Queueprefix: web aplikacija za povezivanje igrača računalnih igara

Balen, Ivan

Sveučilište Jurja Dobrile u Puli

Fakultet Informatike u Puli

Ivan Balen

QueuePrefix: Web aplikacija za povezivanje igrača računalnih igara

Diplomski rad

Pula, rujan, 2021.

Sveučilište Jurja Dobrile u Puli

Fakultet Informatike u Puli

Ivan Balen

QueuePrefix: Web aplikacija za povezivanje igrača računalnih igara

Diplomski rad

JMBAG: 0303054222, redoviti student

Studijski smjer: Informatika

Predmet: Napredni algoritmi i strukture podataka

Znanstveno područje: Društvene znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: izv. prof. dr. sc. Tihomir Orehovački

Pula, rujan, 2021.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani **Ivan Balen**, kandidat za magistra **informatike** ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Ivan Balen

U Puli, 17.09.2021. godine

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, **Ivan Balen** dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom **QueuePrefix: Web aplikacija za povezivanje igrača računalnih igara** koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama. Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

Student

Ivan Balen

U Puli, 17.09.2021. godine

QueuePrefix: Web aplikacija za povezivanje igrača računalnih igara

IVAN BALEN

Sažetak: Za diplomski rad implementirana je web aplikacija za povezivanje igrača računalnih igara koristeći Node.js i React.js biblioteku. React.js je JavaScript biblioteka otvorenog koda koja se koristi za izradu korisničkih sučelja, posebno za aplikacije na jednoj stranici. Aplikacija se sastoji od 2 sloja, klijentski i poslužiteljski sloj. Kroz dijagram obrasca uporabe i slijeda možemo dobiti vizualni dojam funkcionalnosti aplikacije, a u nastavku je objašnjen svaki dio sučelja te njegova svrha i upute za korisnika kako i što može dobiti od pojedinog elementa. Nakon što je prikazano sučelje, pokazana je implementacija, objašnjene su sve tehnologije koje su korištene, i stavljen je primjer za svaku iz koda koja najbolje prikazuje navedenu funkcionalnost.

Ključne riječi: JavaScript, React.js, Node.js, povezivanje, računalne igre, biblioteka.

QueuePrefix: Web application for connecting computer games players

IVAN BALEN

Abstract: A web application for connecting computer games players using Node.js and React.js library was implemented for the thesis. React.js is an open source JavaScript library used to create user interfaces, especially for single-page applications. The application consists of 2 layers, client and server layer. Through the use case and sequence diagram we can get a visual impression of the functionality of the application, and afterwards we have a description of each part of the application's interface and it's purpose and instructions for the user as to how and what he can do with each element. After the interface is shown, the implementation is shown, all the technologies used are explained, and a code example is given for each of them that illustrates the stated functionality.

Keywords: JavaScript, React.js, Node.js, connecting, computer games, library.

Sadržaj

1. Uvod

Društveni mediji računalna su tehnologija koja razmjenom ideja, misli i informacija olakšava izgradnju virtualnih mreža i zajednica. Po dizajnu, društveni mediji temelje se na internetu i korisnicama omogućuju brzu elektroničku komunikaciju sadržaja. Sadržaj uključuje osobne podatke, dokumente, fotografije itd. Korisnici komuniciraju s društvenim medijima putem računala, tableta ili pametnog telefona putem web softvera ili aplikacija.

Inspirirano time, ovaj diplomski rad će opisivati razradu web aplikacije QueuePrefix, aplikacija koja daje nekakav aspekt društvenih mreža igračima tako što mogu napraviti svoj profil sa svojim iskustvom i općenitim informacijama i time se mogu upoznavati s ostalim igračima, raditi objave i tako stvarati nove prijatelje ili poslovne prilike, u slučaju da se okupe kompetitivni igrači. Igrač je osoba koja se bavi interaktivnim igrama, osobito videoigrama, stolnim igrama, igrama gdje preuzima nekakvu imaginarnu ulogu, kartaškim igrama itd. Neki igrači su konkurentni, što znači da se u nekim igrama natječu za novac, stoga takvog igrača možemo deklarirati kao atletičara.

Cilj aplikacije je jednostavan i brz prikaz svih korisnika, zajedno s svim njihovim informacijama koje su potrebne da drugi igrač dozna želi li on stupiti u kontakt s njime i igrati iz zabave ili iz ozbiljnosti, u slučaju da to želi tretirati kao posao, a u slučaju da se igrač ne želi upoznati s drugim igračima, postoji opcija objava gdje može komunicirati s drugim igračima bez da ih upoznaje, već ako mu nešto nije jasno, može postaviti upit i vidjeti što su ostali odgovorili, kao što se recimo može na stranicama poput Reddita ili pristupom chatu koji je postavljen uživo.

Analizom podataka možemo zaključiti da gotovo svi ispitanici koriste društvene mreže. Na temelju rezultata faktorske analize, njihova motivacija za korištenje društvenih mreža je zabava, traženje informacija, osobna korisnost i praktičnost. To su stvari koje su previše općenite i ne odnose se konkretno na igrače, već na širu publiku. Činjenicom da je ovo nekakva mini društvena mreža za igrače, samim time će se i publika filtrirati i privlačit će samo ljude iste kulture. Dakle ciljano tržište su igrači koji žele pristupiti nekakvom sustavu gdje mogu upoznavati druge igrače i znaju da točno

to dobivaju od tog proizvoda. Aplikacija je namijenjena za sve vrste uređaja koji su kompatibilni s web browserima poput Chrome, Mozilla Firefox itd.

1.1. Konkurencija

Nije tajna da su igrači iznimno aktivni i angažirani na mrežnim platformama. Njihova potreba za povezivanjem, strategiranjem i oblikovanjem odnosa je neupitna. S više od 2.5 milijardi igrača u svijetu, evidentno je da je velika većina njih na društvenim mrežama. Discord je popularna platforma za igrače koja im omogućuje međusobnu komunikaciju putem glasa, chata ili videa. Posebno je popularan među PC igračima koji će koristiti ovu platformu za međusobno razgovaranje tijekom igranja svojih igara. Discord omogućuje korisnicima stvaranje poslužitelja na kojima tada možete postaviti različite opcije chata. Ljudi se tada mogu upoznati, družiti na Discord-u, što ga čini atraktivnom platformom za igrače.

Osim Discorda, vrijedno je još napomenuti Reddit i Steam. Vjerojatno postoji subreddit za sve teme ovoga svijeta, pa tako i za igrače. Korisnici na Redditu pojavljuju se na ovim subreddit-ima i pokreću priču o bilo kojoj temi koja se odnosi na igru. Povratne informacije, pritužbe, strategije, greške itd. Reddit također ima moderatore koji će osigurati da se korisnici pridržavaju pravila grupe. Moderirani forum omogućuje kvalitetne rasprave i vrijedan uvid u umove korisnika.

U drugu ruku imamo još Steam, on je sveobuhvatna platforma koja nudi igre, video streaming, hosting i druge usluge društvenih mreža. Njegova je usluga prikupila više od 20 milijuna korisnika, a igre poput Counter-Strike, Dota i PLAYERUNKNOWWN'S BATTLEGROUNDS su neke od najpopularnijih igara na njihovoj platformi. Steam publika je namijenjena da se igrači pridruže čvorištima specifičnim za igre. Unutar tih čvorišta korisnici mogu raspravljati o igri, slati snimke zaslona, videozapise, stvarati smjernice kako da drugi igrači budu bolji u igrama i još mnogo toga. To je zajednica u kojoj se igrači okupljaju radi umrežavanja i druženja, u većini slučajeva se i sprijatelje i onda zajedno igraju na Steamu.

2. SWOT Analiza

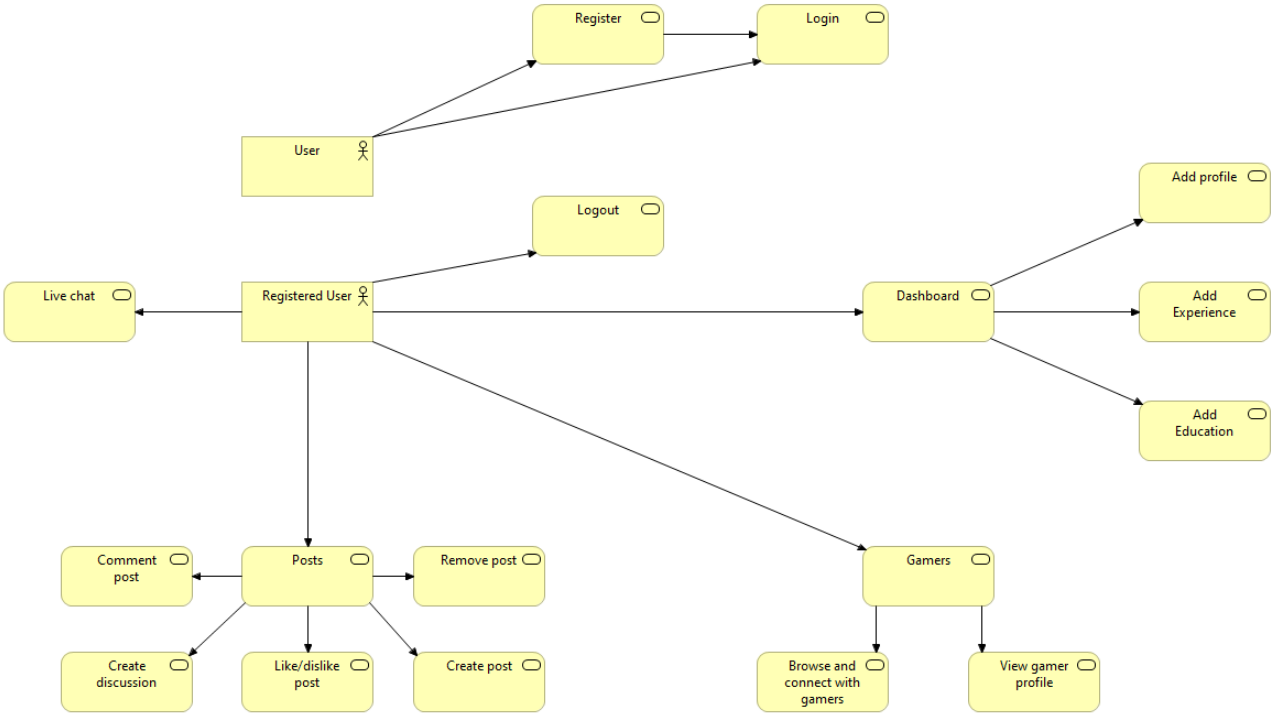
Iz SWOT analize na slici 1 vidljive su prednosti i mane aplikacije. Glavna prednost bi bila jednostavnost i brzina aplikacije, a glavni problem aplikacije je njen budžet koji jednostavno nije realan u usporedbi s konkurencijom koja se nalazi na marketu već duže vrijeme. Primjerice uzmimo Steam, postoje brojni razlozi zbog kojih je Steam uspio postati tako uspješan. Prvo, Valve je imao kritičan IP (vlastiti katalog igara) koji je mogao koristiti za privlačenje velike početne baze korisnika. Steam je pokrenut 2003. godine kako bi podržao rast virusa na Counter-Strikeu, najuspješnijoj pucačini svih vremena (čak i danas). Do sada je prodano preko 50 milijuna primjeraka. Iako je u to vrijeme brzo rastao, CS je pogođen s raznim softverima gdje igrači dobivaju veliku prednost nad drugim igračima kao što je automatsko ciljanje, gledanje kroz zidove itd, a njegova baza igrača bila je fragmentirana u više verzija. Koristeći svoje vlasništvo nad temeljnom IP adresom, Valve je uspio usmjeriti sve igrače putem Steam platforme i riješiti sve kritične točke. Drugo, iako je Steam počeo nuditi samo igre, Valve je prepoznao da je stupanj rascjepkanosti programera igara uz previsoke troškove fizičkog objavljivanja i distribucije igara, sve veći izazov upravljanja digitalnim pravima i porast distribucije digitalnog sadržaja zapravo idealna prilika da Valve otvori platformu na kojoj može omogućiti prodaju video igara. Steam je postao toliko uspješan u tome da čak njegovi konkurenti prodaju svoje igre na njegovoj platformi, poput Microsoft-a.

INTERNI FAKTORI	
SNAGE (+)	**SLABOSTI (–)**
- Efektivna klasifikacija korisnika - Brz rad aplikacije - Aplikacija nije desktop aplikacija, stoga se ne treba instalirati - Korištenje tehnologija koje su na dobrom glasu poput React.js, Node.js	- Loš menadžment - Pogrešna primjena određenih praksi - Nikakav konkretan način komunikacije osim preko objava - Niski budžet - Jednostavnost aplikacije
EKSTERNI FAKTORI	
PRILIKE (+)	**PRIJETNJE (–)**
- Potencijalni novi marketi - Razvijanje novih tehnologija - Rast potražnje - Nadograđivanje aplikacije u slučaju da se budžet povisi	- Jako velika i već dobro uspostavljena konkurencija - Cijena marketinga - Konkurencija je već uspostavila veliki community koji će se teško prebaciti na drugu uslugu - Konkurencija ima veći budžet, resurse I reklamiranje

Slika 1. SWOT analiza.

3. Dijagram obrasca uporabe

Prikazani dijagram jasno opisuje funkcionalnosti ponašanja, odnosne sustave i obrasce uporabe u interakciji sa sudionicima sustava, odnosno akterima. U sustavu se vidi akter pod nazivom „Registered user“ (registrirani korisnik), koji ima pristup glavnim funkcionalnostima aplikacije navedenih u dijagramu. Momenat kada se akter registrira, navigiran je na „Dashboard“ (kontrolna ploča) gdje si treba urediti svoj profil. Pod „Gamers“ (igrači) on može vidjeti ostale račune koji su kreirani, a pod „Posts“ (objave) može komunicirati s drugim korisnicima, kao što je vidljivo na slici 2.

