pa ju korisnik može promijeniti nakon prijave.

Slika 20: Zaboravljena lozinka

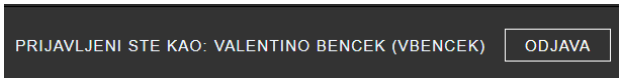
6.4.1.2. Prijava i odjava iz aplikacije

Prijava u aplikaciju realizirana je putem malog „pop-up“ prozora. Prijaviti se mogu korisnici s valjanim imenom, koji znaju vlastitu lozinku i nisu blokirani. Aplikacija vraća odgovarajuće poruke ako prijava nije uspješna.



Slika 21: Prijava u aplikaciju

Nakon uspješne prijave, korisnik vidi vlastito korisničko ime te ime i prezime s kojim se registrirao. Pogled korisnika se renderira pa korisnik više ne vidi gumbe za prijavu i registraciju.



Slika 22: Odjava iz aplikacije

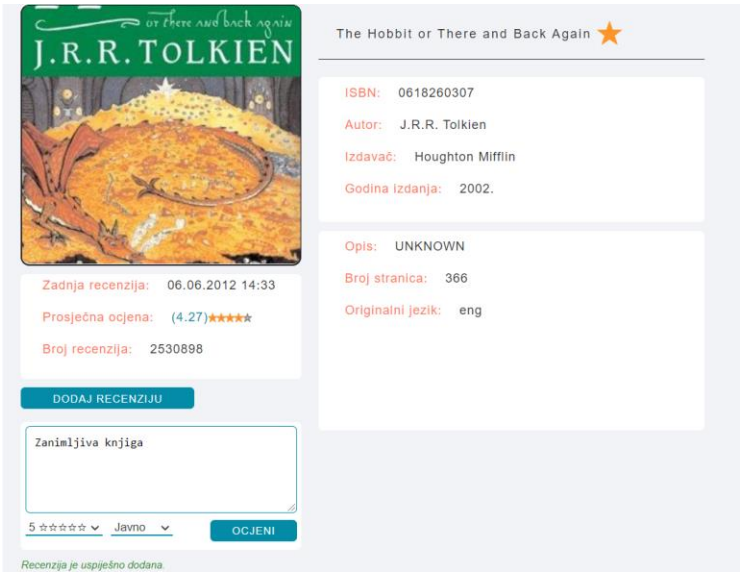
Prijavom i odjavom bazi se *ActiveUserSession* zrno anotirano s *@SessionScope.* U tu klasu sprema se aktivan korisnik. Preko te klase ga je moguće dohvati s bilo koje lokacije u kodu pomoću Injektiranja. Donji programski isječak prikazuje metodu za autentifikaciju. Provjera se da li je aktivni korisnik zabilježen, ukoliko nije, kreće se s verifikacijom. Nakon ispravno unesenih podataka, aktivni korisnik se postavlja te se okida metoda *renderLogin()* koja je ništa drugo neko metoda za promjenu pogleda koji vidi aktivni korisnik. S druge strane, ukoliko aktivni korisnik postoji znači da se radi o odjavi te se okida funkcija *renderLogout()* koja postavlja pogled za odjavljenog korisnika.

```
public void authenticate() {
        if (activeUser == null) {
            // Active user doesn't exist. Login
            UserT tempUser =
userTFacade.findUserByUsername(username);
            if (tempUser != null) {
                Pbkdf2PasswordEncoder encoder = new
Pbkdf2PasswordEncoder();
                boolean passMatch = encoder.matches(password,
tempUser.getPassword());
                boolean isBlocked = tempUser.getIsblocked();
                if (passMatch && !isBlocked) {
                    activeUser = tempUser;
                    renderLogin();
                } else if (!passMatch) {
                    failedLoginMsg =
res.getString("template.login.wrongPass");
```

```
                } else if (isBlocked) {
                    failedLoginMsg =
res.getString("template.login.blockedUser");
                }
            } else {
                failedLoginMsg =
res.getString("template.login.wrongUsername");
            }
        } else {
            // Active user exists. Logout
            renderLogout();
            activeUser = null; }}
```

6.4.1.3. Pogled detalja knjige, kreiranje osvrta

Vrši se dohvat slike s OpenLibrary-a te dohvat podataka iz baze. Zvijezda ispunjena žutom bojom označuje da je aplikacija dodana u vlastitu kolekciju. Podaci nisu 100% dosljedni jer je dio podataka očitan preko skupa podataka (engl. Dataset) (URL: www.kaggle.com/ jealousleopard/ goodreadsbooks) pa broj recenzija i prosječna ocjena ne odgovaraju u potpunost pravoj slici. Recenzija se dodaje putem manje forme gdje korisnik bira hoće li ona biti objavljena (Opcija: javno) ili će biti prikazana samo korisniku (Opcija: privatno)



Slika 23: Detalji knjige

Donja metoda postavlja sliku. Slika se inicijalno dohvaća preko putanje spremljene u bazi, ukoliko putanja ne postoji, tada se dohvaća preko OpenLibrary-a koristeći ISBN.

```
public String setIMG() {
        if (thisBook != null) {
            if (thisBook.getImgPath() != null) {
```

```
                return thisBook.getImgPath();
            } else {
                return "http://covers.openlibrary.org/b/isbn/" +
thisBook.getIsbn() + "-L.jpg";
            }}return "";}
```

Dodavanjem recenzije potrebno je napraviti ažuriranje nad tablicom *Book*. Treba povećati/smanjiti prosječnu ocjenu kao i broj ocjena. Preračunavanje radi Zasebna *RequestScope* klasa koja sadrži metodu za preračunavanje prosječne ocjene. Ovisno o proslijeđenoj akciji, metoda vrši izračune. Ta metoda poziva se unutar metode *updateBook*, koja poziva EJB metodu i stvarno ažurira knjigu u bazi.

```
private double calculateRating(Review review, String action, double
oldRating){
        double result=0;
        Book book = review.getBook();
        int ratingCount=book.getRatingsCount();
        double averageRating=book.getAverageRating();
        double rating = review.getRating();
        double totalRatingSum=ratingCount*averageRating;
        if(action.equals("INSERT")){
            int newRatingCount=ratingCount+1;
            result=(totalRatingSum+rating)/newRatingCount;
        }else if(action.equals("UPDATE")){
            result=(totalRatingSum-oldRating+rating)/ratingCount;
        }else if(action.equals("DELETE")){
            int newRatingCount=ratingCount-1;
            result=(totalRatingSum-rating)/newRatingCount;
        }
        return result;}
```

Recenzije pojedine knjige lako je dohvatiti jer se putem modela vežu na knjigu. Unutar baze povezane su vanjskim ključem te se ta veze prenosi i na model.

```
@Entity
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Book.findAll", query = "SELECT b FROM Book
b")})
public class Book implements Serializable {
…
@OneToMany(cascade = CascadeType.ALL, mappedBy = "book")
private List<Collection> collectionList;
```

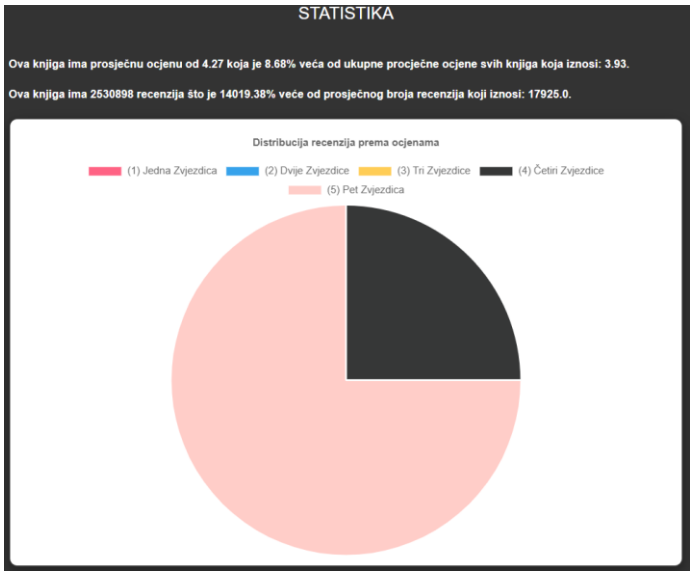
```
…
@XmlTransient
public List<Collection> getCollectionList() {
        return collectionList;}
```

Napravljena je i *custom* metoda za dohvat recenzija koja se također može koristiti. Ona je ipak pogodnija za pretragu recenzija pa se koristi za istu. Njen potpis nalazi se u donjem kodu.

```
List<Review> findReviewsByCriteria(Book book, UserT userT, double
minimumAvgRating, String sortOption, boolean hasDescription, boolean
isPublic, int offset, int limit);
```

Slika 24: Osvrti na knjigu

Već prije u poglavlju spomenuto je kako se primefaces koristi kod administratorske aplikacije. Primefaces se koristi i u korisničkoj aplikaciji, ali samo za generiranje grafa. Graf prikazuje raspon ocjena koje su pridijeljene pojedinoj knjizi.



