Izrada web aplikacije za praćenje statistike igrača u videoigri League of Legends pomoću Angular-a i Node.js-a

Marinac, Tonči

Sveučilište u Rijeci, Fakultet informatike i digitalnih tehnologija

Sveučilišni prijediplomski studij Informatika

Tonči Marinac

Izrada web aplikacije za praćenje statistike igrača u videoigri League of Legends pomoću Angular-a i Node.js-a

Završni rad

Mentor: doc. dr. sc. Lucia Načinović Prskalo

Rijeka, 9.8.2023.

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija
www.inf.uniri.hr

Rijeka, 4.4.2023.

Zadatak za završni rad

Pristupnik: Tonči Marinac

Naziv završnog rada: Izrada web aplikacije za praćenje statistike igrača u videoigri *League of Legends* pomoću Angular-a i Node.js-a

Naziv završnog rada na engleskom jeziku: Creating a web application for tracking player statistics in the video game *League of Legends* using Angular and Node.js

Sadržaj zadatka: Zadatak završnog rada je izraditi web aplikaciju koja omogućuje korisnicima pregled osnovnih informacija o profilima igrača u igri „League of Legends“ korištenjem Angular, Node.js i Firebase tehnologija. U radu će biti opisane korištene tehnologije, arhitektura aplikacije te implementacija i testiranje aplikacije.

Mentor

Doc. dr. sc. Lucia Načinović Prskalo

Voditelj za završne radove

Doc. dr. sc. Miran Pobar

Zadatak preuzet: 4.4.2023.

(potpis pristupnika)

Adresa: Radmile Matejčić 2
51000 Rijeka, Hrvatska
Tel: **+385(0)51 584 700**
E-mail: **ured@inf.uniri.hr**
OIB: **64218323816**
IBAN: **HR1524020061400006966**

Sadržaj

Sažetak

Ovaj završni rad istražuje razvoj web aplikacije „Summoner Spy" koja pruža korisnicima mogućnost praćenja igrača i statistika u popularnoj videoigri „League of Legends“. Na početku zavrpnog rada se predstavlja tema rada i opisuje svrha istraživanja. Videoigra „League of Legends“ je ukratko predstavljena kako bi se čitateljima pružio kontekst. U poglavlju "Pregled tehnologija i alata" analizirane su tehnologije i alati koji su korišteni u razvoju aplikacije. Raspravljene su prednosti i nedostaci odabranih tehnologija, a također su navedeni primjeri drugih aplikacija koje su razvijene koristeći iste tehnologije. "Analiza zahtjeva aplikacije" donosi opis funkcionalnosti aplikacije i postavlja zahtjeve korisnika u središte pozornosti. U poglavlju "Arhitektura aplikacije" detaljno je opisana arhitektura aplikacije, struktura baze podataka i uloga svake komponente u aplikaciji. "Implementacija aplikacije" pruža dublji uvid u razvoj aplikacije koristeći Angular i Node.js. Glavne funkcionalnosti aplikacije detaljno su opisane kako bi se čitateljima omogućilo razumijevanje procesa izrade. Poglavlje "Testiranje aplikacije" istražuje načine testiranja aplikacije i analizira rezultate. Testiranje je uključivalo aktivno uključivanje prijatelja i kolega u testiranje aplikacije kako bi se identificirali i ispravili pronađeni bugovi. Ovaj završni rad pruža sveobuhvatan pregled razvoja web aplikacije "Summoner Spy" i nudi smjernice za buduće istraživanje i unaprjeđenje aplikacije.