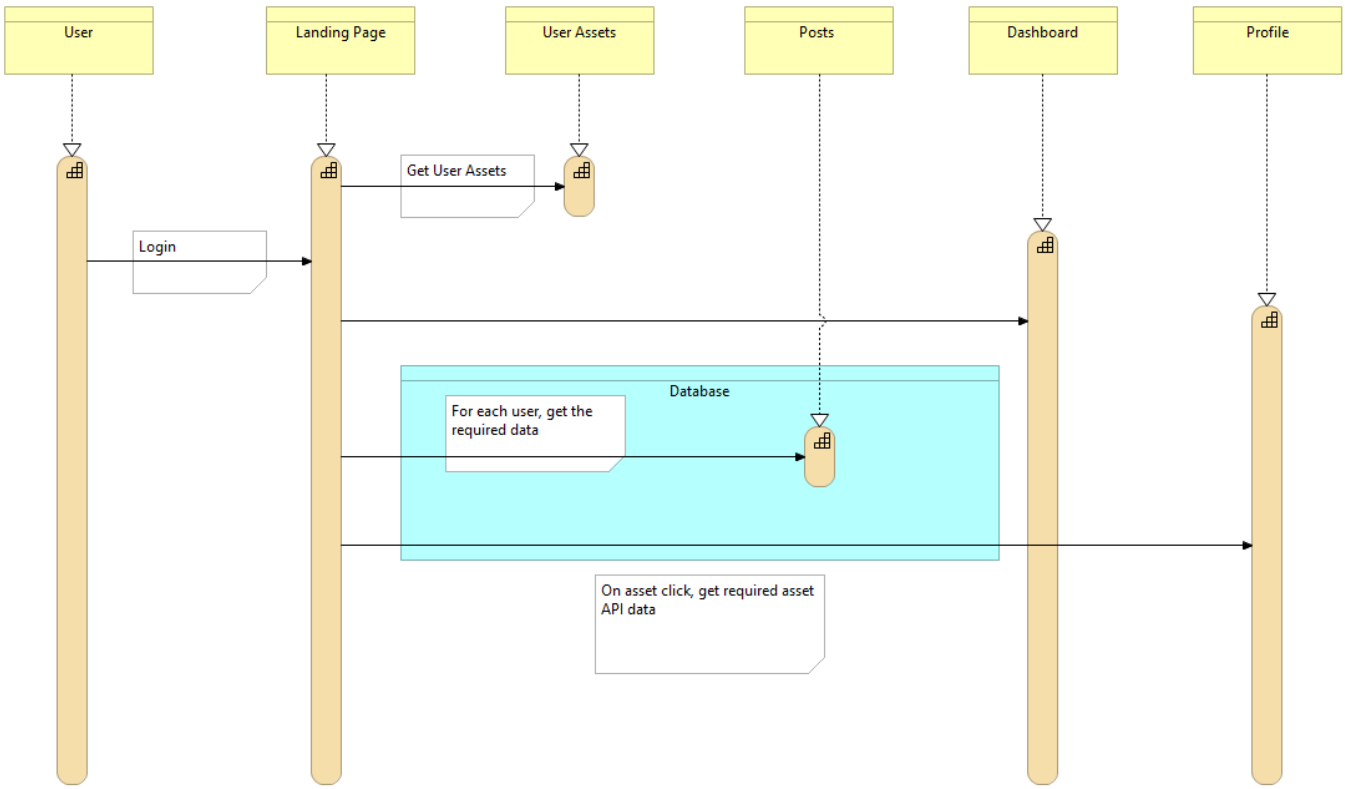


Slika 2. Dijagram obrasca uporabe.

4. Dijagram slijeda

Dijagram slijeda pokazuje sekvencu određenih procesa koji se dešavaju u aplikaciji između korisnika i mogućnosti koje ona nudi. Jednostavnije rečeno, dijagram slijeda prikazuje različite dijelove sustava koji rade u „slijedu“ kako bi se nešto učinilo. Oznake dijagrama slijeda su strukturirane na takav način da predstavljaju vremensku crtu koja počinje pri vrhu i postupno se spušta kako bi označila slijed interakcija. Svaki objekt ima stupac, a poruke koje se razmjenjuju prikazane su strelicama.

U ovom slučaju, dijagram govori da sve započinje s korisnikom koji uđe u sustav, dolazi na početnu stranicu i onda otuda ovisno o njegovoj specifikaciji se određeni elementi pokazuju, točnije dovlače se s API-a. Isto pravilo vrijedi za svakog korisnika i nema nikakvih iznimki, dok god postoji korisnik, tok podataka funkcionira na isti način kao što vidimo na slici 3.



Slika 3. Dijagram slijeda.

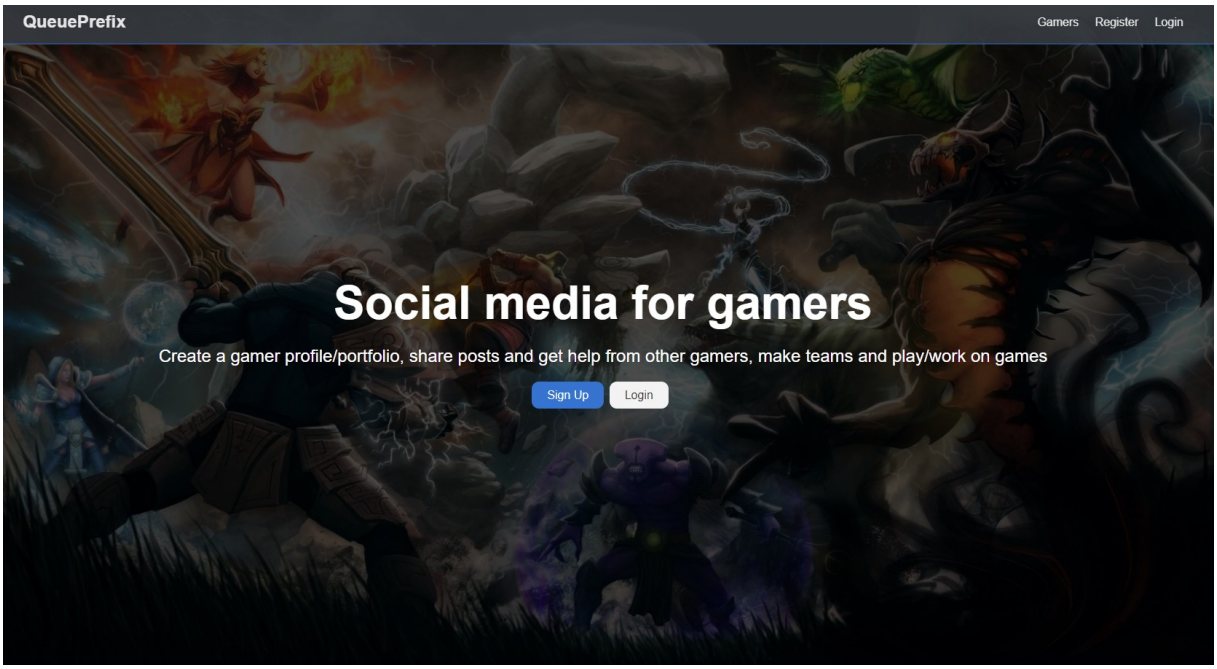
5. Razrada funkcionalnosti

U ovoj aplikaciji postoji samo jedna vrsta korisnika s jednom vrstom korisničkog računa. Korištenjem dijagrama obrasca uporabe sa slike 2 koji opisuje ponašanje sustava u interakciji sa sudionicama sustava (akterima), prikazane su funkcionalnosti koje vide korisnici i akcije koje sustav upotrebljava.

Osim dijagrama obrasca uporabe imamo još i dijagram slijeda koji se nalazi na slici 3 pomoću kojeg vidimo opis slijeda kako se sustav koristi u vremenu, on nam detaljnije pokazuje realizaciju pojedinog obrasca. U daljnjoj razradi funkcionalnosti prikazano je sučelje po pojedinim dijelovima u aplikaciji te sam dizajn.

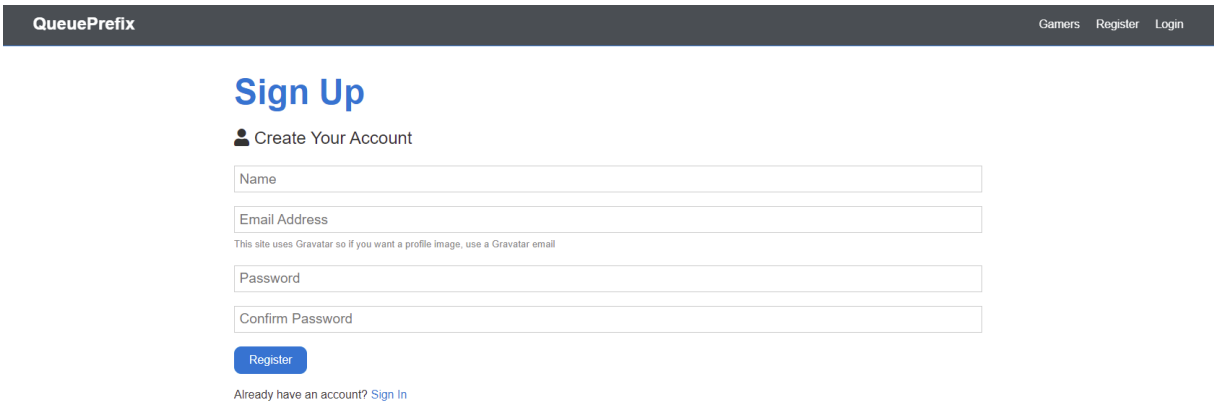
5.1. Početna stranica

Prilikom dolaska na web aplikaciju, korisnik se nalazi na početnoj stranici koja je vidljiva ispod na slici 4. gdje ima opciju za prijavu i registraciju [10]. Osim toga, u slučaju da se ne želi automatski prijaviti na stranicu, može vidjeti popis svih korisnika koji imaju svoj profil ako klikne na Gamers (igrači). Aplikacija sadrži još dvije komponente, to su Dashboard (kontrolna ploča) i LiveChat (chat uživo) koji nisu vidljivi ako je korisnik običan gost, točnije ako nije prijavljen na aplikaciju.



Slika 4. Početna stranica [10].

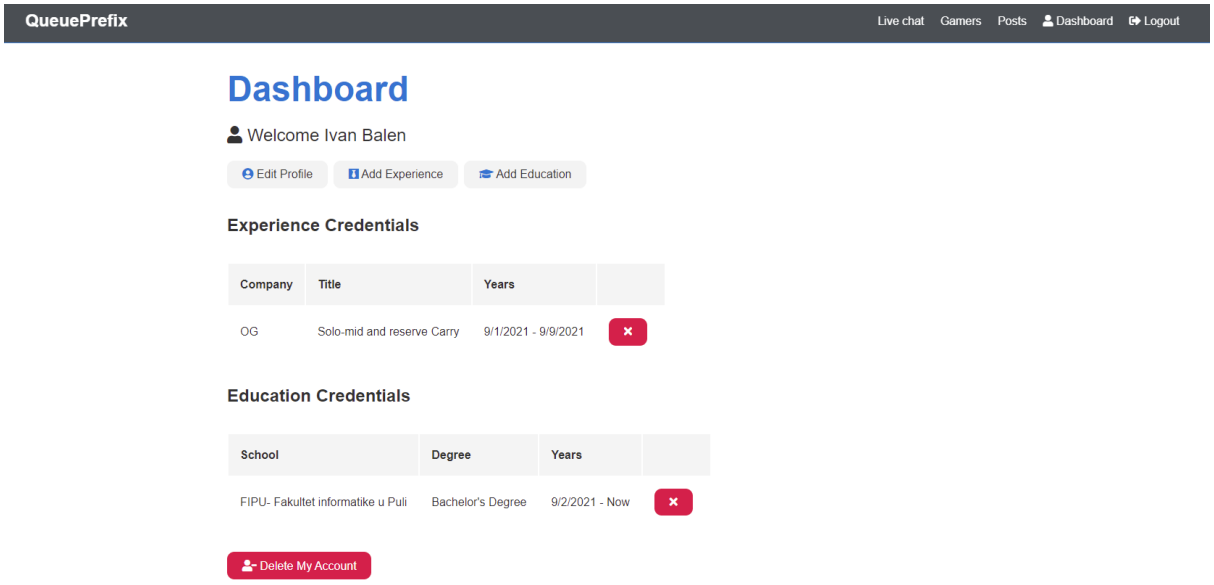
U slučaju da korisnik nije napravio račun, to može napraviti pod opcijom „Register“ (registracija) koju vidimo na slici 5, tako što unese potrebne informacije vezane za svoj račun, ako želi imati sliku na profilu, potrebno je iskoristiti Gravatar mail. Izradom računa, korisnik prelazi na „Dashboard“ (kontrolna ploča).



Slika 5. Stranica za registraciju korisnika.

5.2. Kontrolna ploča

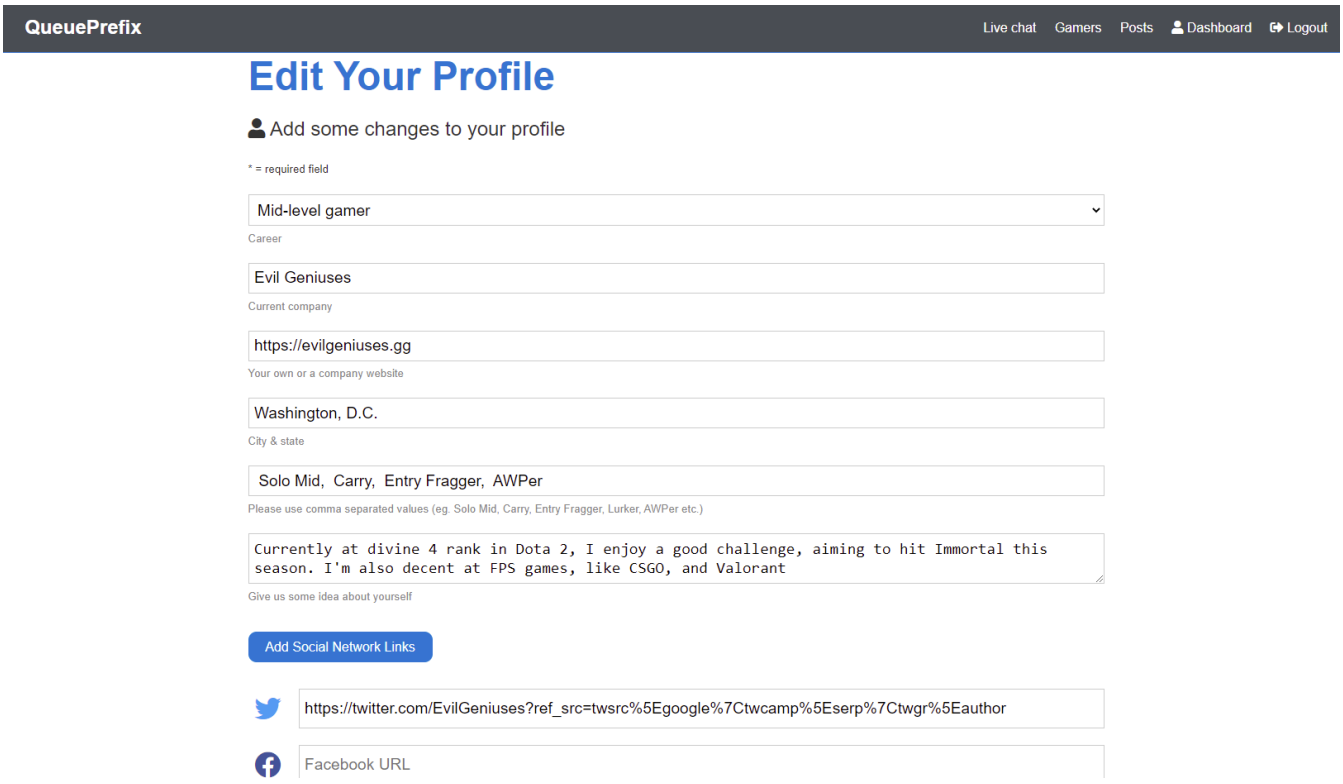
Momenat kada se korisnik registrira, on je preusmjeren na „Dashboard“ (kontrolna ploča) koji je vidljiv na slici 6 gdje si može napraviti svoj profil, dodati iskustvo i edukaciju. Osim toga korisnik također može obrisati dodano iskustvo i edukaciju ili svoj cijeli profil.



Slika 6. Prikaz sučelja za kontrolu podataka o profilu.