Slika 25: Statistika knjige

Graf se generira slijedećim kodom:

```
public PieChartModel createBookReviewsPieModel(Book book){
    PieChartModel pieBookReviews=new PieChartModel();

    ChartData data = new ChartData();
    PieChartDataSet dataSet = new PieChartDataSet();

    List<Number> ratingNumber = new ArrayList<>();
    ratingNumber.add(reviewFacade.countRatingScore(book, 1));
    ratingNumber.add(reviewFacade.countRatingScore(book, 2));
    ratingNumber.add(reviewFacade.countRatingScore(book, 3));
    ratingNumber.add(reviewFacade.countRatingScore(book, 4));
    ratingNumber.add(reviewFacade.countRatingScore(book, 5));
    List<String> ratingValue = new ArrayList<>();
    ratingValue.add(res.getString("combobox.rating.one"));
    ratingValue.add(res.getString("combobox.rating.two"));
    ratingValue.add(res.getString("combobox.rating.three"));
    ratingValue.add(res.getString("combobox.rating.four"));
    ratingValue.add(res.getString("combobox.rating.five"));

    dataSet.setData(ratingNumber);
    dataSet.setBackgroundColor(getPieChartColors());
    data.addChartDataSet(dataSet);
    data.setLabels(ratingValue);
    pieBookReviews.setData(data);

    PieChartOptions options = new PieChartOptions();
    Title title = new Title();
    title.setDisplay(true);
    title.setText(res.getString("ViewBookStatistic.pie.title"));
    options.setTitle(title);

    pieBookReviews.setOptions(options);

    return pieBookReviews;
}
```

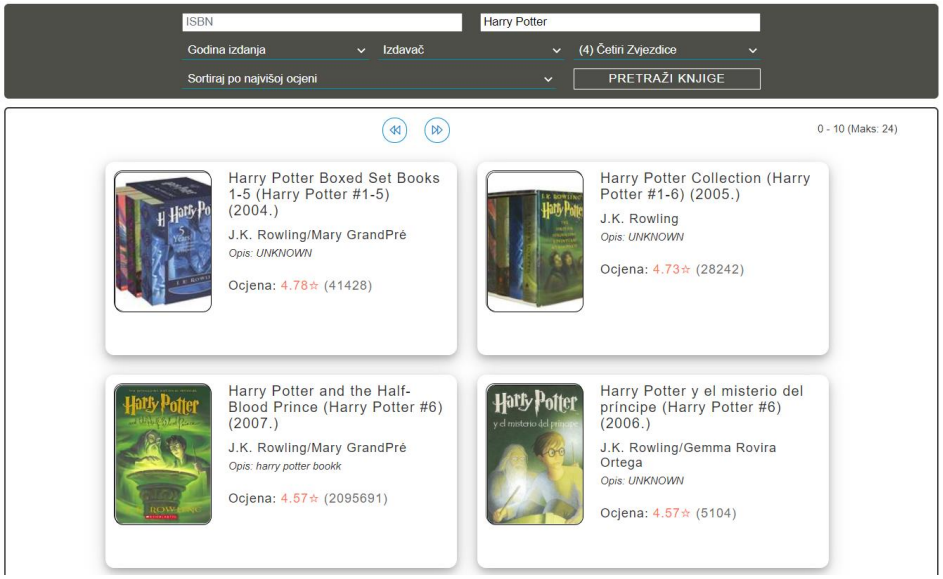
Za prikaz grafa na korisničkom sučelju, potrebno je dodati xhtml element na slijedeći način:

```
<p:pieChart model=
"#{viewBookStatistic.createBookReviewsPieModel(viewBookDetails.thisBook)}"
style="width: 100%; height: 500px;"/>
```

6.4.1.4. Pretraga knjiga, vlastitih knjiga i vlastitih recenzija

Stranice za pretragu sastoje se od dva glavna elementa: forme za unos ključnih riječi (tražilica) i forme s rezultatima. Rad je temeljen na jednostavnom principu: pritiskom na gumb pretrage, parametri tražilice prenose se na URL stranice. Kada se pogled konstruira (anotacija @PostConstruct), parametri se preuzimaju iz URL i prosljeđuju u metodu za dohvat podataka. Paginacija radi na istom principu gdje se broj stranice prenosi u URL. To prikazuje donji kod.

```
private void setSearchParams() {
        Map<String, String> params =
FacesContext.getCurrentInstance().getExternalContext().getRequestPara
meterMap();
        isbn = params.get("ISBN") != null ? params.get("ISBN") : "";
        keyword = params.get("Keyword") != null ?
params.get("Keyword") : "";
        try {
            year = Integer.parseInt(params.get("Year"));
        } catch (Exception e) {
            year = 0;
        }
        publisher = params.get("Publisher") != null ?
params.get("Publisher") : "";
        defaultPublisher=publisher;
        try {
            minRating = Double.parseDouble(params.get("MinRating"));
        } catch (Exception e) {
            minRating = 0;
        }
        sortOption = params.get("SortBy");
        try {
            pageNum = Integer.parseInt(params.get("PageNum"));
        } catch (Exception e) {
            pageNum = 0;
        }setParamsForPagination();}
```



Slika 26: Pretraga knjiga

Za dohvat liste knjiga, korisničkih knjiga kao i osvrta, koriste se sljedeće metode iz EJB-a:

```
List<Book> findBooksByCriteria(String isbn, String keyword, int
publishYear, String publisher, double minimumAvgRating, String sortOption,
int offset, int limit);
```

```
List<Book> findUserBooksByCriteria(UserT userT,String isbn, String keyword,
int publishYear, String publisher, double minimumAvgRating,String
sortOption,int offset,int limit);
```

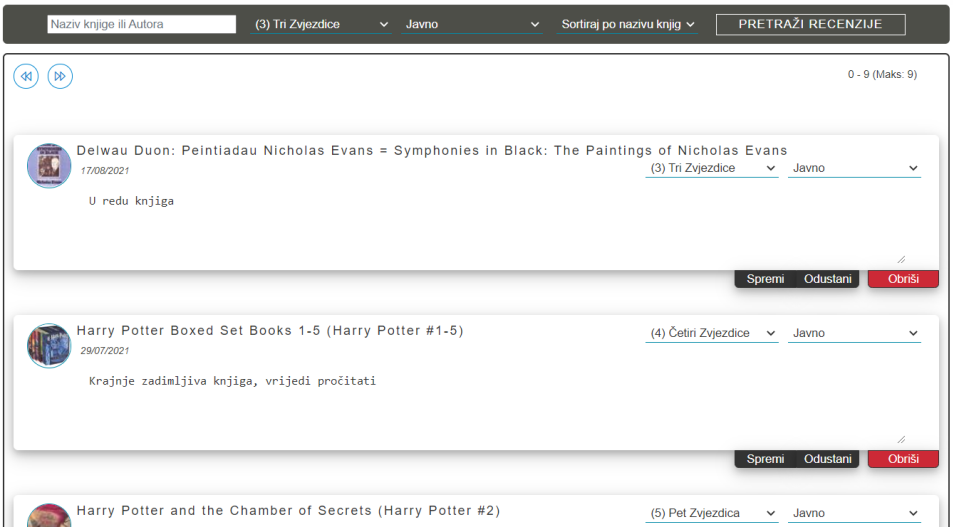
```
List<Review> findMyReviewsByCriteria(UserT userT,String keyword, double
minimumAvgRating, Boolean isPublic, String sortOption, int offset, int
limit);
```