Ključne riječi: web aplikacija, statistika, League of Legends, web razvoj, Angular, Node.js, Express.js, Tailwind, DaisyUI, frontend, backend, korisničko sučelje, autentifikacija, Firebase, NoSQL

1. Uvod

Tema završnog rada obuhvaća razvoj aplikacije pod nazivom „Summoner Spy“ koristeći Angular, Node.js i Firebase tehnologije. Ova aplikacija omogućuje korisnicima pregled osnovnih informacija o profilima igrača u popularnoj igri „League of Legends“. "League of Legends" je iznimno popularna MOBA (Multiplayer Online Battle Arena) videoigra koja je nastala 2009. godine. Ova igra se ističe svojim dinamičnim i taktičkim pristupom, gdje se timovi igrača nadmeću kako bi postigli dominaciju nad protivničkom ekipom. S obzirom na širok spektar likova, taktika i mehanika igre, "League of Legends" nudi duboko iskustvo koje privlači raznoliku skupinu igrača. Timovi se sastoje od pet igrača te je svakom timu cilj, iskorištavanjem bilo kakve vrste prednosti nad drugim timom, uništiti protivničku bazu, odnosno glavnu strukturu koja se zove Nexus [1]. Cilj ovog rada jest detaljno istražiti proces razvoja web aplikacije "Summoner Spy" te demonstrirati korake implementacije kroz upotrebu tehnologija Angular i Node.js. Glavna svrha aplikacije je omogućiti korisnicima da lako pristupe osnovnim podacima o profilima igrača u igri "League of Legends", uključujući povijest odigranih mečeva, rangiranje u različitim modovima te statistike vezane uz najčešće korištene likove. Kroz ovaj rad, prikazat ćemo osnove Angular-a (direktive, manipuliranje i vezanje podataka i sl.) te ćemo proučiti kako se ostvaruje komunikacija s API-jem igre "League of Legends" u Node.js-u kako bismo dohvatili relevantne podatke te njih poslali Angular-u.

2. Pregled tehnologija i alata

U ovoj sekciji, detaljno ćemo analizirati ključne tehnologije i alate koji su korišteni tijekom razvoja web aplikacije "Summoner Spy". Osvrnuti ćemo se na njihove karakteristike, prednosti i nedostatke te pružiti primjere kako su ovi alati doprinijeli izradi konačnog proizvoda.

2.1. Korištene tehnologije i alati

Korišteni su raznovrsni alati i tehnologije kako bi se ostvarila funkcionalnost i atraktivnost web aplikacije "Summoner Spy". Glavni okvir za razvoj front-end dijela aplikacije bio je Angular, moderni okvir za izradu korisničkih sučelja [2]. Angular je omogućio strukturiranje komponenti sučelja te interaktivnost korisničkog iskustva. U kombinaciji s Angularom, koristili smo Tailwind CSS i DaisyUI za stiliziranje sučelja. Tailwind CSS pruža set korisnih klasa koje zamjenjuju korištenje CSS datoteka [3], dok je DaisyUI dodatak koji donosi svoje komponente i funkcionalnosti poput botuna, tablica, navigacije i sl. za lakšu i bržu izradu front-end dijela web aplikacije [4]. Za razvoj poslužiteljske strane, odnosno back-end dijela web aplikacije, koristili smo Express.js, okvir za Node.js, koji je omogućio brzo postavljanje rute i upravljanje zahtjevima. Komunikacija s bazom podataka, kao i autentifikacija korisnika, ostvareni su kroz Firebase platformu. Firebase je omogućio brz i pouzdan pristup podacima te integraciju autentifikacije u aplikaciju.

2.2. Prednosti i nedostatci korištenih alata

Izabrane tehnologije donose niz prednosti i nedostataka koji igraju ključnu ulogu u razvoju aplikacije. Prednosti ovih tehnologija ističu se na različitim razinama. Angular, kao osnova za izradu korisničkog sučelja, omogućava modularno i skalabilno oblikovanje te pruža bogat skup alata za dinamičko upravljanje komponentama, čime olakšava razvoj interaktivnih dijelova aplikacije.