5.3. Uređivanje profila

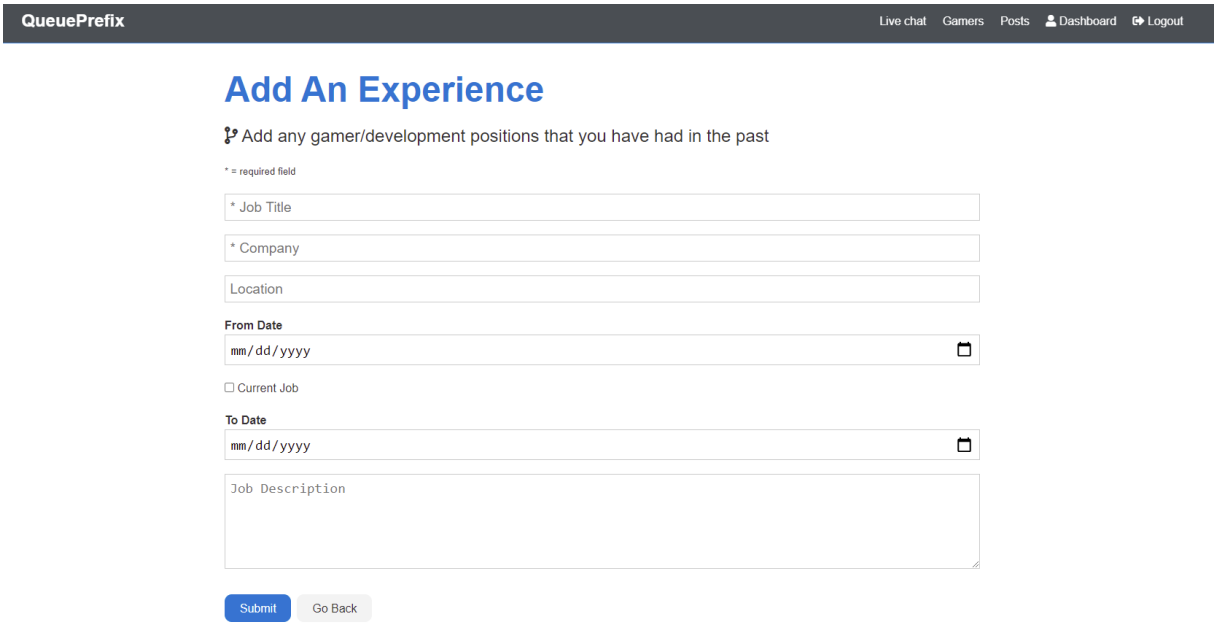
Korisnik može urediti svoj profil po izboru ili ga nadograditi što vidimo na slici 7. Potrebno je upisati njegov trenutni status karijere, firmu ako je zaposlenik ili vlasnik firme, lokacija, određene pozicije koje preferira igrajući igre, kratak opis o sebi, te veze socijalnih mreža. Navedene informacije su jako bitno, jer pomoću njih dobivamo dublji uvid s čime se korisnik bavi i automatski će ostali korisnici znati sebi filtrirati je li žele imati ikakvu kolaboraciju s njime, bilo to iz zabave ili posla.



Slika 7. Stranica za punjenje osnovnih informacija profila.

5.4. Dodavanje iskustva

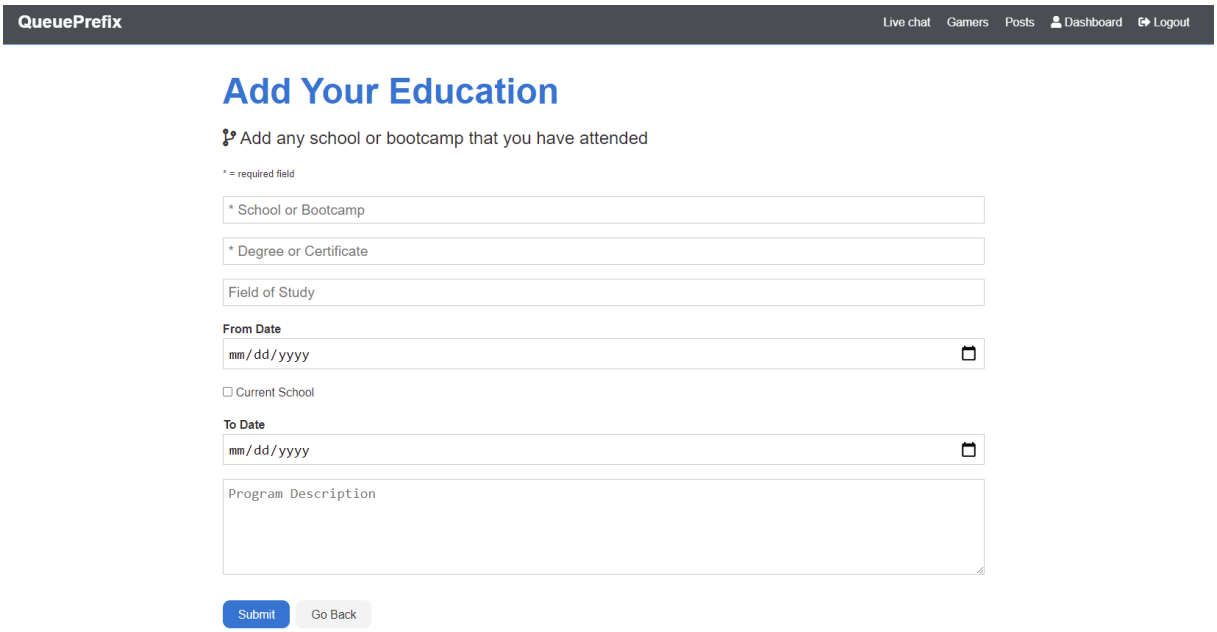
Na slici 8 vidimo da korisnik može dodati iskustvo. U slučaju da je korisnik kompetitivan igrač ili ako uz igranje igara se također bavi i nekakvim razvojem igara, bilo kakvo iskustvo si može nadodati na sljedećoj formi. Potrebno je ispuniti formu koja opisuje njegovu titulu, firmu, lokaciju, datume i kratak opis. Dosta igrača koji igraju iz zabave imaju općeniti interes za video igre, stoga možemo pretpostaviti da određeni igrači su programeri ili slično, pa će korisnici ovime dobiti uvid je li netko također zainteresiran za izradu igara, a ne samo igranje igara.



Slika 8. Stranica za punjenje informacija koje definiraju iskustvo.

5.5. Dodavanje edukacije

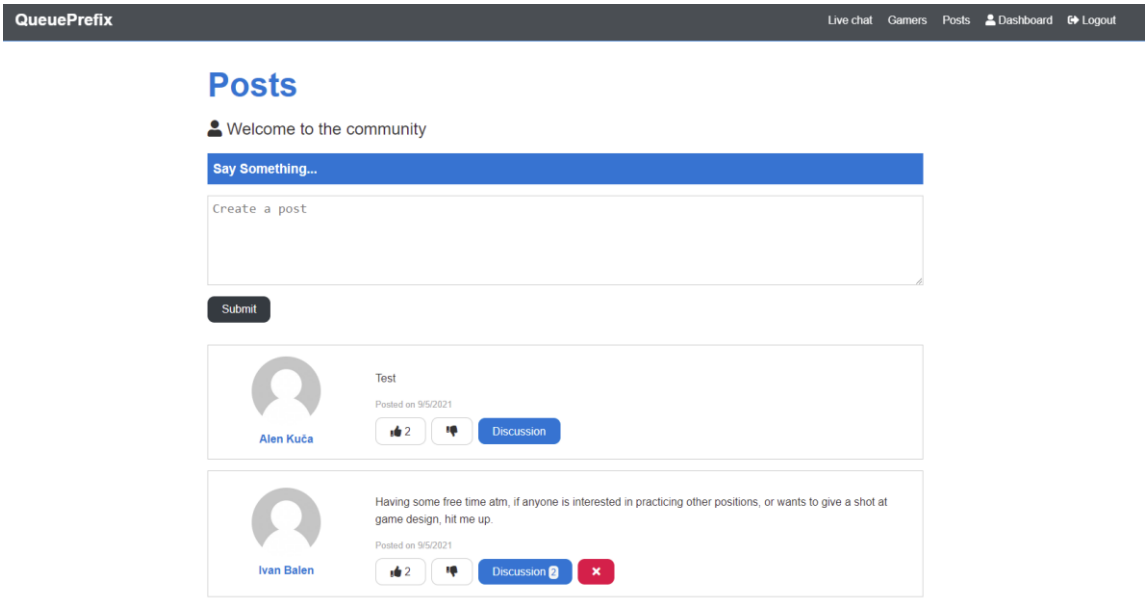
Kako su profesionalni igrači postali sve zastupljeniji u pravom svijetu i kako se počinju identificirati kao atletičari, isto tako su osnovani razni fakulteti orijentirani na video igre i razvoj video igara, sljedeća forma omogućava nadopunu tih informacija. Forma koju vidimo na slici 9 služi korisniku da opiše školu, stupanj diplome i područje proučavanja, uključujući datume trajanja tih procesa, te sam opis cjelokupnog programa edukacije.



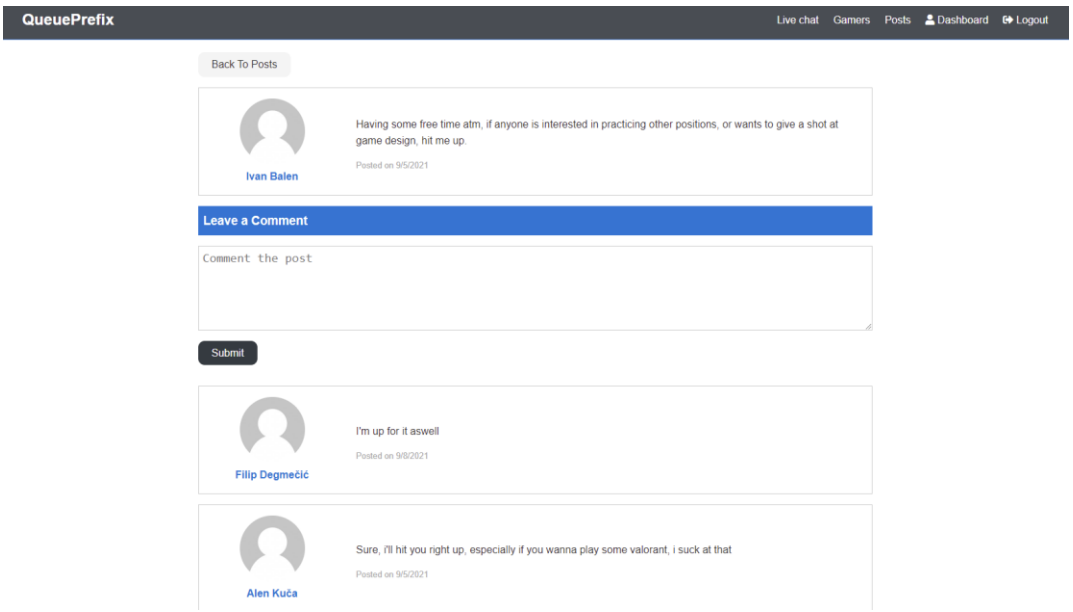
Slika 9. Stranica za punjenje informacija o edukaciji.

5.6. Objave

Korisnici mogu komunicirati preko postova (objava), na objavama možemo vidjeti teme koje su otvorene, te također pokrenuti diskusiju oko te teme. Kao što vidimo, na slici 10 je prikaz svih trenutnih objava, klikom na palac gore ili dolje možemo glasati za i protiv te teme, a pod opcijom „Discussion“ (rasprava) korisnik može ući u dublju raspravu s ostalim sudionicima, vidljivo na slici 11.



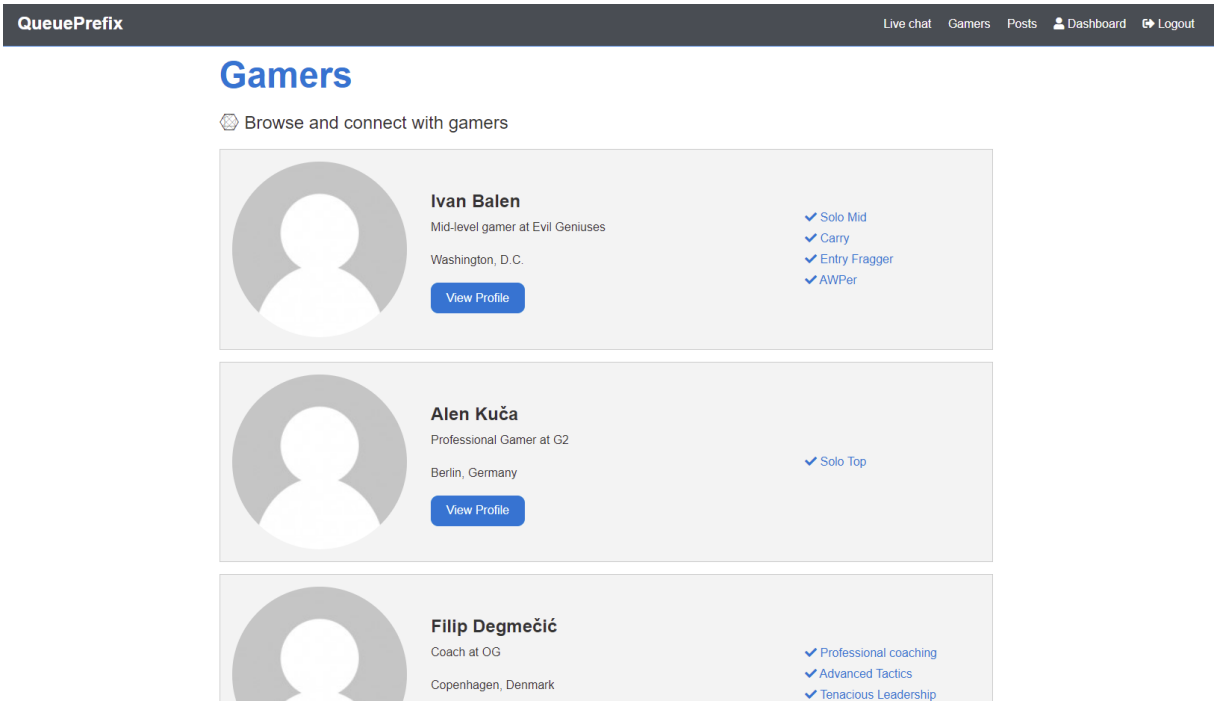
Slika 10. Stranica koja prikazuje postove.



Slika 11. Stranica koja prikazuje diskusiju određenog posta.