Donji programski isječak prikazuje implementaciju metode *findBooksByCriteria* . Metoda koristi Criteria API tehnologiju za konstrukciju SQL upita. Također, neke atribute nije uvijek potrebno stavljati u „WHERE“ dio upita pa se zato prije provjerava da li je neki atribut prazan ili jednak *null*. Moguće je i izabrati opciju sortiranja u istom upitu. Trenutno je ta opcija „hardkodirana“ na tekst. U budućnosti bi se moglo razmisliti o uvađanju neke vrste enumeracije.

```
public List<Book> findBooksByCriteria(String isbn, String keyword,
int publishYear, String publisher, double minimumAvgRating, String
sortOption, int offset, int limit) {
        CriteriaBuilder cb = em.getCriteriaBuilder();
        CriteriaQuery<Book> cq = cb.createQuery(Book.class);
        Root<Book> book = cq.from(Book.class);
        List<Predicate> predicates = new ArrayList<Predicate>();
        if (isbn != null && !"".equals(isbn)) {
            predicates.add(cb.equal(book.get("isbn"), isbn));}
```

```
        if (keyword != null && !"".equals(keyword)) {
            Predicate title = cb.like(cb.upper(book.get("title")),
"%" + keyword.toUpperCase() + "%");
            Predicate authors =
cb.like(cb.upper(book.get("authors")), "%" + keyword.toUpperCase() +
"%");
            predicates.add(cb.or(title, authors)); }
        if (publishYear != 0) {
            predicates.add(cb.equal(cb.function("year",
Integer.class, book.get("publicationDate")), publishYear));
        if (publisher != null && !"".equals(publisher)) {
            predicates.add(cb.equal(book.get("publisher"),
publisher)); }
        if (minimumAvgRating != 0) {
predicates.add(cb.greaterThanOrEqualTo(book.get("averageRating"),
minimumAvgRating));}
        if (sortOption != null && !"".equals(sortOption)) {
            if ("title".equals(sortOption)) {
                cq.orderBy(cb.asc(book.get("title")));}
            if ("rating".equals(sortOption)) {
                cq.orderBy(cb.desc(book.get("averageRating")));}
            if ("reviews".equals(sortOption)) {
                cq.orderBy(cb.desc(book.get("ratingsCount")));}
            if ("date".equals(sortOption)) {
                 Root<Review> review = cq.from(Review.class);
                 predicates.add(cb.equal(book.get("bookId"),
review.get("book").get("bookId")));
                 cq.orderBy(cb.desc(review.get("ratingDate")));}}
        cq.select(book).where(predicates.toArray(new Predicate[]{}));
        //execute query and do something with result
        List<Book> results = em.createQuery(cq)
                .setFirstResult(offset)
                .setMaxResults(limit)
                .getResultList();
        return results;}
```

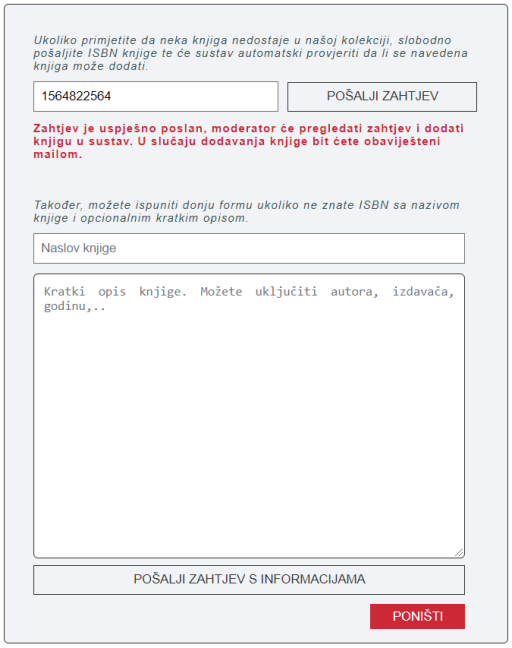
Donja slika prikazuje izgled pogleda za prikaz vlastitih recenzija. Radi po istom principu kao i pretraga knjiga, ali uz druge atribute. Ovdje je moguće i ukloniti vlastitu recenziju kao i ažurirati recenziju. Ažuriranje ili brisanje recenzije povlači pozivanje metode za ažuriranje podataka knjige metodom koja je bila spomenuta prije.



Slika 27: Pretraga recenzija

6.4.1.5. Zahtjev za dodavanje knjiga

Forma za podnošenje zahtjeva za dodavanje knjige sadrži dvije opcije. Prva opcija je slanje ISBN-a. U bazi se tada vrši provjera da li već postoji knjiga s istim ISBN-om. Ako ne postoji, korisnik može podnijeti zahtjev. Druga opcija je unos Naslova knjige. Korisnika se također potiče da ispuni određene informacije kako bih moderator imao što veći broj informacija na raspolaganju prilikom dodavanja same knjige.

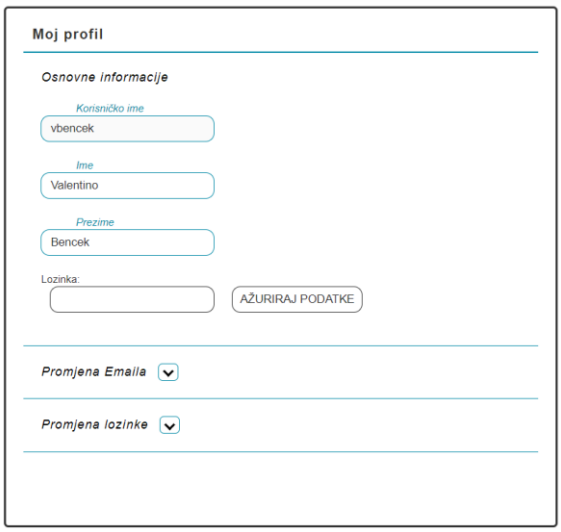


Slika 28: Zahtjev za dodavanjem knjiga

6.4.1.6. Pregled korisničkih informacija

Unutar forme za prikaz korisničkih informacija postoje tri različite forme za izmjenu podataka. Prva forma odnosi se na ažuriranje osnovnih podataka (točnije imena i prezimena). Druga forma ažurira email, a zadnja forma ažurira lozinku. Sve forme zahtijevaju unos lozinke kao obvezan parametar, dok promjena e-maila zahtjeva i ponovni unos potvrdnog koda kojeg korisnik ponovo dobiva na mail. Donji isječak koda prikazuje jednu metodu za ažuriranje informacija.

```
public void updateUserInfo() {
        resetMessages();
        Pbkdf2PasswordEncoder encoder = new Pbkdf2PasswordEncoder();
        if (encoder.matches(password, thisUser.getPassword())) {
            UserT updatedUser = thisUser;
            updatedUser.setFirstname(firstname);
            updatedUser.setLastname(lastname);
            userTFacade.edit(updatedUser);
            rendermessageInfoOK = true;
            messageInfo = res.getString("viewUserProfile.msg.updateData");

        } else {
            rendermessageInfoNotOK = true;
            messageInfo = res.getString("viewUserProfile.msg.invalidPass");
        }}
```