Angular nameće razvojnim programerima konkretni način pisanja kôda i stroge uzorke kodiranja, potičući ih da usklade svoje prakse s unaprijed definiranim standardima i uzorcima kôda. Ovo dosljedno nametanje pravila i uzoraka kodiranja od strane Angular-a doprinosi povećanju čitljivosti, održivosti i skalabilnosti kôda, olakšavajući timsku suradnju, smanjujući potencijalne greške te ubrzavajući proces razvoja i održavanja aplikacija. Tailwind CSS i DaisyUI pružaju učinkovit pristup oblikovanju sučelja putem definiranih klasa, što ubrzava proces stiliziranja i održavanja dizajna. Ova kombinacija omogućava brzu implementaciju i dosljednost izgleda elemenata sučelja a posebice je značajna u slučaju kada za web stranicu ne postoji gotov dizajn. Express.js donosi prednost brzog postavljanja poslužiteljske strane i efikasnog upravljanja rutama i zahtjevima. Ovaj okvir olakšava komunikaciju između klijentske i poslužiteljske strane, doprinoseći tako glatkom iskustvu korisnika. Firebase, kao platforma za autentifikaciju, upravljanje podacima i hosting, ističe se brzinom i jednostavnošću implementacije ovih ključnih aspekata, a kao NoSQL baza podataka ne zahtjeva ikakvo poznavanje SQL-a. To omogućava da se fokusirate na funkcionalnost aplikacije bez značajnih napora oko infrastrukture. S druge strane, treba uzeti u obzir i određene nedostatke. Naučiti sve koncepte i funkcionalnosti Angulara može zahtijevati neko vrijeme, posebno ako se niste susretali sa sličnim tehnologijama. Iako Tailwind CSS i DaisyUI pojednostavljuju oblikovanje, korištenje gotovih klasa može ograničiti dizajnersku kreativnost i fleksibilnost u postizanju specifičnih izgleda.

2.3. Primjeri aplikacija izrađenih pomoću korištenih alata

Najistaknutiji tehnološki skupovi koji se koriste za kreiranje web aplikacija su MEAN i MERN skupovi alata. U oba slučaja, M, E i N predstavljaju MongoDB koji služi kao baza podataka, Express.js i Node.js koji djeluju kao backend komponente. Glavna razlika između ovih dvaju kombinacija leži u izboru frontend alata. MEAN stog koristi Angular, dok MERN stog koristi React. Unatoč tome što u ovom projektu Firebase zamjenjuje MongoDB iz MEAN stoga, korisno je razmotriti neke od najpoznatijih aplikacija koje su izrađene pomoću MEAN stoga kako bismo bolje razumjeli njihov utjecaj. Budući da je Angular stvorio Google, neizbježno je spomenuti Gmail i YouTube kao iznimno popularne web aplikacije koje su razvijene pomoću Angulara i Node.js-a. Nadalje, tu je LinkedIn, koji je u početku koristio Ruby on Rails za backend, ali se 2011. godine prebacio na Node.js. Ovaj prelazak nije samo smanjio potrošnju serverovih resursa, već je i znatno ubrzao performanse aplikacije. Mjerenjima je dokazano da je aplikacija postala 20 puta brža od prethodne verzije. Nadalje, i primjeri poput Netflix-a, UBER-a, Trello-a, PayPal-a i eBay-a svjedoče o odličnoj skalabilnosti i funkcionalnostima koje se postižu upotrebom ovih alata [5].

3. Analiza zahtjeva aplikacije

U skladu s temom analize zahtjeva aplikacije „Summoner Spy“ i potrebom da se odgovori na specifične zahtjeve korisnika, dalje ćemo detaljno istražiti ključne funkcionalnosti koje ova aplikacija nudi. Proučit ćemo kako ova aplikacija omogućuje korisnicima pregled povijesti mečeva, rangova, heroja i drugih važnih informacija igre „League of Legends“. Nadalje, analizirat ćemo kako se korisnički zahtjevi uklapaju u samu aplikaciju i kako se „Summoner Spy“ razlikuje od sličnih platformi kao što su op.gg i u.gg.