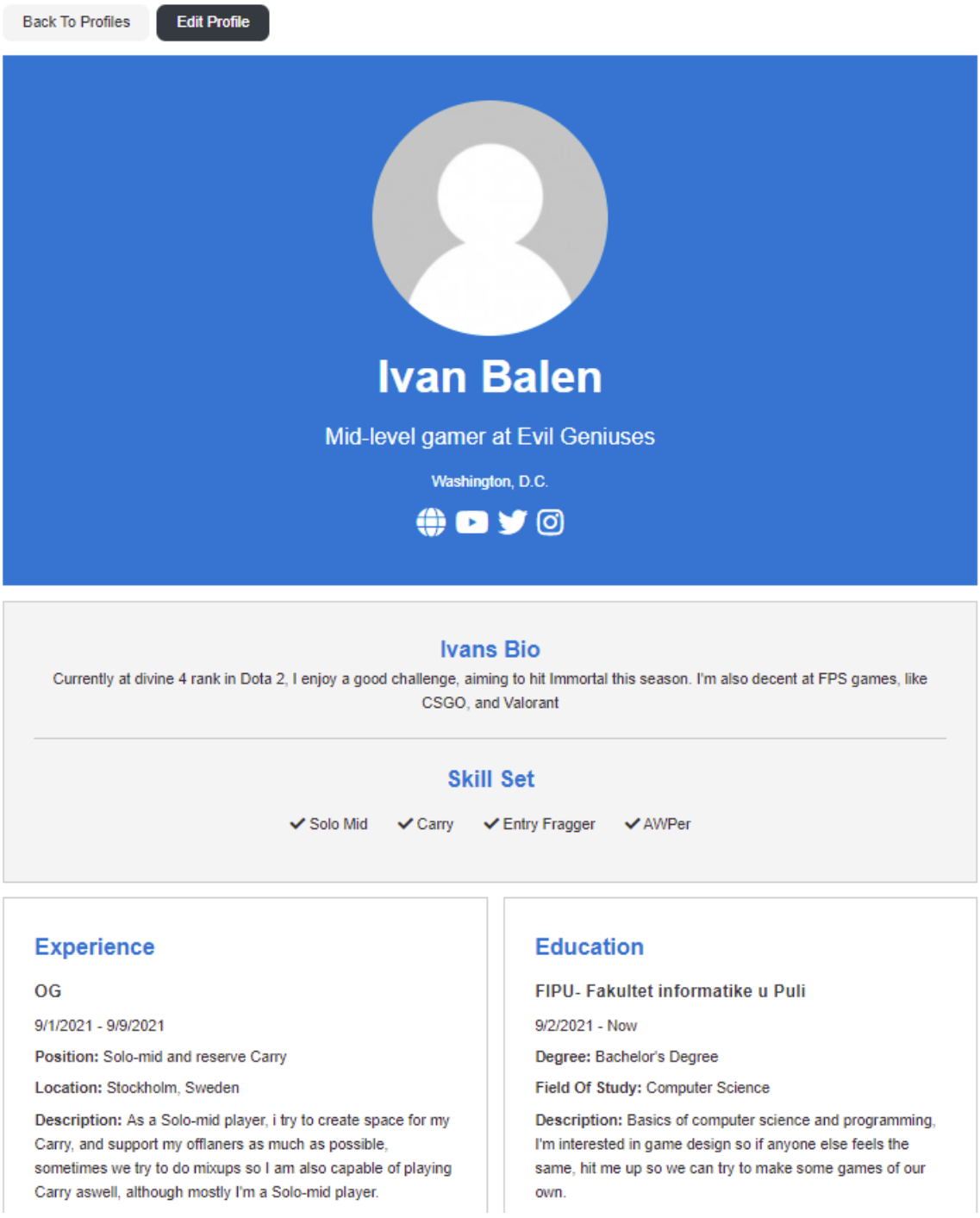
5.7. Igrači

Ovdje korisnici mogu vidjeti popis svih registriranih korisnika i njihov skraćeni opis koji uključuje njihovu trenutnu poziciju, lokaciju i područja u kojima su najviše istaknuti. Ideja ovoga je da korisnik dobije nekakav uvid tko je u čemu najiskusniji i njegove glavne kvalitete, samim time je njemu to dovoljna količina informacija da on sebi filtrira je li on želi igrati nešto s tim igračem, u slučaju da želi detaljniji opis svega vezano za odabranog korisnika, klikom na „View Profile“ (pogledaj profil) može dobiti detaljniji opis njegovog profila.



Slika 12. Stranica koja prikazuje popis registriranih korisnika.

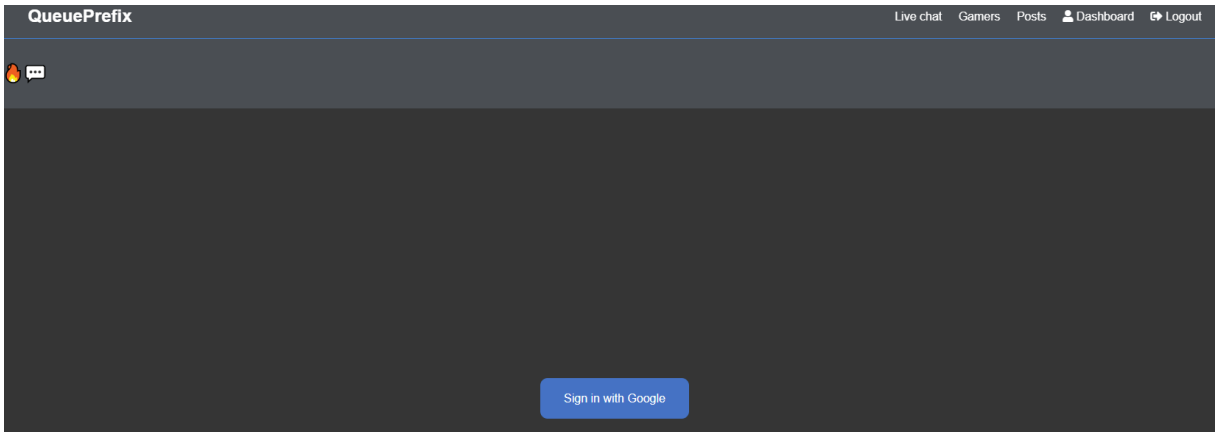
Klikom na „View Profile“ (pogledaj profil), je usmjeren na detaljan opis profila odabranog korisnika sa slike 13. Tamo se nalazi popis njegovog iskustva, njegova kratka biografija, edukacija, te veze na ostale socijalne mreže. Ovdje se nalaze sve informacije koje je korisnik naveo o sebi sa slike 7, 8 i 9. Sve su ovo krucijalne informacije koje pomažu drugim korisnicima da dobiju što bolji uvid u to tko se s čime bavi i tko ima kakve ukuse vezano za igre.



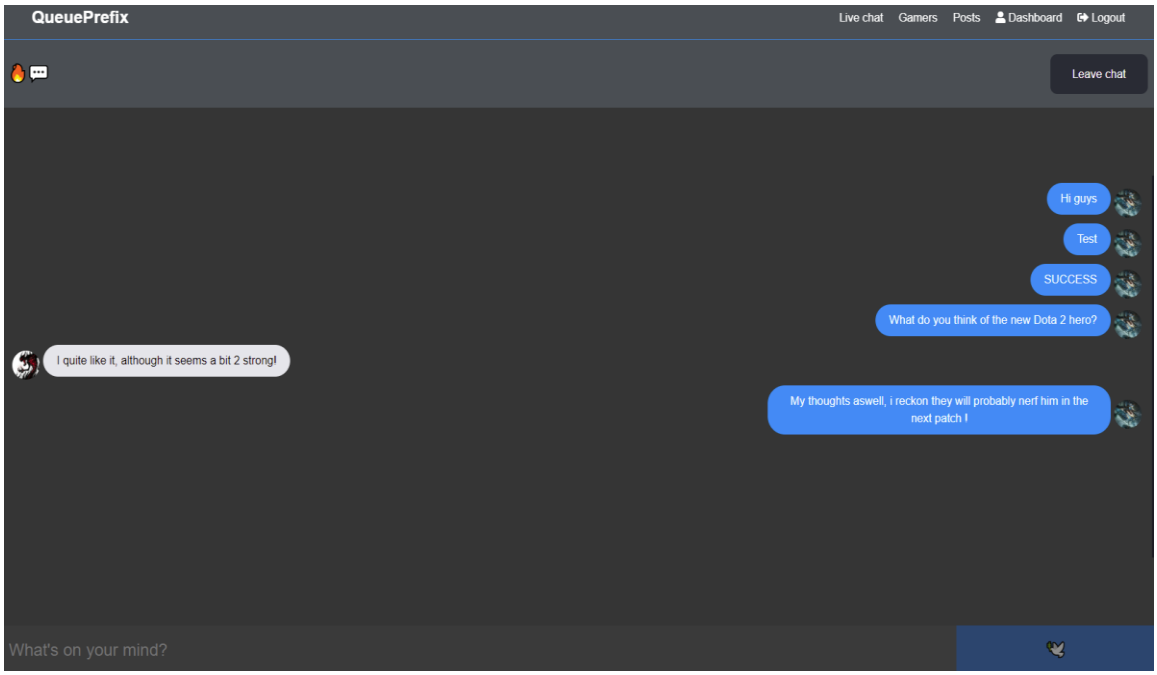
Slika 13. Stranica koja prikazuje detaljan opis profila odabranog korisnika.

5.8 Chat uživo

Posljednja funkcionalnost aplikacije je mogućnost komuniciranja s ostalim korisnicima preko chata koji je uživo postavljen. Kada korisnik otvori Live chat (chat uživo) koji je prikazan na slici 14, potrebno je da se prijavi sa svojim google računom i nakon toga je otvoren za raspravu s ostalim korisnicima. Korisnik može izaći iz chata što rezultira da se automatski odjavi sa svog računa koji koristi za chat, a može mu i ostati upaljen ako ode na drugi tab. Korisnikove poruke su pokazane s desne strane, a poruke ostalih sudionika su pokazane s lijeve strane.



Slika 14. Prikaz prijave na chat.



Slika 15. Stranica koja prikazuje Live chat.

6. Implementacija

Aplikacija je izgrađena pomoću React.js frontend biblioteke, a baza aplikacije je izgrađena pomoću MongoDB i Express.js.

6.1. MongoDB

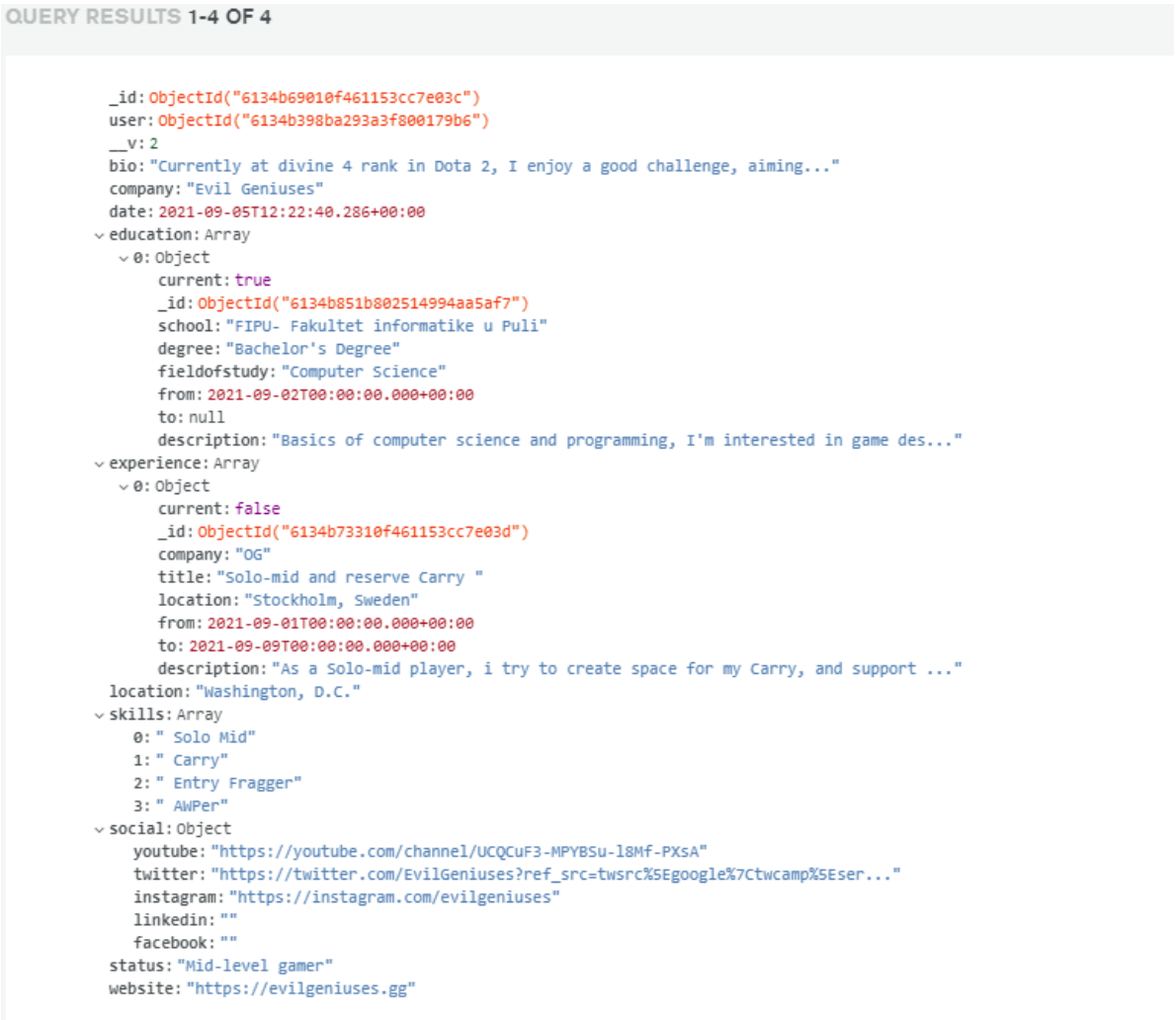
MongoDB je baza podataka izgrađena na proširenoj arhitekturi koja je postala popularna među programerima svih vrsta koji grade skalabilne aplikacije pomoću agilnih metodologija [1]. MongoDB je napravljen za ljude koji izgrađuju internetske i poslovne aplikacije koje se trebaju brzo razvijati i elegantno skalirati. Tvrtke i razvojni timovi svih veličina koriste MongoDB jer njegov model podataka je snažan i njegov način dohvaćanja podataka omogućuje razvojnim programerima brzo kretanje. Horizontalna, opsežna arhitektura MongoDB-a može podržati ogromne količine podataka i prometa. MongoDB se može svugdje koristiti i to na više načina, kao npr. besplatno putem izdanja zajednice otvorenog koda, u najvećim podatkovnim centrima zbog izdanja licence ili u bilo kojem oblaku kroz MongoDB Atlas. On je razvio veliki i zreli ekosustav za platforme, što znači da ima svjetsku zajednicu programera i konzultanata, pa je lako dobiti pomoć. MongoDB radi na svim vrstama računalnih platformi i lokalno i u oblaku (poput AWS, Azure i Google Cloud). Jezik se može prebaciti na više svjetskih jezika i također ima podršku na razini poduzeća. Razlog zašto se treba koristiti MongoDB je zato što programer ide dalje i brže pri razvoju softverskih aplikacija gdje mora obrađivati sve vrste podataka na skalabilan način. Tisuće tvrtki kao što su Bosch, Barclays i Morgan Stanley vode svoja poslovanja na MongoDB-u i koriste ga za upravljanje svojih najzahtjevnijih aplikacija u područjima poput IoT, igara, logistike, bankarstva, e-trgovine i upravljanja sadržajem. Izvrstan je izbor ako trebate predstavljati podatke s prirodnim klasterima i varijabilnošću tijekom vremena ili u svojoj strukturi podržavate brzi iterativni razvoj. Omogućuje suradnju velikog broja timova, povećava promet čitanja i pisanja na visokoj razini. Razvija vrstu implementacija kako se poslovanje mijenja, pohranjuje i upravlja i pretražuje podatke s tekstualnim, geoprostornim ili dimenzijskim vremenskim serijama. MongoDB kao tvrtka je narasla jer broj slučajeve upotrebe s ovim karakteristikama neprestano raste. MongoDB je pionir onoga što se počelo nazivati NoSQL bazama podataka, koje su se

razvile jer RDBMS sustavi su temeljeni na SQLu i nisu podržavali opseg ili brze razvojne cikluse potrebne za stvaranje modernih aplikacija.