Slika 29: Korisnički profil

6.4.2. Administratorska aplikacija

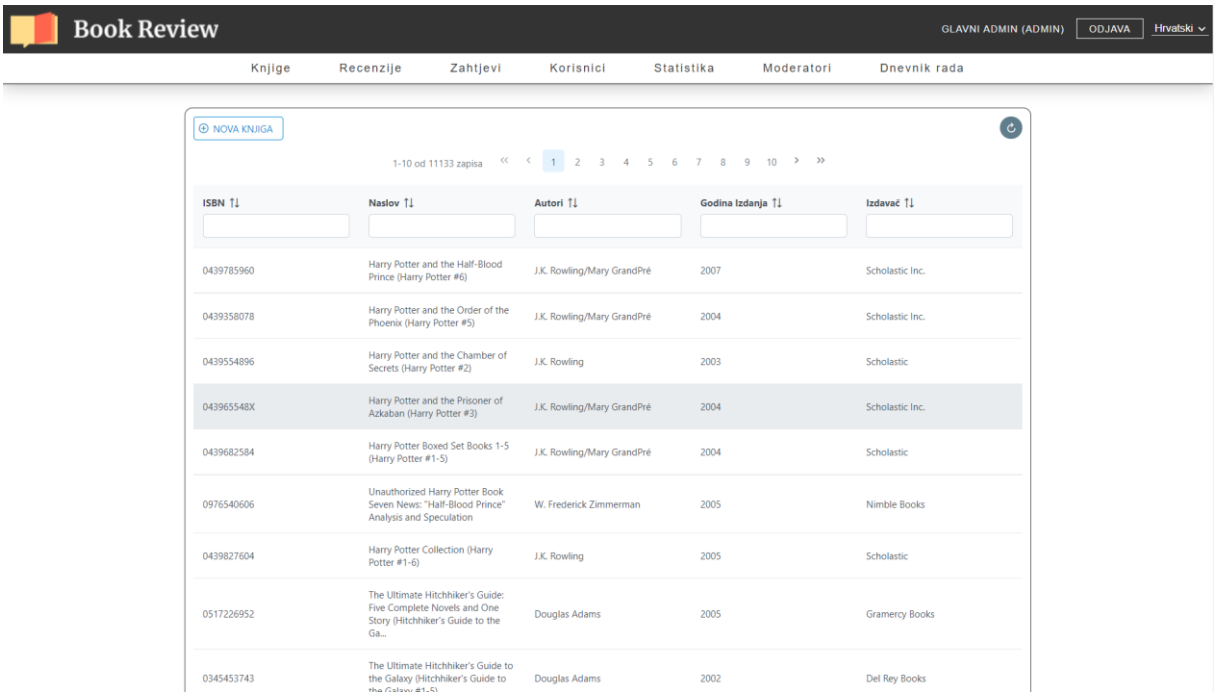
Administratorska aplikacija oslanja se na programski okvir Primefaces. Veliki dio dizajna navigacije, zaglavlja i podnožja preuzet je iz korisničke aplikacije. Također se koristi predložak za mogućnost ponovnog iskorištenja koda. Aplikacija poslužuje dva tipa korisnika: administratora i moderatora. Moderatoru je zabranjen pristup do određenih stranica. Točnije, do pogleda za dodavanje moderatora i dnevnika rada. Aplikacija nema mogućnost rada kao gost te zahtjeva obaveznu prijavu u sustav.

6.4.2.1. Početna stranica

Korisnik se mora prijaviti u aplikaciju. Prilikom prijave, dohvaća se njegov status i tip korisnika. Korisnik ne smije biti blokiran kako bi se mogao prijaviti. Nakon uspješne prijave korisnika se preusmjerava na početnu stranicu koja prikazuje sve knjige s mogućnosti filtriranja. Renderiranje pojedinih pogleda vrši se na način da se provjerava da li je aktivni korisnik ima ulogu moderatora. Logika iza prijave i odjave ista je kao i na korisničkoj aplikaciji.

Slika 30: Prijava u administratorsku aplikaciju

Donja slika prikazuje početnu stranicu koju vidi administrator. Za moderatora, ta je stranica ista, samo bez mogućnosti otvaranja pogleda *Moderatori* i *Dnevnik rada.* Što se pozadine tiče, podaci se dohvaćaju preko EJB-a s *findAll*() metodom. Ovdje nije potrebno koristiti vlastite filtere jer okvir obavlja cijeli posao filtriranja i sortiranja prema željenim kolonama. Svi pogledi koriste *@ViewScope* anotaciju. Aplikacija također podržava dvojezičnu lokalizaciju. Pritiskom na vlastito korisničko ime (gornji desni kut) moguće je izmijeniti korisničke podatke kao kod korisničke aplikacije.



Slika 31: Početna stranica administratorske aplikacije

6.4.2.2. Pretrage i prikaz tabličnih podataka

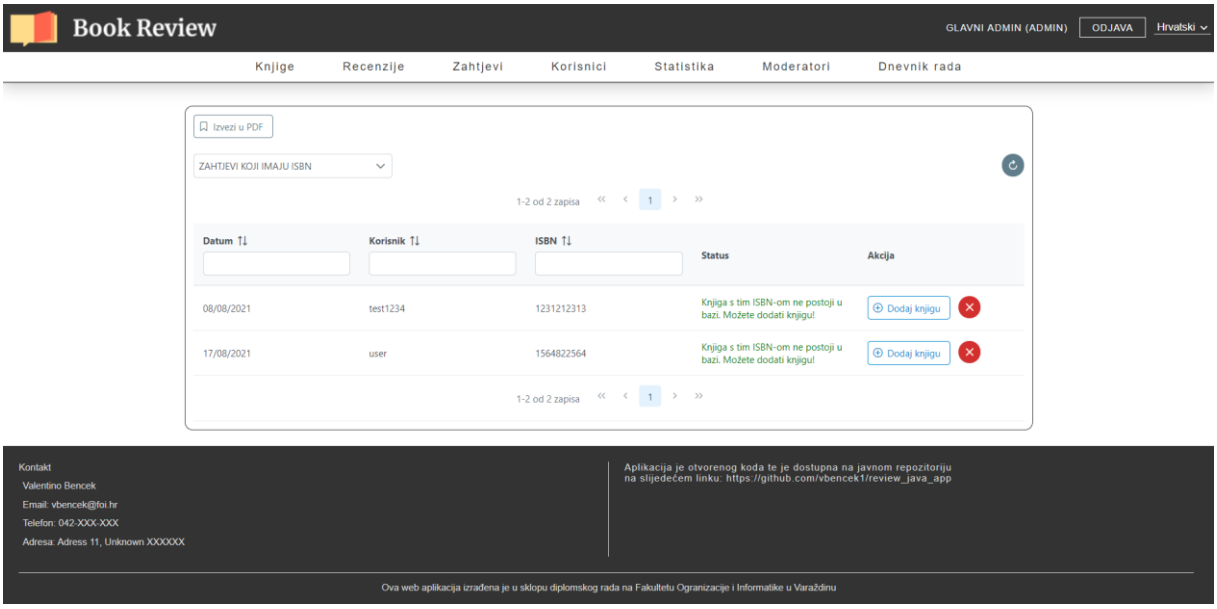
Za svaki prikaz koristi se posebna xhtml datoteka s pripadajućim zrnom. Donja slika tako prikazuje pogled za zahtjeve dodavanja knjige. Pogled omogućuje izvoz zapisa u pdf datoteku. Koristi se biblioteka otvorenog koda za generiranje PDF-a. potrebno je samo proslijediti listu za koju se želi generirati PDF.

```
public void generatePdf(){
        PdfGenerator pdfGenerator= new PdfGenerator();
        if ("2".equals(option)) {
            pdfGenerator.generatePdfForRequests(listRequestIsbn);
        } else {
            pdfGenerator.generatePdfForRequests(listRequestInfo);
        }
        String
summary=res.getString("admin.viewAdminRequests.pdf.summary");
        String
details=res.getString("admin.viewAdminRequests.pdf.details");
        addMessage(summary, details);
    }
```

Nadalje, postoji filter koji mijenja listu koja se prikazuje: zahtjevi s ISBN-om i zahtjevi s nazivom knjige. Korisnik može i ukloniti pojedini zahtjev. To povlači slanje e-maila svim

korisnicima čiji zahtjev je obrisan. Preko pojedinog zahtjeva može se i dodati knjiga. To je detaljnije objašnjeno u sljedećem poglavlju.