NoSQL je pojam koji uključuje baze podataka orijentirane na dokumente poput MongoDB-a, stupčaste baze podataka, baze podataka u memoriji itd. Dokumenti u MongoDB-u su datoteke JSON i BSON. Razlog zašto je JSON praktičan je zato što je to prirodan oblik pohrane podataka, čitkiv je za čovjeka, strukturirane i nestrukturirane informacije mogu se pohraniti u isti dokument, te se može ugnijezditi za pohranu složenih podatkovnih objekata. JSON također ima fleksibilnu i dinamičnu šemu, pa dodavanje polja ili izostavljanje polja nije problem. Ono što je možda najvažnije, struktura podataka je pod kontrolom programera. Programeri prilagođavaju i formatiraju bazu podataka kako se aplikacija razvija bez pomoći administratora baze podataka. Po potrebi, MongoDB može koordinirati i kontrolirati promjene u strukturi dokumenata pomoću provjere šeme. MongoDB je stvorio binarni JSON format (BSON) kako bi povećao učinkovitost i podržao više vrsta podataka. Podaci pohranjeni u BSON-u mogu se pretraživati i indeksirati, što značajno povećava performanse. MongoDB podržava širok izbor metoda indeksiranja, uključujući tekstualni, decimalni, geoprostorni i djelomični način. Geoprostorni je dodan da se dokumenti mogu ispitivati prema lokaciji. MongoDB je uvijek posvećivao vrijeme i energiju kako bi osigurao da programeri imaju ugodno iskustvo dok ga koriste. Programeri su oduvijek voljeli JSON jer je to jednostavan i moćan način za opisivanje i pohranu podataka. Ne samo da možete pohraniti podatke koji izgledaju kao redci i stupci, već možete i ugraditi dokumente jedan u drugi, tehnika koja omogućuje spremanje i dohvat složenih podatkovnih objekata. Kad jednom znate opisati podatke u JSON-u, baza podataka predstavlja manji problem. Jednostavan je za instalaciju i programerima se sviđa činjenica da se MongoDB pobrinuo da se baza podataka može koristiti s raznim programskim jezicima, uključujući: C, C# i .NET, C++, Erlang, Haskell, Java, JavaScript, Perl, PHP itd. Kako se sve više poslovnih korisnika pridružilo MongoDB zajednici, dodane su značajke koje podržavaju korištenje i rad MongoDB-a u IT odjelima poduzeća, uključujući prvorazrednu podršku.

MongoDB atlas temeljna je baza podataka u središtu MongoDB oblaka. On omogućuje programerima da odmah počnu raditi u bilo kojem od javnih oblaka i lako migriraju instalacije MongoDB-a na lokalnom nivou u oblak. Opsežna arhitektura MongoDB-a distribuira posao na mnogim manjim i jeftinijim računalima, znači da

možemo stvoriti aplikaciju za koju smo sigurni da će se nositi s naglim prometom kako poslovanje raste. Inženjerske inovacije podržavaju ogroman broj čitanja i pisanja kod MongoDB-a. U središtu ovih inovacija je MongoDB-ov pristup shardingu (usitnjavanju) koji omogućuje pohranjivanje skupina informacija dok se informacije šire po računalu. Za usporedbu, većina SQL baza podataka koristi proširenu arhitekturu koja je ograničena jer se oslanja na stvaranje bržih i snažnijih računala. MongoDB također podržava transakcije baze podataka koje omogućuju da se mnoge promjene u bazi podataka grupiraju zajedno ili da se naprave ili odbiju u paketu. Ovo je jedna od najvažnijih značajki za podršku naprednim aplikacijama. Ono što je također praktično kod MongoDB-a je način na koji se prikazuju podaci u bazi.



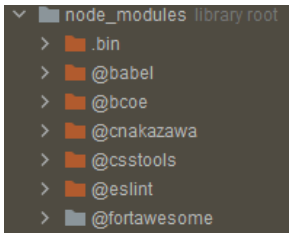
Slika 16. Prikaz izgleda MongoDB baze podataka za profil korisnika.

6.2. Node.js

Node.js je platforma izgrađena za vrijeme izvođenja JavaScripta, radi jednostavne izrade brzih i skalabilnih aplikacija [2]. Node.js koristi I/O model koji ne blokira događaje, što ga čini laganim i učinkovitim, savršenim za podatkovno intenzivne aplikacije u stvarnom vremenu koje se izvode na distribuiranim uređajima. Node.js je otvoreno okruženje za rad na više platformi za razvoj aplikacija na strani poslužitelja i umrežavanja. Node.js aplikacije napisane su u JavaScriptu i mogu se izvoditi unutar vremena izvođenja Node.js na OS X, Microsoft Windows i Linux. Node.js također nudi bogatu biblioteku različitih JavaScript modula što u velikoj mjeri pojednostavljuje razvoj web aplikacija pomoću Node.js-a.

Vrijedno je za napomenuti da postoje bitne značajke koje čine Node.js prvim odabirom softverskih arhitekata. Sve API biblioteke Node.js su asinkrone, odnosno ne blokiraju se. To u biti znači da poslužitelj zasnovan na Node.js nikada ne čeka da API vrati podatke. Poslužitelj prelazi na sljedeći API nakon što ga pozove, a mehanizam obavijesti pomaže poslužitelju da dobije odgovor iz prethodnog API poziva. Biblioteka Node.js-a je izgrađena na Google Chrome V8 JavaScript mašini, vrlo je brza u izvršavanju koda. Pojedinačni, ali visoko skalabilan Node.js koristi model s jednim ponavljanjem događaja. Mehanizam događaja pomaže poslužitelju da odgovori na način koji ne blokira i čini poslužitelja visoko skalabilnim za razliku od tradicionalnih poslužitelja koji stvaraju ograničene niti za obradu zahtjeva.

Node.js koristi jedan program s navojem i isti program može pružiti uslugu mnogo većem broju zahtjeva od tradicionalnih poslužitelja poput Apache HTTP poslužitelja. Node.js aplikacije nikada ne spremaju podatke u međuspremnik. Ove aplikacije jednostavno ispisuju podatke u komadima, a što se tiče licence, licenca je objavljena pod MIT licencom. Osim Node.js, u priču ulazi i NPM, on je upravitelj paketa Node.js-a. Svaka aplikacija će imati nekakve posebne pakete ili biblioteke koje će biti potrebno instalirati i preko Node.js-a, možemo točno to napraviti, u konzoli komandom „npm install“ ćemo dobiti osnovni paket modula Node.js-a.



Slika 17. Prikaz nekoliko paketa nakon što napravimo „npm install“.

6.3. Express.js

Express je najpopularnije Node.js web okruženje i temeljna je biblioteka za brojna druga popularna okruženja [3]. Pruža mehanizme za obrađivanje zahtjeva s različitim HTTP aktivnostima i komandama na različitim URL putanjama (rutama), postavljanje uobičajenih postavki web aplikacija kao što su port koji će se koristiti za povezivanje i mjesto predložaka koji se koriste za generiranje odgovora. Osim toga, koristimo „posrednički softver“ za obradu zahtjeva u bilo kojoj točki unutar dretvi za obradu zahtjeva. Iako je Express.js sam po sebi minimalistički, programeri su stvorili kompatibilne pakete međuopreme za rješavanje gotovo svih problema razvoja weba.

Postoje biblioteke za rad s kolačićima, sesijama, prijavama korisnika, parametrima URL-a, POST podacima, sigurnosnim zaglavljima i mnogim drugim. Web okruženja često se referenciraju na mišljene i nepromišljene. Mišljena okruženja su oni koji misle o pravom putu za rješavanje bilo kojeg određenog zadatka. Često podržavaju brzi razvoj u određenoj domeni (rješavanje problema određene vrste) jer je pravi način da se bilo što učini obično dobro razumljiv i dobro dokumentiran. Međutim oni mogu biti manje fleksibilni u rješavanju problema izvan svoje glavne domene i imaju tendenciju ponuditi manje izbora za komponente i pristupe koje mogu koristiti. Nasuprot tome, nepromišljena okruženja imaju daleko manje ograničenja u pogledu najboljeg načina lijepljenja komponenti radi postizanja cilja. Oni programerima olakšavaju korištenje najprikladnijih alata za izvršavanje određenog zadatka iako uz cijenu koja je potrebna da sami pronađemo te komponente. Express.js pripada ovoj drugoj kategoriji, on može umetnuti gotovo bilo koji kompatibilni posrednički softver u lanac za obradu zahtjeva, bilo kojim redoslijedom. Aplikaciju možemo konstruirati tako da bude u jednoj datoteci pomoću bilo kakve strukture direktorija. Na tradicionalnoj web stranici na temelju podataka web aplikacija čeka HTTP zahtjeve od web preglednika (ili drugog klijenta). Kada primi zahtjev, aplikacija utvrđuje koje su radnje potrebne na temelju uzorka URL-

a i eventualno povezanih informacija sadržanih u POST podacima ili GET podacima. Ovisno o tome što je potrebno, tada može čitati ili pisati podatke iz baze podataka ili izvršavati druge zadatke potrebne za udovoljavanje zahtjevu. Aplikacija će tada vratiti odgovor web pregledniku, često dinamički stvarajući HTML stranicu za preglednik tako da umetne preuzete podatke u rezervirana mjesta u HTML predlošku. Express nudi metode za navođenje koje sve funkcije se pozivaju za određeni HTTP zahtjev i pruža URL uzorak tih ruta za metode koje služe za navođenje. Prvo što se zahtijeva je ekspresni modul i stvaranje Express aplikacije. Ovaj objekt, koji se tradicionalno naziva aplikacijom, ima metode za usmjeravanje HTTP zahtjeva, konfiguriranje posredničkog softvera, generiranje HTML prikaza, registriranje stroja predložaka i mijenjanje postavki aplikacija koje kontroliraju ponašanje aplikacije (npr. način okruženja, jesu li definicije rute osjetljive na velika i mala slova itd.).

```
const express = require('express');
const connectDB = require('./config/db');
const path = require('path');

const app = express();


connectDB();


app.use(express.json());


app.use('/api/users', require('./routes/api/users'));
app.use('/api/auth', require('./routes/api/auth'));
app.use('/api/profile', require('./routes/api/profile'));
app.use('/api/posts', require('./routes/api/posts'));


if (process.env.NODE_ENV === 'production') {

  app.use(express.static('client/build'));

  app.get('*', (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> , res : Response<ResBody, Locals> ) => {
    res.sendFile(path.resolve(__dirname, 'client', 'build', 'index.html'));
  });
}

const PORT = process.env.PORT || 5000;

app.listen(PORT, callback: () => console.log(`Server started on port ${PORT}`));
```

Slika 18. Prikaz konekcije na bazu, definiranje ruta, korištenje posredničkog softvera.

Srednji dio koda sa slike 18 koji počinje sa „app.get“ prikazuje definiciju rute. Metoda „app.get“ navodi funkciju povratnog poziva koja će se dozvati svaki puta kada postoji HTTP GET zahtjev s rutom ('/') u odnosu na korijen web lokacije. Funkcija povratnog poziva uzima zahtjev i objekt odgovora kao argumente, te poziva „sendFile“ na odgovor da vrati podatak. Finalni blok pokazuje specifični port na kojemu je adresa.

Modul je JavaScript biblioteka koju možemo uvesti u drugi kod pomoću funkcije „Node require()“. Express je sam po sebi modul, kao i sav posrednički softver i baza podataka koju koristimo u našim Express aplikacijama, vidljivo na slici 19.

```
const mongoose = require('mongoose');
const config = require('config');
const db = config.get('mongoURI');

const connectDB = async () => {
    try {
        await mongoose.connect(db, options: {
            useNewUrlParser: true,
            useCreateIndex: true,
            useFindAndModify: false,
            useUnifiedTopology: true
        });

        console.log('MongoDB Connected...');
    } catch (err) {
        console.error(err.message);
        process.exit( code: 1);
    }
};

module.exports = connectDB;
```

Slika 19. Definiranje konekcije baze.

JSON web znak je predloženi internetski standard za stvaranje podataka s opcionalnim potpisom i/ili izbornom enkripcijom čiji korisni teret sadrži JSON koji potvrđuje određeni broj zahtjeva. Znakovi se potpisuju ili privatnom tajnom ili javnim/privatnim ključem. Na primjer, poslužitelj bi mogao generirati znak koji ima zahtjev „prijavljen kao administrator“ i dostaviti to klijentu. Klijent bi tada mogao upotrijebiti taj znak da dokaže da je prijavljen kao administrator. Znakovi se mogu

potpisati privatnim ključem jedne strane (obično poslužiteljskim) tako da ta strana može naknadno provjeriti je li token (znak) ispravan. Ako druga strana, na neki prikladan i pouzdan način posjeduje odgovarajući javni ključ i ona može provjeriti legitimnost znaka. Znakovi su dizajnirani da budu kompaktni, sigurni za URL i upotrebljivi, osobito u kontekstu jednokratne prijave u web pregledniku.

JWT potraživanja obično se mogu koristiti za prosljeđivanje identiteta autentificiranih korisnika između pružatelja identiteta davatelja usluga ili bilo koje druge vrste zahtjeva kako to zahtijevaju poslovni procesi. Slika 20 nam prikazuje uporabu JWT-a tako što preuzimamo znak iz zaglavlja, provjeravamo ako je ili nije znak radi validacije i koristimo posredni softver sa slike 21 koji provjerava je li ima valjani ID objekta.

```
const jwt = require('jsonwebtoken');
const config = require('config');

module.exports = function (req, res, next) {

  const token = req.header('x-auth-token');


  if (!token) {
    return res.status(401).json({ msg: 'No token, authorization denied' });
  }


  try {
    jwt.verify(token, config.get('jwtSecret'), options: (error, decoded) => {
      if (error) {
        return res.status(401).json({ msg: 'Token is not valid' });
      } else {
        req.user = decoded.user;
        next();
      }
    });
  } catch (err) {
    console.error('something wrong with auth middleware');
    res.status(500).json({ msg: 'Server Error' });
  }
};
```

Slika 20. JSON Web Token.

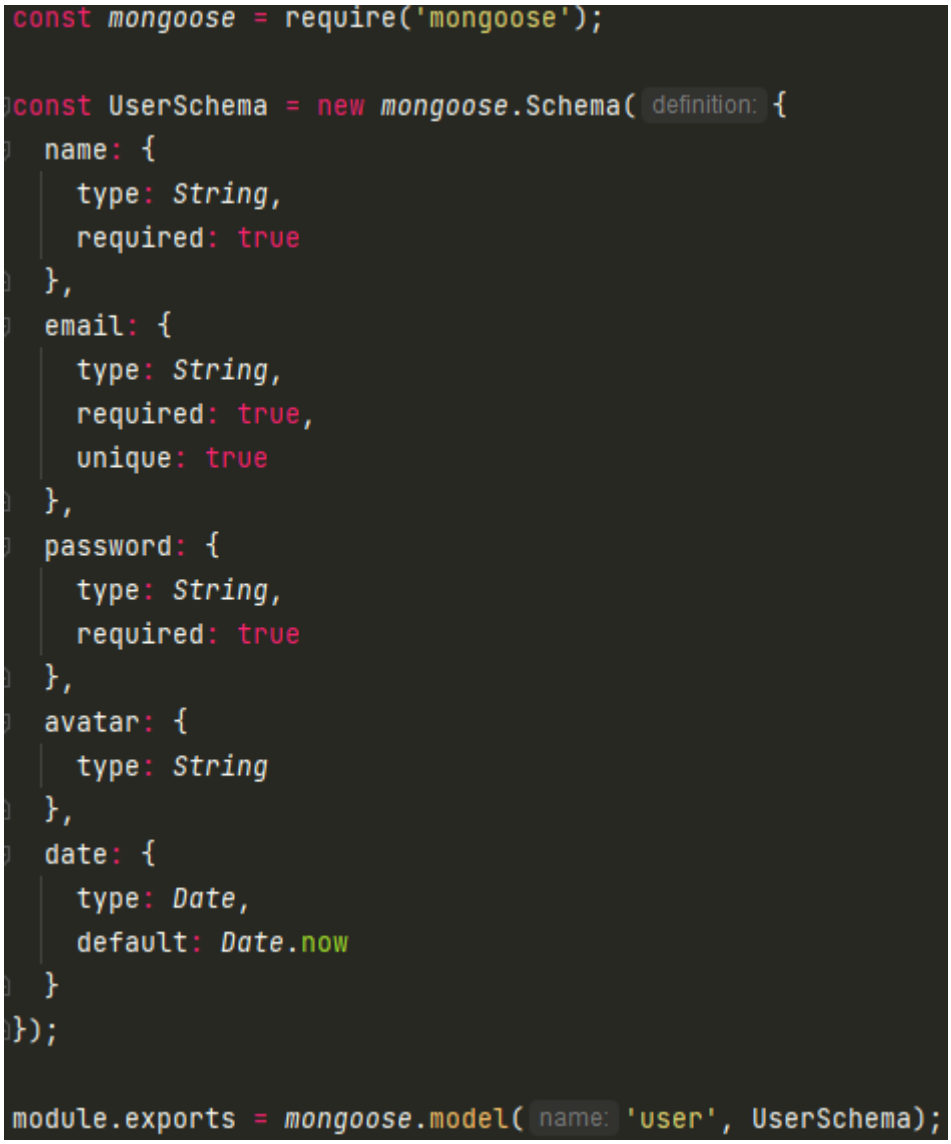
```
const mongoose = require('mongoose');
const checkObjectId = (idToCheck) => (req, res, next) => {
  if (!mongoose.Types.ObjectId.isValid(req.params[idToCheck]))
    return res.status(400).json({ msg: 'Invalid ID' });
  next();
};

module.exports = checkObjectId;
```

Slika 21. Validacija ID objekta.

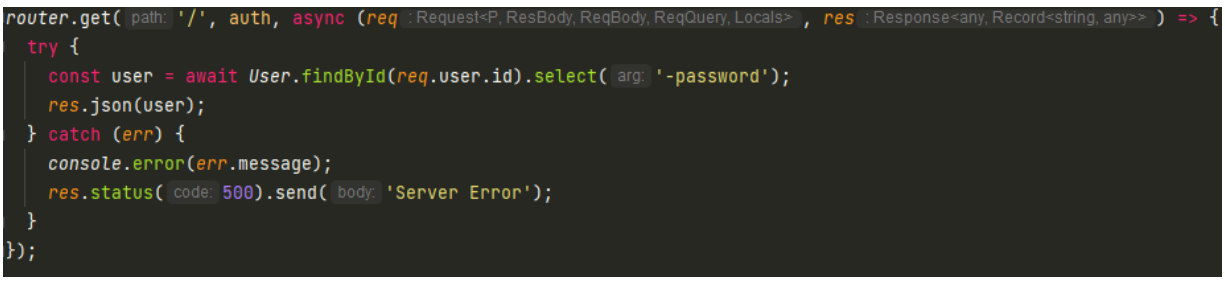
API je kratica za Application Programming Interface, koja je softverski posrednik koji omogućuje da dvije aplikacije međusobno razgovaraju [4]. Svaki put kada koristite aplikacije poput Facebook-a, pošaljete trenutnu poruku ili provjerite vrijeme na telefonu, koristite API. Aplikacija se povezuje s internetom i šalje podatke poslužitelju. Poslužitelj zatim preuzima te podatke, tumači ih, izvodi potrebne radnje i šalje ih natrag. Aplikacija zatim tumači te podatke i predstavlja željene podatke na čitljiv način, dakle sve se to događa putem API-a.

Tijekom godina, ono što je API često opisivalo je bilo kakvo generičko sučelje za povezivanje s aplikacijom. No, u novije vrijeme moderni API-ji poprimili su neke karakteristike koje ih čine iznimno vrijednima i korisnima. Moderni API-ji pridržavaju se standarda (obično HTTP i REST) koji su prilagođeni programerima, lako dostupni i razumljivi općenito. Tretiraju se više kao proizvodi nego kao kod. Dizajnirani su za potrošnju za određenu publiku (npr. Razvojne programere mobilnih uređaja), dokumentirani su i verzionirani tako da korisnici mogu imati određena očekivanja od njegova održavanja i životnog ciklusa. Budući da su mnogo standardiziraniji, imaju mnogo jaču disciplinu za sigurnost i upravljanje, kao i da se nadziru i upravljaju prema učinku i opsegu. Kao i svaki drugi komad produciranog softvera, suvremeni API ima vlastiti životni ciklus razvoja softvera, testiranja, izgradnje, upravljanja i izdavanja verzija. Također moderni API je dobro dokumentiran za potrošnju i izradu verzija. Prije nego što riješimo API, potrebno je također napraviti model po kojemu ćemo modificirati naše podatke i kod, slika 22 pokazuje primjer modela korisnika.



Slika 22. Prikaz modela korisnika.

Nakon što smo definirali model korisnika, idući korak je da omogućimo našem API-u da on zahtjeva taj model, kako bi ga mogli proslijediti i kako bi napravili odgovarajuće HTTP metode za korisnika. Slike 23 i 24 će pokazati kako izgleda HTTP GET i POST metoda za korisnikov račun.



Slika 23. HTTP GET metoda.

U ovom slučaju, imamo blok koji pokuša naći korisnika po njegovom identifikacijskom broju i ako ga nađe, uzme njegovu registracijsku lozinku i vraća nam natrag njegove podatke u obliku JSON-a.

```
const express = require('express');
const router = express.Router();
const gravatar = require('gravatar');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const config = require('config');
const { check, validationResult } = require('express-validator');
const normalize = require('normalize-url');

const User = require('../../models/User');

router.post(
  path: '/',
  check( fields: 'name',  message: 'Name is required').notEmpty(),
  check( fields: 'email',  message: 'Please include a valid email').isEmail(),
  check(
    fields: 'password',
    message: 'Please enter a password with 6 or more characters'
  ).isLength( options: { min: 6 }),
  async (req : Request<{[p: string]: any}, any, any, {[p: string]: any}, Record<string, any>> , res : Response<any, Record<string, any>> ) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status( code: 400).json( body: { errors: errors.array() });
    }
```

Slika 24. HTTP POST metoda.

Osim GET i POST metoda, također se koristi DELETE metoda kod HTTP-a, slika 25 će ilustrirati HTTP zahtjev za brisanje korisnikovog profila.

```
router.delete( path: '/', auth, async (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> , res : Response<any, Record<string, any>> ) => {
    try {

        await Promise.all( values: [
            Post.deleteMany( filter: {user: req.user.id}),
            Profile.findOneAndRemove( filter: {user: req.user.id}),
            User.findOneAndRemove( filter: {_id: req.user.id})
        ]);

        res.json( body: {msg: 'User deleted'});
    } catch (err) {
        console.error(err.message);
        res.status( code: 500).send( body: 'Server Error');
    }
});
```

Slika 25. HTTP DELETE metoda za brisanje korisnikovog profila.

7. React.js

React je najpopularnija front-end JavaScript biblioteka u području web razvoja. Koriste ga velike tvrtke i novoosnovani startupi [5]. React donosi mnoge prednosti što ga čini boljim izborom od ostalih okruženja poput Angular.js. Služi za izgradnju brzih i interaktivnih korisničkih sučelja za web i mobilne aplikacije. Otvorenog je koda, zasnovan na komponentama odgovorne samo za sloj prikaza aplikacije. U arhitekturi Model View Controller (MVC), sloj pogleda odgovoran je za izgled i osjećaj aplikacije. React je stvorio Jordan Walke, softverski inženjer u Facebook-u. Današnja popularnost React-a zasjenila je popularnost svih ostalih front-end razvojnih okruženja. To je zato što on omogućava jednostavno stvaranje dinamičkih aplikacija jer zahtijeva manje kodiranja i nudi više funkcionalnosti. Poboljšane performanse su također jedna od stvari koje su prednosti korištenja React-a, dakle React ima svoj virtualni DOM čime se brže stvaraju web aplikacije.

Virtualni DOM uspoređuje prethodna stanja komponenti i ažurira samo stavke u stvarnom DOM-u koje su promijenjene, umjesto ponovnog ažuriranja svih komponenti, kao što to rade uobičajene web aplikacije. Komponente su gradivni elementi svake React aplikacije, a jedna se aplikacija obično sastoji od više komponenti. Ove komponente imaju svoju logiku i kontrole te se mogu ponovno koristiti u cijeloj aplikaciji, što zauzvrat dramatično skraćuje vrijeme razvoja aplikacije. Jednosmjerni tok podataka znači da razvojni programeri prilikom projektiranja React aplikacije često ugnijezde podređene komponente unutar roditeljskih komponenti. Budući da podaci teku u jednom smjeru, postaje lakše otklanjati pogreške i znati gdje se problem javlja u aplikaciji u danom trenutku.

React je lako naučiti jer uglavnom kombinira osnovne HTML i JavaScript koncepte s nekim korisnim dodacima. Ipak, kao što je slučaj s drugim alatima i okruženjima, potrebno je provesti neko vrijeme da bi se pravilno razumio. React-ova biblioteka se može koristiti za razvoj web i mobilnih aplikacija, no to nije sve što može učiniti. Postoji razvojno okruženje zvano React Native, izveden iz samog React-a, koji je iznimno popularan i koristi se za stvaranje prekrasnih mobilnih aplikacija. Dakle, u stvarnosti se React može koristiti za izradu web i mobilnih aplikacija. Namjenski alati za jednostavno ispravljanje pogrešaka su također jedna od prednosti korištenja React-a,

jer je Facebook objavio Chrome-ovo proširenje koje se može koristiti za ispravljanje pogrešaka u React aplikacijama. To čini proces ispravljanja puno bržim i lakšim.

JSX je proširenje sintakse za JavaScript. Koristi se s React-om za opisivanje korisničkog sučelja. Korištenjem JSX-a možemo zapisati HTML strukture u istu datoteku koja sadrži JavaScript kod. To čini kod lakšim za razumijevanje i ispravljanje pogrešaka jer izbjegava uporabu složenih JavaScript DOM struktura.

React čuva lagani prikaz „pravog“ DOM-a u memoriji, a to je poznato kao „virtualni“ DOM (VDOM). Manipuliranje stvarnim DOM-om mnogo je sporije od manipuliranja VDOM-om jer se ništa ne crta na ekranu. Kad se promijeni stanje objekta, VDOM mijenja samo taj objekt u stvarnom DOM-u umjesto ažuriranja svih objekata. Kad se stanje objekta promijeni u React aplikaciji, VDOM se ažurira. Zatim uspoređuje prethodno stanje, a zatim ažurira samo one objekte u stvarnom DOM-u umjesto ažuriranja svih objekata. To čini da se stvari kreću brzo, osobito u usporedbi s drugim naprednim tehnologijama koje moraju ažurirati svaki objekt čak i ako se samo jedan objekt promijeni u web aplikaciji. React fragmentira složeno korisničko sučelje na pojedinačne komponente, dopuštajući više korisnika da rade na svakoj komponenti istovremeno, čime se ubrzava vrijeme razvoja.

React-ovo jednosmjerno vezivanje podataka održava sve modularno i brzo. Jednosmjerni tok podataka znači da kada programer dizajnira React aplikaciju, često ugniježđuju podređene komponente unutar roditeljskih komponenti. Na taj način razvojni programer zna gdje i kada dolazi do pogreške, dajući im bolju kontrolu nad cijelom web aplikacijom.

Komponente su gradivni elementi svake React aplikacije, a jedna se aplikacija obično sastoji od više komponenti. Komponenta je u biti dio korisničkog sučelja. React dijeli korisničko sučelje za višekratnu uporabu koje se mogu zasebno obraditi. Postoje dvije vrste komponenti u React-u: funkcionalne i klasne komponente. Funkcionalne komponente su komponente koje nemaju vlastito stanje i sadrže samo metodu iscrtavanja pa se nazivaju i komponente bez stanja. One mogu izvesti podatke iz drugih komponenti kao svojstva (props). Klasne komponente su komponente koje mogu držati i upravljati svojim stanjem i imati zasebnu metodu iscrtavanja za vraćanje JSX-a na zaslon. Nazivaju se i sastavnim komponentama jer mogu imati stanje.

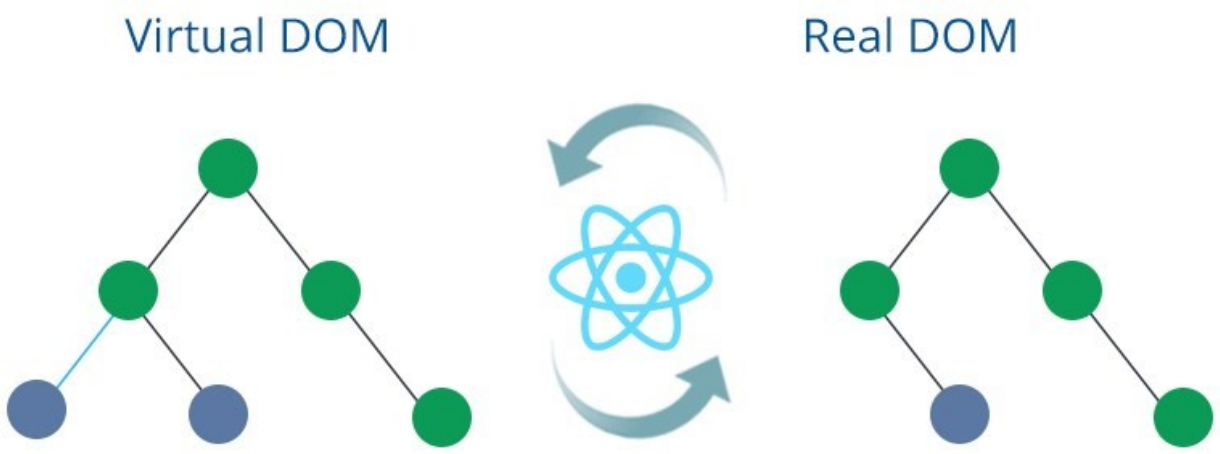
Stanje je ugrađeni React objekt koji se koristi za sadržavanje podataka ili informacija o komponenti. Stanje komponente može se promijeniti s vremenom, a kad god se promijeni, komponenta se ponovno prikazuje. Promjena stanja može se dogoditi kao odgovor na radnju korisnika ili događaje generirane sustavom, a te promjene određuju ponašanje komponente i način na koji će se ona prikazati.

Svojstva (props) su ugrađeni objekti koji pohranjuju vrijednosti atributa oznake i rade slično kao HTML atributi. Omogućuju način prijenosa podataka iz jedne komponente u drugu komponentu na isti način na koji se argumenti prenose u funkciji.

7.1. DOM i VDOM

DOM je kratica za Document Object Model. To je hijerarhijski prikaz web stranice ili aplikacije. Budući da je manipulacija DOM-om vrlo spora, praktično je koristiti VDOM. Jednostavnim riječima, virtualni DOM je kopija izvornog DOM-a koji se čuva u memoriji i sinkronizira s pravim DOM-om u bibliotekama poput ReactDOM-a. Taj se proces naziva reconciliation (pomirenje). Virtualni DOM ima ista svojstva kao i pravi DOM, ali nema moć izravne promjene sadržaja zaslona.

Ako zamislimo virtualni DOM kao nacrt stroja, promjene napravljene u nacrtu ne odražavaju se na sam stroj. Dakle, kada dođe do ažuriranja virtualnog DOM-a, React uspoređuje virtualni DOM sa snimkom virtualnog DOM-a snimljenom neposredno prije ažuriranja virtualnog DOM-a. Uz pomoć ove usporedbe React utvrđuje koje komponente u korisničkom sučelju treba ažurirati. Taj se proces naziva diferencijacija. Algoritam koji se koristi za proces diferencijacije naziva se algoritam diferencijacije. Nakon što React zna koje su komponente ažurirane, tada zamjenjuje izvorne DOM čvorove ažuriranim DOM čvorom. Na slici 26 vidimo razliku između virtualnog i stvarnog DOM-a [11].



Slika 26. Reprezentacija razlike virtualnog i pravog DOM-a [11].