Slika 32: Email obavijesti o odbitku zahtjeva

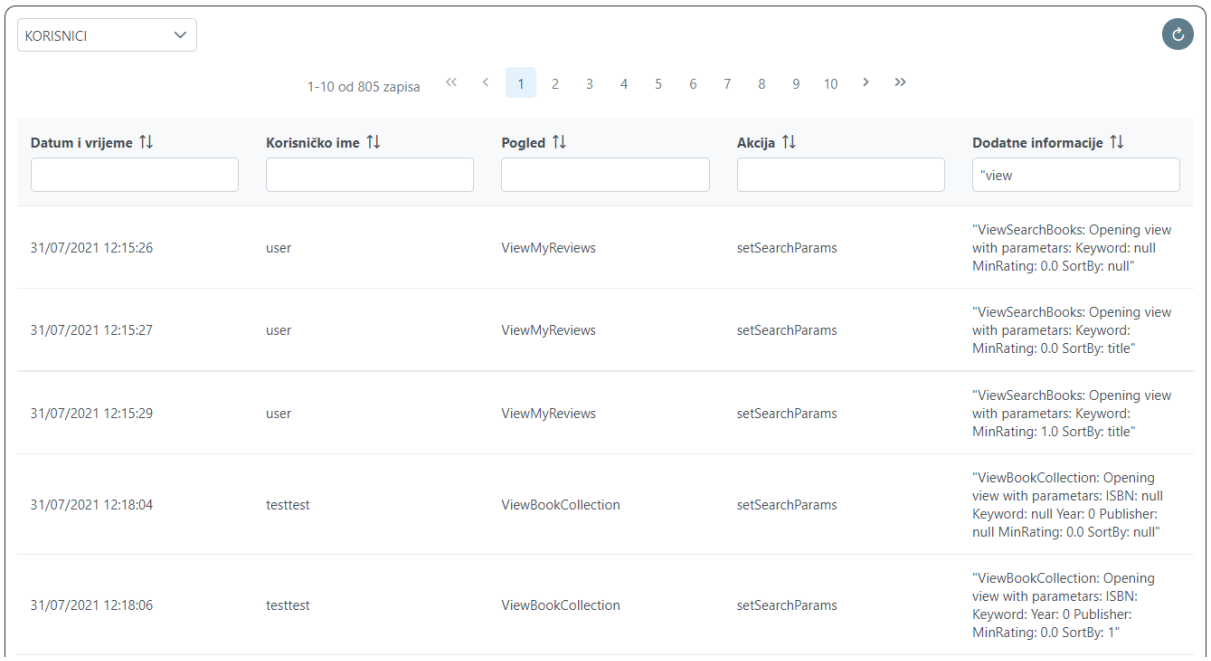


Slika 33: Prikaz zahtjeva za dodavanje knjiga

Svi tablični prikazi realizirani su na isti ili sličan način. Koriste komponentu *dataTable* iz Primefaces okvira. Toj komponenti se prosljeđuje lista koja se puni podacima iz baze. Neki pogledi omogućuju i selektiranje pojedinog retka gdje se prosljeđuje ID i na taj način se otvaraju detalji. Lista se dohvaća preko zrna, u atribut *var* definira pojedini model preko kojeg se dalje dohvaćaju atributi za prikaz. Donji isječak koda prikazuje dio iz xhtml datoteke za generiranje pogleda Dnevnika rada.

```
<p:dataTable id="logs" widgetVar="vtLog"
value="#{viewAdminLog.listDataLog}" var="row"

                              rows="10" paginator="true"
rendered="#{viewAdminLog.renderUserLog}"

                              paginatorTemplate="{CurrentPageReport}
{FirstPageLink} {PreviousPageLink} {PageLinks} {NextPageLink}
{LastPageLink} {RowsPerPageDropdown}"


currentPageReportTemplate="{startRecord}-{endRecord}
#{l['admin.viewAdminBooks.paginator.of']} {totalRecords}
#{l['admin.viewAdminBooks.paginator.records']}">

                    <p:column sortBy="#{row.actionDate}"
filterBy="#{row.actionDate}">
```

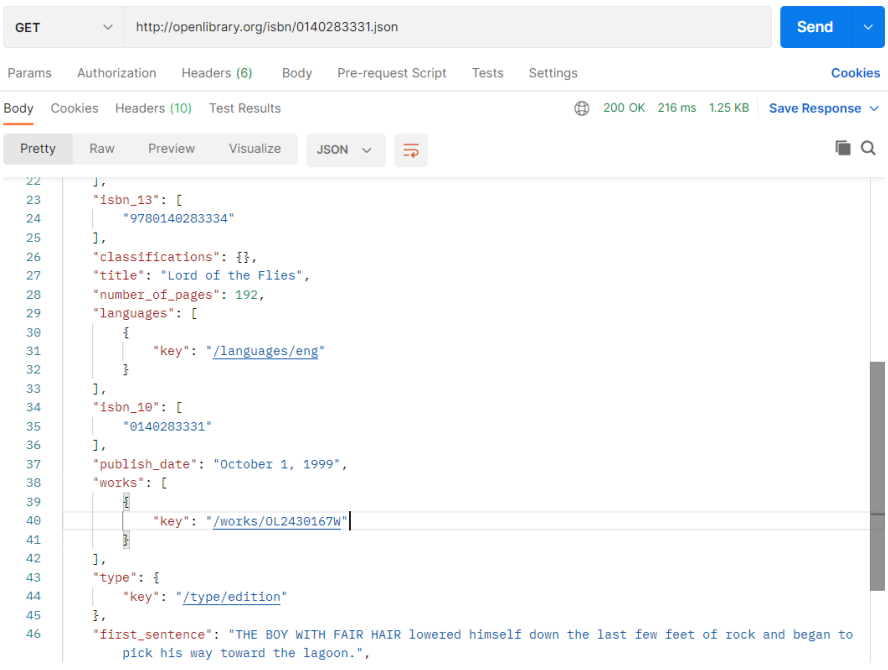
```
                        <f:facet
name="header">#{l['admin.viewAdminLog.date']}</f:facet>#{viewAdminLog
.convertToFriendlyDate(row.actionDate)}
                    </p:column>
                    <p:column sortBy="#{row.userId.username}"
filterBy="#{row.userId.username}">
                        <f:facet
name="header">#{l['admin.viewAdminLog.username']}</f:facet>#{row.user
Id.username}
                    </p:column>
                    <p:column sortBy="#{row.viewName}"
filterBy="#{row.viewName}">
                        <f:facet
name="header">#{l['admin.viewAdminLog.viewName']}</f:facet>#{row.view
Name}
                    </p:column>
                    <p:column sortBy="#{row.methodName}"
filterBy="#{row.methodName}">
                        <f:facet
name="header">#{l['admin.viewAdminLog.methodName']}</f:facet>#{row.me
thodName}
                    </p:column>
                    <p:column sortBy="#{row.parametars}"
filterBy="#{row.parametars}">
                        <f:facet
name="header">#{l['admin.viewAdminLog.params']}</f:facet>#{row.parame
tars}
                    </p:column>
</p:dataTable>
```



Slika 34: Prikaz dnevnika rada

6.4.2.3. Kreiranje knjiga

Knjiga se može kreirati na dva načina: dodavanjem nove knjige preko pogleda s listom knjiga, u tom slučaju korisnik sam puni informacije o knjizi, ili preko zahtjeva. Ukoliko se kreira preko zahtjeva s ISBN-om, šalje se REST poziv na OpenLibrary API. Primjer takvog poziva korištenjem Postman-a vidljiv je na donjoj slici. Kao odgovor dobiva se JSON kojeg je potrebno obraditi. Preko podataka iz JSONA kreira se novi objekt knjige.



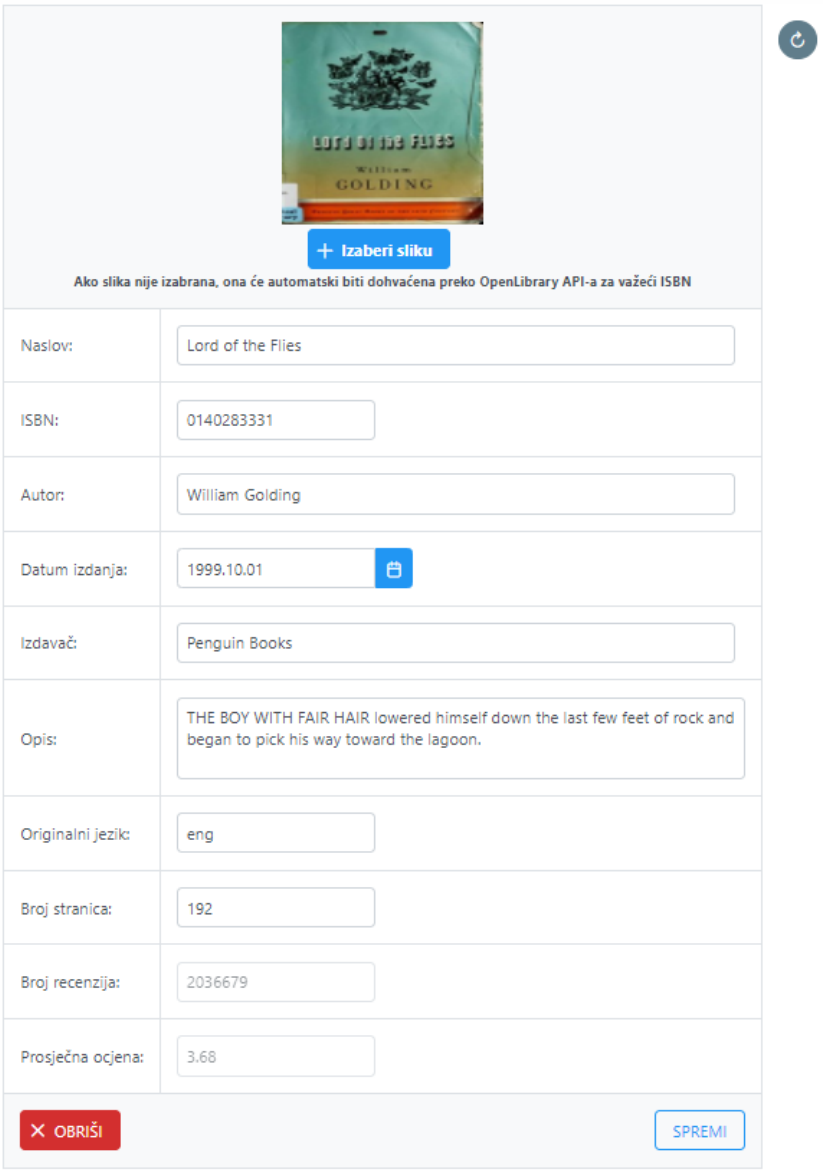
Slika 35: Postman - slanje zahtjeva na OpenLibrary

Ispod se nalazi programski isječak za slanje REST poziva. Pomoću klase Klijenta vrši se slanje GET upita na zadanu adresu. Kao odgovor dobiva se JSON dokument. Taj JSON dokument obrađuje se te se pohranjuje u java varijable. Zatim se kreira novi objekt knjige koji se puni s tim varijablama i takav objekt vraća se dalje pogledu na korištenje. Potrebno je poslati i REST poziv za dobivanje naziva autora i ključ jezika knjige. Poziv se šalje na isti princip.

```
public Book getBookByIsbn(String ISBN) {
        Book book = new Book();
        book.setIsbn(ISBN);
        try {
            Client client = ClientBuilder.newClient();
            String json = client.target(BASE_URI)
                    .path("isbn").path(ISBN + ".json")
                    .request(MediaType.APPLICATION_JSON)
                    .get(String.class);
            ObjectMapper mapper = new ObjectMapper();
            JsonNode node;
            node = mapper.readTree(json);
            …
            OVDJE RADIM PRIJENOS IZ JSONA NA VARIJABLE
            …
            …
            KREIRAM NOVU KNJIGU PREKO VARIJABLI
            …
        } catch (Exception ex) {
            …
        }return book; }
```

Nakon uspješnog dohvata podataka, forma za unos knjiga se automatski popunjava. Moguće je napraviti i modifikacije nad dohvaćenim podacima. Pritiskom na gumb „Spremi“ se spremaju podaci u bazu. Korisnik može poništiti svoje radnje pritiskom na sivi „Ponovo očitaj“ gumb. Također, odabirom postojeće knjige iz liste knjiga, korisnik može izmijeniti podatke ili izbrisati odabranu knjigu. Kreiranje knjige preko zahtjeva povlači slanje maila korisnicima koji su podnijeli taj zahtjev te se taj zahtjev briše iz baze. Donje slike prikazuje dobivenu e-mail poruku kao i formu s detaljima.

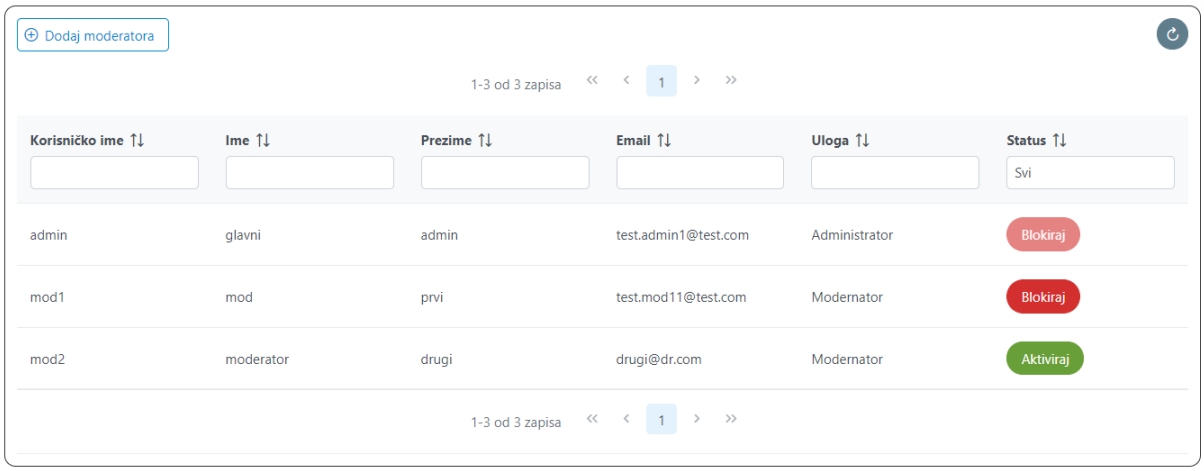
Slika 36: Email obavijest dodavanje knjige



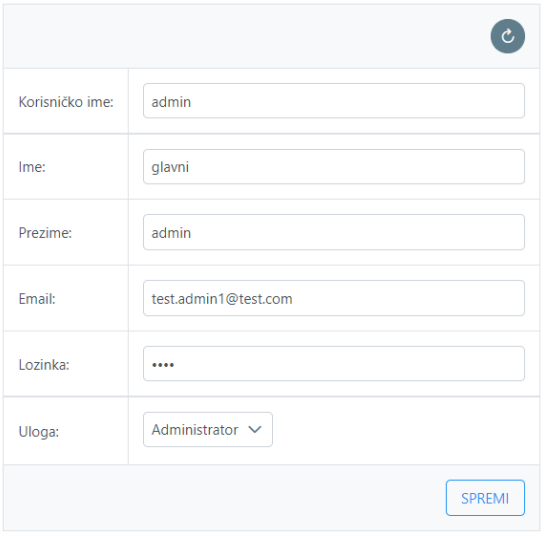
Slika 37: Detalji knjige administratorske aplikacije

6.4.2.4. Blokiranje korisnika i dodavanje moderatora

Moderator i administrator mogu blokirati korisnike dok samo administrator može blokirati i dodavati moderatore. Administratorski računi ne mogu biti blokirani. Putem forme za dodavanje moderatora može se dodati i administrator sustava ukoliko se odabere prava uloga. Preko iste forme mogu se izmjenjivati i informacije o moderatoru. Informacije o korisnicima se ne mogu mijenjati. Računi korisnika i moderatora ne mogu biti izbrisani već mogu samo biti blokirani. Donje slike prikazuju listu moderatora (lista za korisnike je istog izgleda) te formu za dodavanje/izmjenu moderatora.



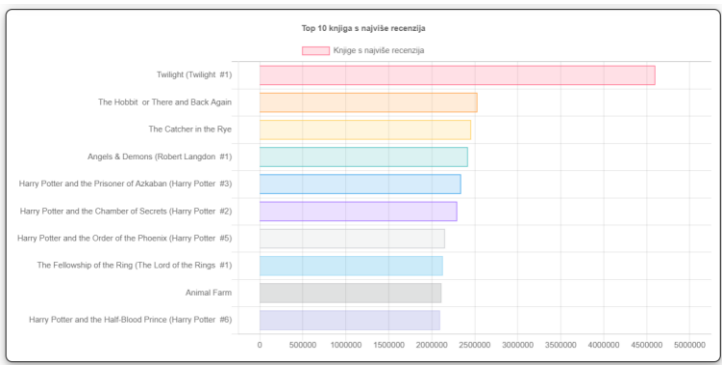
Slika 38: Pogled s listom moderatora



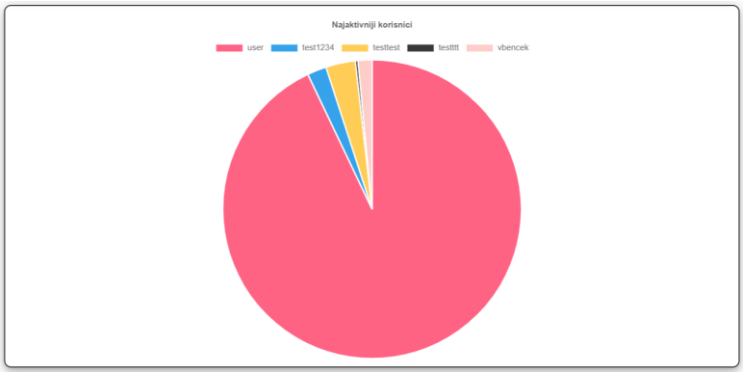
Slika 39: Detalji moderatora/administratora

6.4.2.5. Statistika

Generiranje grafa već je objašnjeno kod korisničke aplikacije. Ovdje se grafovi generiranju na isti način, korištenjem podataka dohvaćenih preko EJB-a.