7.2. Redux

Redux je predvidljiv spremnik stanja osmišljen kako bi pomogao u pisanju JavaScript aplikacija koje se dosljedno ponašaju u klijentskom, poslužiteljskom i izvornom okruženju i koje je lako testirati [6]. Iako se uglavnom koristi kao alat za upravljanje stanja uz React, možemo ga koristiti sa bilo kojim drugim JavaScript okruženjem ili bibliotekom. Korištenjem Redux-a se stanje aplikacije čuva u „trgovini“, a svaka komponenta može pristupiti bilo kojem stanju koje joj je potrebno iz trgovine.

U posljednje vrijeme jedna od najvećih rasprava u front-end svijetu vodi se o Reduxu. Nedugo nakon objavljivanja, Redux je postao jedna od vrućih tema rasprave. Mnogi ga podržavaju dok su drugi ukazali na probleme. Redux omogućuje da upravljamo stanjem aplikacije na jednom mjestu i održavamo promjene u aplikaciji i da te promjene budu predvidljive i sljedive. Olakšava zaključivanje o promjenama aplikacije. Sve te prednosti dolaze s kompromisima i ograničenjima. Upravljanje stanjem je način za olakšavanje komunikacije i razmjene podataka među komponentama. Ono stvara opipljivu strukturu podataka koja predstavlja stanje aplikacije iz koje možemo čitati i pisati. Tako možemo vidjeti inače nevidljiva stanja dok radimo s njima. Većina biblioteka i razvojnih okruženja su izgrađena tako da komponente upravljaju svojim stanjem bez ikakve potrebe za vanjskom bibliotekom ili alatom. Dobro se snalazi u aplikacijama s nekoliko komponenti, ali kako aplikacija raste, upravljanje stanjima podijeljenim po komponentama postaje kompleksno. U

aplikaciji u kojoj se podaci dijele među komponentama, može postati zbunjujuće da zaključimo gdje bi stanje trebalo postojati. U idealnom slučaju, podaci u komponenti trebaju biti u samo jednoj komponenti, pa dijeljenje podataka među komponenata postaje teško, npr. u Reactu za razmjenu podataka među sestrinskim komponentama, stanje mora živjeti u roditeljskoj komponenti. Nadređena komponenta pruža način ažuriranja ovog stanja i prosljeđuje se kao svojstvo tim komponentama. Ako zamislimo što se događa kada se stanje mora podijeliti između komponenti koje su udaljene u stablu, došli bi do zaključka da se stanje mora prenositi s jedne komponente na drugu sve dok ne dođe tamo gdje je potrebno.

U osnovi, stanje će se morati podizati do najbliže nadređene komponente i do sljedeće dok ne dođe do pretka koji je zajednički za obje komponente kojima je potrebno stanje, a zatim se prosljeđuje. Zbog toga je stanje teško održavati. To također znači da će se prosljeđivati podaci komponentama kojima to nije potrebno. Jasno je da upravljanje stanjem postaje neuredno kako se aplikacija komplicira. Zato je potreban alat za upravljanje stanja poput Redux-a koji olakšava održavanje ovih stanja.

Način na koji Redux radi je jednostavan. Postoji središnja trgovina koja sadrži cijelo stanje aplikacije. Svaka komponenta može pristupiti pohranjenom stanju bez slanja svojstava s jedne komponente na drugu. Postoje 3 glavna segmenta kod Reduxa, a to su: Akcije, trgovina i reduktori. Jednostavno rečeno, akcije su događaji. To je jedini način na koji možemo poslati podatke iz svoje aplikacije u Redux trgovinu. Podaci mogu biti iz korisničkih interakcija, API poziva ili čak slanja obrazaca. Akcije se šalju pomoću metode „store.dispatch()“. Akcije su obični JavaScript objekti i moraju imati svojstvo tipa koje označava vrstu radnje koju treba izvesti. Također moraju imati korisni teret koji sadrži podatke na kojima bi radnja trebala poraditi. Akcije se stvaraju putem kreatora akcija. Evo primjera akcije koja se može izvršiti kada se korisnik učitava, vidljivo sa slike 27.

```
import api from '../utils/api';
import { setAlert } from './alert';
import {
  REGISTER_SUCCESS,
  REGISTER_FAIL,
  USER_LOADED,
  AUTH_ERROR,
  LOGIN_SUCCESS,
  LOGIN_FAIL,
  LOGOUT
} from './types';


export const loadUser = () => async dispatch => {
  try {
    const res = await api.get( url: '/auth');

    dispatch({
      type: USER_LOADED,
      payload: res.data
    });
  } catch (err) {
    dispatch({
      type: AUTH_ERROR
    });
  }
};
```

Slika 27. Akcija za učitavanje korisnika.

Reduktori su čiste funkcije koje uzimaju trenutno stanje aplikacije, izvode akciju i vraćaju novo stanje. Ta su stanja pohranjena kao objekti i određuju kako se stanje aplikacije mijenja kao odgovor na radnju poslanu u spremište. Temelji se na funkciji smanjivanja u JavaScriptu, gdje se jedna vrijednost izračunava iz više vrijednosti nakon što je izvedena funkcija povratnog poziva. Evo primjera na slici 28 kako reduktori rade u Reduxu.

```
const authReducer = (state {…} = initialState, action) => {
  const { type, payload } = action;

  switch (type) {
    case USER_LOADED:
      return {
        ...state,
        isAuthenticated: true,
        loading: false,
        user: payload
      };
    case REGISTER_SUCCESS:
    case LOGIN_SUCCESS:
      return {
        ...state,
        ...payload,
        isAuthenticated: true,
        loading: false
      };
    case ACCOUNT_DELETED:
    case AUTH_ERROR:
    case LOGOUT:
      return {
        ...state,
        token: null,
        isAuthenticated: false,
        loading: false,
        user: null
      };
    default:
      return state;
```

Slika 28. Prikaz reduktora kod autorizacije.

Kao čiste funkcije, ne mijenjaju podatke u objektu koji im je proslijeđen, niti izvršavaju bilo kakve nuspojave u aplikaciji. S obzirom na isti objekt, uvijek bi trebali proizvesti isti rezultat. Trgovina ima stanje aplikacije. Preporučuje se zadržavanje samo jedne trgovine u bilo kojoj Redux aplikaciji, primjer trgovine možemo vidjeti na slici 29. Pomoću pomoćnih metoda možemo pristupiti pohranjenom stanju, ažurirati stanje te registrirati ili odjaviti slušatelje. S Reduxom postoji jedno opće stanje u

trgovini, a svaka komponenta ima pristup tom stanju. Time se eliminira potreba kontinuiranom prelaska stanja s jedne komponente na drugu. Također možemo odabrati jedan specifičan dio trgovine, a ne cijelu, to čini aplikaciju optimiziranijom.

```
import { createStore, applyMiddleware } from 'redux';
import { composeWithDevTools } from 'redux-devtools-extension';
import thunk from 'redux-thunk';
import rootReducer from './reducers';
import setAuthToken from './utils/setAuthToken';

const initialState = {};

const middleware = [thunk];

const store = createStore(
  rootReducer,
  initialState,
  composeWithDevTools(applyMiddleware(...middleware))
);

let currentState = store.getState();

store.subscribe(() => {

  let previousState = currentState;
  currentState = store.getState();

  if (previousState.auth.token !== currentState.auth.token) {
    const token = currentState.auth.token;
    setAuthToken(token);
  }
});

export default store;
```

Slika 29. Prikaz trgovine Redux-a.

Kad koristimo Redux s React-om, stanja više nećemo morati dizati. Tako ćemo lakše pratiti koja radnja uzrokuje bilo kakvu promjenu. U Reduxu je stanje uvijek predvidljivo. Ako se isto stanje i radnja prenose na reduktor, uvijek se dobije isti rezultat jer su reduktori čiste funkcije. Stanje je također nepromjenjivo i nikada se ne mijenja. To omogućuje provedbu mukotrpnih zadataka poput beskonačnog poništavanja i

ponavljanja. Također je moguće implementirati putovanje kroz vrijeme, točnije sposobnost kretanja naprijed-unatrag među prethodnim stanjima i pregledavane rezultata u stvarnom vremenu.

Redux je strog u pogledu načina na koji bi kod trebao biti organiziran, što nekome tko poznaje Redux olakšava razumijevanje strukture bilo koje Redux aplikacije. To općenito olakšava održavanje. Ovo također pomaže da odvojimo svu poslovnu logiku od stabla komponenti. Za velike aplikacije važno je da aplikacija bude predvidljiva i održiva. Redux olakšava ispravljanje pogrešaka u aplikaciji. Zapisivanjem akcija i stanja lako je razumjeti greške u kodiranju, pogreške u mreži i druge oblike grešaka koje se mogu pojaviti tijekom proizvodnje. Osim evidentiranja, ima odlične razvojne alate koji omogućuju radnje tijekom putovanja kroz vrijeme, trajne radnje na osvježavanju stranice itd. Za srednje i velike aplikacije otklanjanje pogrešaka oduzima više vremena. Možemo pretpostaviti da bi održavanje globalnog stanja aplikacije dovelo do neke degradacije performansi. U velikoj mjeri to nije slučaj. Redux interno implementira mnoge optimizacije performansi, tako da se povezana komponenta ponovno iscrtava samo kada to zaista treba. Redux aplikacije je lako testirati jer se funkcije koriste za promjenu stanja čistih funkcija. Možemo zadržati dio stanja aplikacije u lokalnoj pohrani i vratiti ga nakon osvježavanja. Redux se također može koristiti za iscrtavanje na strani poslužitelja. Pomoću lokalne pohrane možemo upravljati početnim iscrtavanjem aplikacije slanjem stanja poslužitelju zajedno s njezinim odgovorom na zahtjev poslužitelja. Potrebne komponente zatim se generiraju u HTML-u i šalju klijentima.

Neki od problema korištenja Reduxa su to da on dolazi bez enkapsulacije, bilo koja komponenta može pristupiti podacima što može uzrokovati sigurnosne probleme. Osim toga, on sadrži puno „boilerplate“ koda, točnije kod koji se ponavlja u više sekcija. Podosta je ograničen oko dizajna, kako je stanje nepromjenjivo u reduxu, reduktor ažurira stanje vraćajući svaki puta novo stanje što može uzrokovati pretjeranu upotrebu memorije. Nakon što se aplikacija dovoljno poveća, praktično je premjestiti sve konstante u posebni modul koji se koristi za projekt kao što je vidljivo na slici 30.

```
export const SET_ALERT = 'SET_ALERT';
export const REMOVE_ALERT = 'REMOVE_ALERT';
export const REGISTER_SUCCESS = 'REGISTER_SUCCESS';
export const REGISTER_FAIL = 'REGISTER_FAIL';
export const USER_LOADED = 'USER_LOADED';
export const AUTH_ERROR = 'AUTH_ERROR';
export const LOGIN_SUCCESS = 'LOGIN_SUCCESS';
export const LOGIN_FAIL = 'LOGIN_FAIL';
export const LOGOUT = 'LOGOUT';
export const GET_PROFILE = 'GET_PROFILE';
export const GET_PROFILES = 'GET_PROFILES';
export const GET_REPOS = 'GET_REPOS';
export const NO_REPOS = 'NO_REPOS';
export const UPDATE_PROFILE = 'UPDATE_PROFILE';
export const CLEAR_PROFILE = 'CLEAR_PROFILE';
export const PROFILE_ERROR = 'PROFILE_ERROR';
export const ACCOUNT_DELETED = 'ACCOUNT_DELETED';
export const GET_POSTS = 'GET_POSTS';
export const GET_POST = 'GET_POST';
export const POST_ERROR = 'POST_ERROR';
export const UPDATE_LIKES = 'UPDATE_LIKES';
export const DELETE_POST = 'DELETE_POST';
export const ADD_POST = 'ADD_POST';
export const ADD_COMMENT = 'ADD_COMMENT';
export const REMOVE_COMMENT = 'REMOVE_COMMENT';
```

Slika 30. Tipovi akcija.

7.3. JSX

Slika 29 pokazuje primjer JSX-a, a on je umetnuta oznaka koja izgleda kao HTML i pretvara se u JavaScript [7]. JSX izraz počinje otvorenom oznakom nalik HTML-u, a završava odgovarajućom završnom oznakom. JSX oznake podržavaju XML sintaksu za samo zatvaranje, pa prema potrebi možemo ostaviti zaključnu oznaku isključenom. JSX izrazi se procjenjuju na elemente React-a. Stilove možemo postaviti dodjeljivanjem običnog JavaScript objekta atributu „style“. Ne koristimo CSS sintaksu. Umjesto toga, koristi se sličan format doslovnog JavaScript objekta. Postoji skup rukovatelja događajima koje možemo spojiti na način koji bi trebao izgledati vrlo poznato svima koji poznaju HTML.

```
import React, { Fragment, useEffect } from 'react';
import PropTypes from 'prop-types';
import { Link } from 'react-router-dom';
import { connect } from 'react-redux';
import Spinner from '../layout/Spinner';
import ProfileTop from './ProfileTop';
import ProfileAbout from './ProfileAbout';
import ProfileExperience from './ProfileExperience';
import ProfileEducation from './ProfileEducation';
import { getProfileById } from '../../actions/profile';

const Profile = ({ getProfileById, profile: { profile }, auth, match }) => {
  useEffect( effect: () => {
    getProfileById(match.params.id);
  }, deps: [getProfileById, match.params.id]);

  return (
    <Fragment>
      {profile === null ? (
        <Spinner />
      ) : (
        <Fragment>
          <Link to="/profiles" className="btn btn-light">
            Back To Profiles
          </Link>
          {auth.isAuthenticated &&
            auth.loading === false &&
            auth.user._id === profile.user._id && (
              <Link to="/edit-profile" className="btn btn-dark">
                Edit Profile
              </Link>
```

Slika 31. Primjer JSX-a.

Iako izgleda tako, JSX nije predložak u smislu da su „handlebars“ (ručke) i EJS (ugrađeni Javascript predlošci) predlošci. To nije jednostavna zamjena znaka i ispisivanje „innerHTML-a“. To je zapravo deklarativna sintaksa koja se koristi za izražavanje virtualnog DOM-a. JSX se tumači i pretvara u virtualni DOM, koji se razlikuje od pravog DOM-a. Umjesto prepisivanja cijelog stabla DOM-a, primjenjuju se samo razlike. Zbog toga se React brzo iscrtava. Osim toga, JSX ugrađuje zajedničku zaštitu od XSS napad (skriptiranje na više stranica).

JSX nije ograničen na HTML. Možemo ga koristiti za stvaranje proizvoljnih stabala objekata. Netflix koristi tu mogućnost za zrcaljenje svoje arhitekture web aplikacija na velikom broju uređaja koristeći vlastiti prilagođeni objektni model za TV iscrtavanje. React Native koristi ga za iscrtavanje elemenata korisničkog sučelja izvornog za uređaj. U tom smislu, JSX je zapravo puno fleksibilniji i svestraniji nego što može biti predložak upravljača. Unatoč tome što koristi poznatu sintaksu atributa, JSX ne obrađuje događaje poput iste sintakse u HTML-u.

JSX događaji automatski se delegiraju na korijenski čvor React-a. Zapravo, ide se korak dalje. Postavlja se jedan rukovatelj događaja na korijenskom čvoru koji obrađuje sve događaje. Što to znači? Obično kada slušatelje događaja priključimo izravno na element s kojim korisnik stupa u interakciju, na kraju imamo mnogo slušatelja događaja u memoriji za jednu stranicu. React automatski stvara jednog slušatelja, tako da više nikada ne moramo razmišljati o delegiranju događaja. Ovo je stvarno sjajno za stvari poput beskonačnog pomicanja, jer ne moramo brinuti o beskonačno rastućoj potrošnji memorije (curenje memorije).