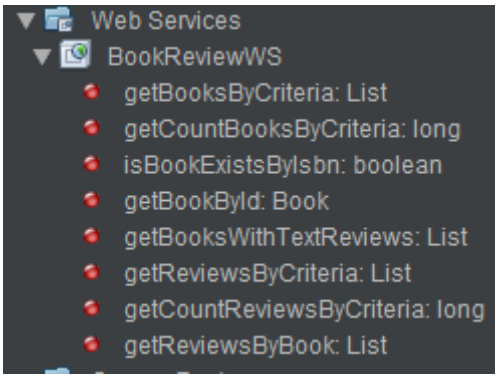
Slika 40: Graf s najpopularnijim knjigama



Slika 41: Statistika aktivnosti korisnika

6.4.3. Web servis

Administratorska aplikacija pruža vlastiti SOAP web servis. Omogućuje drugim aplikacijama da dohvaćaju informacije koje koristi aplikacija. Radi se o informacijama poput: ocjena knjiga, dohvata knjiga po kriterijima, dohvata osvrta na knjige i slično. Donjom slikom prikazane su sve metode koje web servis pruža.

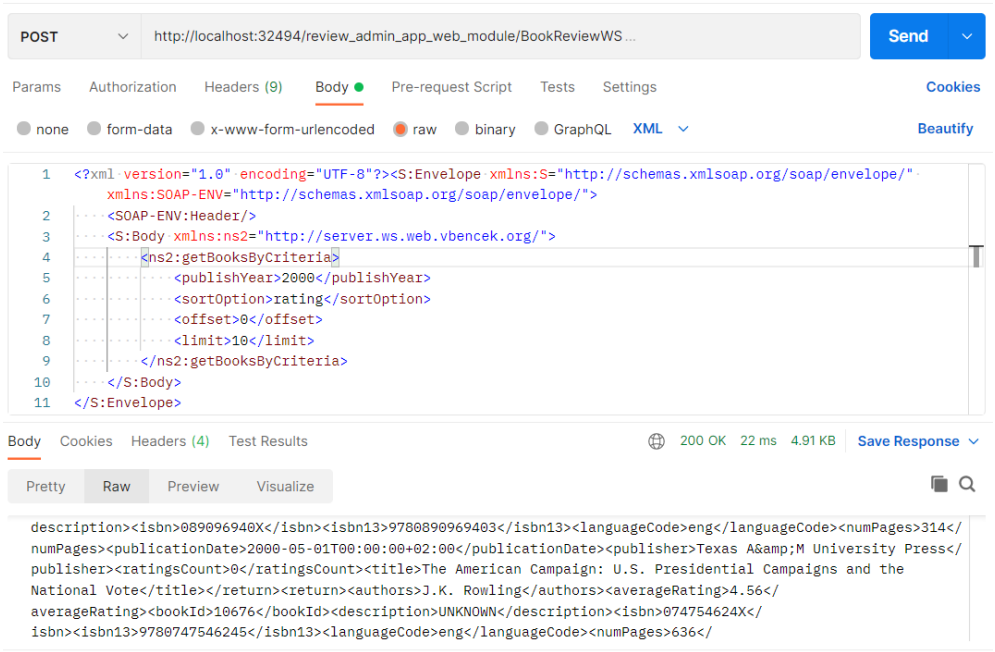


Slika 42: Metode web servisa

Programski isječak metode za dohvat broja knjiga prema kriterijima gdje se podaci ponovo dohvaćaju iz EJB-a odnosno baze:

```
@WebMethod(operationName = "getCountBooksByCriteria")
    public long getCountBooksByCriteria(
            @WebParam(name = "isbn") String isbn,
            @WebParam(name = "keyword") String keyword,
            @WebParam(name = "publishYear") int publishYear,
            @WebParam(name = "publisher") String publisher,
            @WebParam(name = "minimumAvgRating") double
minimumAvgRating) {
        return bookReviewData.getCountBooksByCriteria(isbn, keyword,
publishYear, publisher, minimumAvgRating);
    }
```

Donjoj slikom prikazan je primjer dohvaćanja podataka preko web servisa. Potrebno je poslati valjan XML format za zadanim parametrima kako bi se podaci mogli dohvatiti. Zahtjev se mora slati na adresu na kojoj je aplikacija postavila web servis.



Slika 43: Primjer slanja SOAP WS zahtjeva

6.5. Kritički osvrt na vlastitu web aplikaciju

Svaki programski proizvod, pa i ova aplikacija, ima mjesta za napredak. Nakon testiranja gotovog rješenja web aplikacije, postoje stvari koje bi se mogle realizirati na drugačiji način te bi u konačnici dale bolje rješenje. Aplikacija ima mnogo mjesta za napredak. Postoji velik broj funkcionalnosti koje bi se mogle implementirati, a da aplikacija istovremeno ostaje jednostavna i pregledna.

Iako sam zadovoljan kreiranom aplikacijom, jedna od stvari koju bih promijenio je dizajn korisničke aplikacije. Smatram kako dizajn nije dovoljno moderan da zadovolji današnje standarde dizajna. Kod dizajna mi se sviđa jednostavnost. Korisnik ne bi trebao imati problema prilikom navigacije kroz korisničko sučelje te bi mu sadržaj trebao biti čitljiv i razumljiv. Također, jedna od stvari koje bih nadodao je sadržaj. Detalji knjige dosta su šturi i aplikacija se većinom oslanja na svoje korisnike koji bi dodavali sadržaj. Dalje, što se dizajna tiče, razmotrio bih opciju dodavanja zasebnog dizajna za mobilne uređaje. Trenutno, manji ekrani imaju prilagođeni pogled koji baš i ne odgovara standardu izgleda stranica na mobilnim uređajima. Samim time, poboljšao bih i responzivnost dizajna jer nije velika pažnja posvećena tome.

Potrebno je poboljšati korisničko iskustvo. Trenutno, postoji manji broj poruka koje se prikazuju korisniku kao način dobivanja povratne informacije o njegovim akcijama.

Promijenio bih i nekoliko stvari koje se nalaze u pozadini (engl. Backend). Tijekom implementacije, primijeti sam da postoje metode koje se ponavljaju. Te metode trebalo bi prikupiti i napraviti zasebnu klasu iz koje bi se pozivale kako bih ih mogao ponovo iskoristiti (engl. Reusability). Implementirao bih i bolji način zapisa u dnevnik (engl. Logging) . Trenutan način je prikazuje samo SQL upite i nekoliko personaliziranih poruka. Potrebno je dodati i filter koji lovi zahtjeve (engl. Request) kako bih pojedini zahtjev mogao upisati u bazu, tablicu dnevnika rad. Trenutan način unosa u dnevnik rada nije pogodan za skalabilne aplikacije jer se „ručno“ odlučuje što će se zapisati, a što ne.

Aplikacija obrađuje dosta opširnu temu te su mogućnosti za nadogradnju velike. Dodao autora kao zasebnu klasu te bih imao mogućnosti prikazati informacije o njemu te filtrirati po atributima autora. Otvorila bi se i mogućnost za dodavanje autora u kolekciju te lakši pristup njegovim knjigama. Isti princip mogao bi se primijeniti i na izdavača. Također, proširio bih statistiku gdje bih napravio obradu većeg skupa podataka i generirao opširni prikaz statistike za pojedine elemente.

Zadovoljan sam krajnjim rezultatom implementacije, smatram da je aplikacija jednostavna i intuitivna za korištenje. Također, sadržaj aplikacije je ažuran i dohvat samih podataka optimiziran te aplikacija ima kratko vrijeme odaziva na zahtjev. Kroz administratorsku i korisničku aplikaciju, web aplikacija ima pokrivene sve CRUD operacije nad objektima.

7. Zaključak

Danas, web aplikacije, prisutne su u svim aspektima života. Koriste se za vlastite potrebe korisnika, u javnoj upravi, u poslovanju i slično. Mogu biti jednostavne za korištenje te ih može koristiti bilo koja informatički pismena osoba ili mogu biti složene. Složene aplikacija zahtijevaju trening i učenje korisnika.