Ono što možemo s JSX napraviti je prosljeđivati svojstva, to jest podatke koji se prenose u komponentu kao atributi elementa. Negdje drugdje u aplikaciji React prisluškuje promjene stanja i ponovno se poziva metoda „render()“ koja prosljeđuje promijenjene podatke u svojstva. Može se također primijetiti da komponenta nema znanja o tome što slušatelji događaja rade.

7.4. React router

Usmjeravanje (routing) je proces u kojem je korisnik usmjeren na različite stranice na temelju njihove radnje ili zahtjeva [8]. React usmjerivač uglavnom se koristi za razvoj web aplikacija za jednu stranicu. On se koristi za definiranje više ruta u aplikaciji. Kada korisnik upiše određeni URL u pregledniku i ako se taj URL podudara s bilo kojom rutom unutar datoteke usmjerivača, korisnik će biti preusmjeren na tu određenu rutu.

React usmjerivač je standardni sustav React biblioteke koji se koristi pomoću React Router paketa. Omogućuje sinkronizirani URL preglednik s podacima koji će biti prikazan na web stranici. Održava standardnu strukturu i ponašanje aplikacije. Igra veliku ulogu za prikaz više prikaza u aplikaciji na jednoj stranici. Bez React usmjerivača nije moguće prikazati više prikaza u React aplikacijama, primjer izgleda usmjerivača ćemo vidjeti na slici 32. Većina web stranica društvenih medija poput Facebook-a, Instagram-a koristi React Router za iscrtavanje više sadržaja.

React sadrži tri različita paketa za usmjeravanje:

- React Router: Pruža osnovne komponente usmjeravanja i funkcije za aplikacije React Router-a.
- React-router-native: Koristi se za mobilne aplikacije.
- React-router-dom: Koristi se za dizajn web aplikacija.

Nije moguće instalirati React-usmjerivač izravno u aplikaciju. Da bismo koristili usmjeravanje, prvo moramo instalirati module za usmjeravanja DOM-a u aplikaciji. To možemo napraviti s komandom „npm-install react-router-dom –save“.

```
const Routes = () => {
  const chatPath = "/livechat"
  const match = useLocation()
  console.log(match.pathname)
  return (
    <section className={chatPath===match.pathname ? 'chatContainer' : "container"}>
      <Alert />
      <Switch>
        <Route exact path={chatPath} component={LiveChat} />
        <Route exact path="/register" component={Register} />
        <Route exact path="/login" component={Login} />
        <Route exact path="/profiles" component={Profiles} />
        <Route exact path="/profile/:id" component={Profile} />
        <PrivateRoute exact path="/dashboard" component={Dashboard} />
        <PrivateRoute exact path="/create-profile" component={ProfileForm} />
        <PrivateRoute exact path="/edit-profile" component={ProfileForm} />
        <PrivateRoute exact path="/add-experience" component={AddExperience} />
        <PrivateRoute exact path="/add-education" component={AddEducation} />
        <PrivateRoute exact path="/posts" component={Posts} />
        <PrivateRoute exact path="/posts/:id" component={Post} />
        <Route component={NotFound} />
      </Switch>
    </section>
  );
};
```

Slika 32. Popis usmjerivača aplikacije.

Ponekad želimo imati više veza na jednoj stranici. Kada kliknemo na bilo koju od tih veza, ona bi trebala učitati stranicu koja je povezana s tom rutom bez ponovnog učitavanja web stranice. Da bismo to učinili, moramo uvesti komponentu <Link>. Primjer izvođenja komponente link možemo vidjeti na slici 33. <Link> je glavni način da omogućimo korisnicima navigaciju kroz aplikaciju. On može znati gdje se ruta povezuje te gdje je ona aktivna i automatski može uhvatiti bilo koju klasu koja je aktivna, te njena svojstva. Bit će aktivan cijelo vrijeme ako je trenutna ruta povezana ili ako je potomak povezane rute.

```
import React from 'react';
import { Link } from 'react-router-dom';
import PropTypes from 'prop-types';

const ProfileItem = ({
  profile: {
    user: { _id, name, avatar },
    status,
    company,
    location,
    skills
  }
}) => {
  return (
    <div className='profile bg-light'>
      <img src={avatar} alt='' className='round-img' />
      <div>
        <h2>{name}</h2>
        <p>
          {status} {company && <span> at {company}</span>}
        </p>
        <p className='my-1'>{location && <span>{location}</span>}</p>
        <Link to={`/profile/${_id}`} className='btn btn-primary'>
          View Profile
        </Link>
      </div>
      <ul>
        {skills.slice(0, 4).map((skill, index) => (
          <li key={index} className='text-primary'>
            <i className='fas fa-check' /> {skill}
          </li>
```

Slika 33. Primjer korištenja Link-a.

7.5. React Hooks

React kuke (hooks) su nova značajka predstavljena u React 16.8 verziji. Omogućuje korištenje stanja drugih React značajki bez pisanja klase [9]. Kuke su funkcije koje se spajaju u stanje reagiranja i značajke životnog ciklusa iz komponenti. Ne rade unutar klasa. Kompatibilne su tako da ne sadrže nikakve promjene. Ako napišemo komponentu funkcije, a zatim joj želimo dodati neko stanje, prethodno to činimo pretvarajući ju u klasu, no pomoću kuke, to možemo učiniti unutar postojeće komponente. Kuke su slične funkcijama JavaScripta, ali moramo se pridržavati određenih pravila kada ih koristimo. Pravilo kuka je da sva logika stanja u komponenti bude vidljiva u izvornom kodu,a osim toga bitno je još izdvojiti:

- Kuke se pozivaju samo na najvišoj razini. Ne trebaju se pozivati unutar petlji, uvjeta ili ugniježđenih funkcija, samo na najvišoj razini React funkcija. Ovo pravilo osigurava da se kuke pozivaju istim redoslijedom svaki puta kad se komponenta prikaže.
- Ne možemo pozvati kuku iz uobičajenih JavaScript funkcija. Umjesto toga, možemo pozvati kuku iz komponentnih funkcija React-a. Ono što je također interesantno kod korištenja React kuka je da iskusan programer može napraviti svoju vlastitu prilagođenu kuku, koju može koristiti kroz svoju aplikaciju, dakle postoje već predodređene React kuke, a moguće je i napraviti vlastitu kuku.

Preduvjeti za korištenje React kuke je Node verzija 6 ili novija, NPM verzija 5.2 ili novija i alat za stvaranje React aplikacija. Stanje kuke je novi način deklariranja stanja u React aplikaciji. Kuka koristi funkcionalnu komponentu „useState()“, vidljivo sa slike 34 za postavljanje i dohvaćanje stanja. Kuka za efekte omogućuje nam izvođenje akcija u funkcijskim komponentama. Drugim riječima, kuke efekata ekvivalentne su metodama životnog ciklusa „componentDidMount(), componentDidUpdate(), componentWillUnmount()“. Akcije imaju zajedničke značajke koje većina web aplikacija mora izvesti, kao što su ažuriranje DOM-a, dohvaćanje i trošenje podataka s poslužiteljskog API-ja i postavljanje pretplate.

```
const Register = ({ setAlert, register, isAuthenticated }) => {
  const [formData, setFormData] = useState( initialState: {
    name: '',
    email: '',
    password: '',
    password2: ''
  });

  const { name, email, password, password2 } = formData;

  const onChange = (e) =>
    setFormData( value: { ...formData, [e.target.name]: e.target.value });

  const onSubmit = async (e) => {
    e.preventDefault();
    if (password !== password2) {
      setAlert('Passwords do not match', 'danger');
    } else {
      register({ name, email, password });
    }
  };
```

Slika 34. Primjer useState kuke pri registraciji.

U React komponenti postoje dvije vrste nuspojava, a to su efekti bez čišćenja i efekti s čišćenjem. Efekti bez čišćenja se koriste s „useEffect()“ kukom čiji je primjer vidljiv na slici 35 koja ne blokira preglednik u ažuriranju zaslona. Čini aplikaciju osjetljivijom. Najčešći primjeri efekta koji ne zahtijevaju čišćenje su ručne DOM mutacije, mrežni zahtjevi, evidentiranje itd. Efekti s čišćenjem zahtijevaju čišćenje nakon ažuriranja DOM-a. Ako želimo postaviti pretplatu na neki vanjski izvor podataka, važno je očistiti memoriju kako ne bismo uveli curenje memorije. React vrši čišćenje memorije kada se komponenta demontira. Međutim, kako to znamo, efekti se pokreću za svaku metodu prikazivanja sadržaja, a ne samo jednom. Stoga React također čisti efekte iz prethodnog iscrtavanja prije sljedećeg izvođenja efekata.

```
const Profiles = ({ getProfiles, profile: { profiles, loading } }) => {
  useEffect( effect: () => {
    getProfiles();
  }, deps: [getProfiles]);

  return (
    <Fragment>
      {loading ? (
        <Spinner />
      ) : (
        <Fragment>
          <h1 className='large text-primary'>Gamers</h1>
          <p className='lead'>
            <i className='fab fa-connectives' /> Browse and connect with
            gamers
          </p>
          <div className='profiles'>
            {profiles.length > 0 ? (
              profiles.map(profile => (
                <ProfileItem key={profile._id} profile={profile}
                />


              ))
            ) : (
              <h4>No profiles found...</h4>
            )}
          </div>
        </Fragment>
      )}
```

Slika 35. Primjer useEffect kuke za profil korisnika.

8. Zaključak

Razvoj web aplikacija nije ono što je bio nekad, čak ni prije par godina. Danas postoji toliko mogućnosti, a neupućeni su često zbunjeni oko toga što koristiti. Ne samo veliki skup tehnologije koja se koristi, već čak i za alate koji pomažu u razvoju. Svaka web aplikacija je izgrađena pomoću više tehnologija. Kombinacije ovih tehnologija nazivaju se „stack“ (stog).

Kako je web razvoj sazrijevao, njihova interaktivnost je dolazila do izražaja, aplikacije namijenjene za jednu stranicu su postale sve popularnije. Temeljni građevni blok React-a je komponenta koja održava vlastito stanje i prikazuje sama sebe. Sve što radimo u React-u je gradnja komponenti. Zatim sastavimo komponente u jedan skup koji opet postaje zasebna komponenta i on prikazuje potpuni prikaz podataka na stranici. Komponenta enkapsulira stanje podataka i pogled ili način na koji su oni ispisani. To olakšava pisanje i zaključivanje cijele aplikacije jer se tom metodom usredotočujemo na jednu po jednu stvar. Komponente međusobno razgovaraju dijeljenjem informacija o stanju u obliku svojstava.

NPM je zadani upravitelj paketa za Node.js. Pomoću npm-a možemo instalirati razne pakete ili biblioteke, a i upravljati ovisnostima među njima. Iako je npm započeo kao spremište za Node.js module, brzo se pretvorio u upravitelja paketa za isporuku drugih modula temeljenih na JavaScriptu, osobito onih koji se mogu koristiti u pregledniku. Iako je React uglavnom klijentski kod i može biti izravno uključen u HTML kao skriptna datoteka, preporučljivo je da se React instalira putem npm-a. Nakon što se instalira kao paket, potrebni su nam alati za izgradnju poput webpack-a, koji mogu sastaviti module, kao i biblioteke. Node.js ima asinkroni, ulazno/izlazni model koji ne blokira događaje, za razliku od upotrebe dretve za postizanje više zadataka. Većina drugih jezika ovisi o dretvama za istodobno obavljanje poslova.

Ovaj stog obuhvaća cijeli ciklus web razvoja od front end razvoja (strana klijenta) do back end razvoja (strana poslužitelja) koristeći JavaScript. Podržava arhitekturu MVC (Model View Controller) kako bi razvojni proces tekao glatko. Uz javaScript stog, programeri samo trebaju poznavati JavaScript i JSON. Izvodljivost četiri najbolje tehnologije, tj. MongoDB, ExpressJS, React i Node.Js dolazi s unaprijed izgrađenim

opsežnim paketom alata za testiranje, te otvorenim izvorom u okvirima i dobrom podrškom njegove zajednice.

U svrhu diplomskog rada je napravljena web aplikacija koristeći MERN tehnologije. Aplikacija ima 2 sloja, klijentski i poslužiteljski sloj. React.js je korišten za izradu klijentskog sloja, a Node.js i Express.js služe za poslužiteljski sloj. Za bazu podataka korišten je MongoDB. Korisnici aplikacije mogu nadograđivati svoj portfolio, upoznavati druge igrače, sprijateljiti se s njima i pričati preko chata koji je postavljen uživo.

Postoje aplikacije koje su bolje i praktičnije od ove aplikacije i imaju aktivne korisnike, kao npr. Steam. Pojedine funkcionalnosti aplikacije su ograničene i postoji puno prostora za nadogradnju aplikacije, primjerice optimizacija chata, bolje filtriranje profila korisnika. Osim toga, moguće su nadogradnje gdje korisnici mogu ocijeniti drugog korisnika ili promovirati njegovo iskustvo tako što glasaju da je uistinu dobar u tome u čemu je naveo da ima iskustva.

U svrhu diplomskog rada također je dokazana snaga React.js i jednostavno korištenje njegovih svojstava. Bez korištenja React.js ili nekog drugog razvojnog okruženja, proces izrade bi bio kompliciraniji, stoga je najbolje rješenje koristiti nekakav okvir ili biblioteku za klijentski sloj, a Node.js za poslužiteljski sloj.

9. Literatura

[1] MongoDB službena dokumentacija, Why Use MongoDB and When to Use It? <raspoloživo na: https://www.mongodb.com/why-use-mongodb>, [pristupljeno: 15.08.2021]

[2] Tutorialspoint (resursi za internetske edukacije), Node.js – Introduction, <raspoloživo na: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm>, [pristupljeno: 15.08.2021]

[3] Mozilla MDN Web Docs (resursi za programere), Express/Node introduction, <raspoloživo na: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction>, [pristupljeno: 20.08.2021]

[4] Mulesoft (edukacija o API-u i općenito spajanje podataka sa sistemom) What is an API? (Application Programming Interface), <raspoloživo na: https://www.mulesoft.com/resources/api/what-is-an-api>, [pristupljeno: 21.08.2021]

[5] Taha Sufiyan (svojstva React-a) What is ReactJS: Introduction To React and Its Features, <raspoloživo na: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs#features_of_react>, [pristupljeno: 25.08.2021]

[6] Neo Ighodaro (Redux) Why use Redux? A tutorial with examples, <raspoloživo na: https://blog.logrocket.com/why-use-redux-reasons-with-clear-examples-d21bffd5835/>, [pristupljeno: 29.08.2021]

[7] Eric Elliott (JSX) JSX Looks Like An Abomination But it's Good for You, <raspoloživo na: https://medium.com/javascript-scene/jsx-looks-like-an-abomination-1c1ec351a918>, [pristupljeno: 01.09.2021]

[8] JavaTpoint (Routing) React Router and the Need of React Router, <raspoloživo na: https://www.javatpoint.com/react-router>, [pristupljeno: 01.09.2021]

[9] JavaTpoint (Hooks) React Hooks and When to use Hooks, <raspoloživo na: https://www.javatpoint.com/react-hooks>, [pristupljeno: 04.09.2021]

[10] Background image (Landing page) Dota 2 legacy wallpaper, <raspoloživo na: https://wallpaper.dog/large/17089950.jpg>, [pristupljeno: 15.08.2021]

[11] Razlika Virtual DOM-a i pravog DOM-a, <raspoloživo na: https://s3.ap-southeast-1.amazonaws.com/arrowhitech.com/wp-content/uploads/2020/07/02034639/compare.jpg>, [pristupljeno: 25.08.2021]

10. Popis slika