Najveći dio ovog rada odnosi se na otvorenost izvornog koda aplikacije. Otvoreni kod oko sebi okuplja zajednicu ljudi, koji za cilj imaju međusobno razmjenjivanje ideja, rješenja, programskih proizvoda te tehnologija. Očuvanje koncepta otvorenosti bitno je za razvoj informatičke tehnologije jer kao proizvod daje transparentan softver. Taj softver može se dalje nadograđivati ili se iz njega može kreirati novi softver, i tako dalje za idući softver. Zapravo, otvorenost koda doprinosi konstantnom razvoju softvera. Konstantnim razvojem dobiva se siguran, optimiziran i kvalitetan kod.

Vlastita aplikacija razvijena je korištenjem otvorenih tehnologija. To je uvelike olakšalo proces razvoja aplikacije. Korištene tehnologije licencirane su raznim licencama. Ukoliko bi se odlučio ovu aplikaciju pretvoriti u komercijalni softver, morao bih neke biblioteke izbaciti ili dodatno platiti. Naravno, postoje biblioteke pod MIT licencom koje bih mogao koristiti bez potrebe otvaranja vlastitog koda, ali postoje i tehnologije koje ne bih mogao koristiti. Bez bi bilo gotovo nemoguće razviti ovu aplikaciju ili znatno otežano. Također, korištenjem otvorenih tehnologija, aplikacija mora također biti izdana kao otvoreni projekt. Ovo je zapravo i velika prednost jer se na projekt mogu uključiti drugi programeri. Proširenje ove aplikacije zahtjevan je posao za jednu osobu, no uz pomoć zajednice, aplikacija bi mogla postati dobra alternativa postojećim aplikacijama slične tematike. Licenciranjem aplikacije GPLv3 licencom, osigurava se da će danje inačice ove aplikacije biti javno dostupne, odnosno njihov izvorni kod će također morati biti otvoren.

Popis literature

[1] Yeeply (2016), *Web Application Development: 5 relevant types and examples*, dostupno na URL: https://en.yeeply.com/blog/6-different-kinds-web-app-development/ , preuzeto 10.03.2021.

[2] J. McGovern, S. Tyagi, S. Mathew, M. Stevens, Java Web Services Architecture. Morgan Kaufmann, 2010.

[3] Sparr M. (2018), *How Web Applications Work*, dostupno na URL: https://medium.com/@mikesparr/how-web-applications-work-3824f4b7ebeb , preuzeto 13.03.2021.

[4] Datarob, *What is Web Application Architecture?,* dostupno na URL: https://datarob.com/what-is-web-application-architecture/ , preuzeto 15.03.2021.

[5] L. Shklar, R. Rosen, Web application architecture: Principles, protocols and practices. John Wiley & Sons, 2008.

[6] V. Osetskyi (2018) , *Web Application Architecture*, dostupno na URL: https://existek.com/blog/web-application-architecture/ , preuzeto 16.03.2021.

[7] R. Lepofsky, The manager's guide to web application security: A concise guide to the weaker side of the web, 1st ed. Berlin, Germany: APress, 2014.

[8] Iron Bastion (2019), *A Guide to Open-Source Software Security Risks & Best Practices*, dostupno na URL: https://blog.ironbastion.com.au/open-source-software-security-risks-practices/ , preuzeto 19.03.2021.

[9] M. Howard, D. LeBlanc, Writing secure code: Practical strategies and proven techniques for building secure applications in a networked world, 2nd ed. Redmond, WA: Microsoft Press, 2003.

[10] Netbeans, *Securing a Web Application in NetBeans IDE*, dostupno na URL: https://netbeans.apache.org/kb/docs/web/security-webapps.html , preuzeto 20.03.2021.

[11] Bateson C. (2018), *Top 5 Java Application Security Frameworks to Build Secure Apps*, dostupno na URL: https://chrisbateson80.medium.com/top-5-java-application-security-frameworks-to-build-secure-apps-e5256b45452b , preuzeto 20.03.2021.

[12] Vuksanovic P. I., Sudarevic B., *Use of web application frameworks in the development of small applications*, MIPRO, 2011.

[13] MacPherson L. (2019), *The Pros and Cons of Building a Mobile App vs. a Web App*, dostupno na URL: https://designli.co/blog/the-pros-and-cons-of-building-a-mobile-app-vs-a-web-app/ , preuzeto 24.03.2021.

[14] Digitalskynet.com (2020), *Desktop App vs Web App: Comparative Analysis*, dostupno na URL: https://digitalskynet.com/blog/Desktop-App-vs-Web-App-Comparative-Analysis , preuzeto 24.03.2021.

[15] Zhaohui W., *Research on the Application of Open Source Software in Digital Library*, Fujian University of Technology, 2011.

[16] Pickett P. (2019), *How Open-Source Software Works*, dostupno na URL: https://www.thebalancecareers.com/what-is-open-source-software-2071941 , preuzeto 28.03.2021.

[17] Guru99.com*, Difference between Website and Web Application*, dostupno na URL: https://www.guru99.com/difference-web-application-website.html , preuzeto 24.03.2021.

[18] Gramstad R. A., *Proprietary Software, Free and Open-Source Software, and Piracy: An Economic Analysis*, Department of Economics University of Oslo, 2012.

[19] Thompson A. (2020), *What is proprietary software and how does it work?,* dostupno na URL: https://entrepreneurhandbook.co.uk/proprietary-software/, preuzeto 7.04.2021.

[20] Al-Rousan T., Hadidi B. i drugi, *Web Application Development Processes: Requirements, Demands and Challenges, Faculty of Science and Information Technology*, Isra University, Amman, Jordan

[21] Riisalo, Tuomas & Haddad Nosrati, Navid, *The design and development of a web application to improve business processes and performance in an innovative media company: a case study of JS Suomi Ltd*, Laurea University of Applied Sciences, 2017.

[22] Baskerville R.,Ramesh B. i drugi, *Is Internet-Speed Software Development Different?,* IEEE Xplore, 2003.

[23] Existek (2018) , *Web Application Development Process* , dostupno na URL: https://existek.com/blog/web-application-development-process-flow/ , preuzeto 11.04.2021.

[24] Vogel L. (2016), Introduction to Java Web development – Tutorial, dostupno na URL: https://www.vogella.com/tutorials/JavaWebTerminology/article.html , preuzeto 13.04. 2021.

[25] Nourie D. (2006) , Java Technologies for Web Applications , dostupno na URL: https://www.oracle.com/technical-resources/articles/java/webapps.html , preuzeto 13.04.2021.

[26] Potdar V., Chang E., *Open Source and Closed Source Software Development Methodologies,* School of Information System, Curtin University of Technology, 2004.

[27] T.Y.B.Sc. (CS), *Principles of Web Design & Web Technologies, Institute of Distance and Open Learning*, University of Mumbai, 2019

[28] Logvinenko A., *SaaS Web Application Development Principles to Be Followed*, Mobidev, 2020

[29] Little C. (2019) , *7 challenges in web application design and development* , dostupno na URL: https://tillerdigital.com/blog/7-challenges-in-web-application-development/ , preuzeto 20.04.2021.

[30] UI Bakery Team (2020) , *Web application development in 2020: challenges, changes, trends, plans* , dostupno na URL: https://uibakery.io/post/web-application-development-2020 , preuzeto 21.04.2021.

[31] Synopsys.com, *Open source software,* dostupno na URL: https://www.synopsys.com/glossary/what-is-open-source-software.html , preuzeto 28.03.2021

[32] Opensource (2021) , *Licenses & Standards* , dostupno na URL: https://opensource.org/licenses , preuzeto 24.04.2021.

[33] Goldstein A. (2019) , *Open Source Licenses Explained* , dostupno na URL : https://resources.whitesourcesoftware.com/blog-whitesource/open-source-licenses-explained , preuzeto 24.04.2021.

[34] Tldrlegal (2021) , *Software Licenses in Plain English,* dostupno na URL: https://tldrlegal.com/ , preuzeto 24.04.2021.

[35] Synopsys (2019) *, Top open source licenses and legal risk for developers,* dostupno na URL: https://www.synopsys.com/blogs/software-security/top-open-source-licenses/ , preuzeto 24.04.2021.

Popis slika

Popis tablica