3.1. Opis funkcionalnosti aplikacije

Aplikacija "Summoner Spy" pruža korisnicima pristup raznovrsnim informacijama o igračima u popularnoj igri "League of Legends". Ova web aplikacija omogućava pregled povijesti odigranih mečeva, rangova u različitim modovima, top 3 igračeva heroja prema mastery bodovima, te omogućava pristup leaderboardima za određene regije i rang mode. Osim toga, korisnici mogu pratiti listu besplatnih heroja, dodavati omiljene igrače i pregledavati njihove profile. Aplikacija donosi intuitivan način za pristup i analizu igračevih statistika, poboljšavajući ukupno iskustvo igrača.

3.2. Zahtjevi korisnika

Korisnički zahtjevi odražavaju potrebu za brzim i točnim informacijama o igračima i njihovim performansama u igri. Korisnici žele pregledno i jednostavno sučelje koje će im omogućiti brzo pronalaženje podataka o svojim omiljenim igračima i njihovim mečevima. Također, korisnicima je važno da mogu pratiti promjene u rangovima, otkrivati najbolje igrače u svojoj regiji te pratiti besplatne heroje kako bi bolje planirali svoje igračke strategije.

3.3. Usporedba sličnih aplikacija

Usporedba "Summoner Spy" aplikacije s postojećim platformama kao što su op.gg i u.gg otkriva nekoliko ključnih razlika u funkcionalnostima i korisničkom iskustvu. I op.gg i u.gg su uspostavljene platforme koje pružaju detaljne informacije i statistike o igračima "League of Legends", ali "Summoner Spy" se ističe dodatnim mogućnostima koje su nedostajale. Na primjer, "Summoner Spy" omogućava korisnicima dodavanje igrača u svoju listu favorita i omogućava personalizaciju iskustva. Dok op.gg i u.gg pružaju statistike, "Summoner Spy" korisnicima pruža mogućnost da prate svoje omiljene igrače te se bolje povežu s njihovim performansama. Također, "Summoner Spy" donosi listu besplatnih heroja, što je značajka koja može pomoći novim igračima pri donošenju odluke kojeg heroja kupiti. U konačnici, dok op.gg i u.gg pružaju dublje analize i statistike, "Summoner Spy" se ističe svojom korisničkom personalizacijom, jednostavnim pristupom ključnim informacijama i prilagodbom svim razinama vještine.

4. Arhitektura aplikacije

Arhitektura aplikacije „Summoner Spy“ temelji se na klasičnom MVC (Model-View-Controller) uzorku, koji omogućava jasnu organizaciju i razdvajanje komponenti aplikacije za lakše održavanje i razvoj. Aplikacija slijedi princip SPA (Single Page Application) arhitekture, gdje se korisničko sučelje učitava samo jednom, a navigacija i promjene podataka odvijaju se asinkrono, bez potrebe za ponovnim učitavanjem stranica. Praćenje principa SPA arhitekture je zadano od strane Angular-a. Angular funkcionira tako što sadrži glavnu HTML datoteku „index.html“. Ta datoteka je zapravo jedina HTML datoteka koja se učita, a sve dalje što se dešava je samo izmjenjivanje komponenti koje će se prikazati. Sastoji se od klasičnog „boilerplate“ HTML kôda, a iz nje se poziva glavna „app“ komponenta kao što je vidljivo na slici 1.

```html
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>SS | SummonerSpy</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link rel="stylesheet" href="./dist/main.css">
  <link rel="stylesheet"
  href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css"
  integrity="sha512-iecdLmaskl7CVkqkXNQ/ZH/XLlvWZOJyj7Yy7tcenmpD1ypASozpmT/E0iPtmFIB46ZmdtAc9eNBvH0H/ZpiBw=="
  crossorigin="anonymous" referrerpolicy="no-referrer" />
  <link rel="stylesheet"
  href="https://cdnjs.cloudflare.com/ajax/libs/notyf/3.10.0/notyf.min.css"
  integrity="sha512-ZX18S8AwqoIm9QCd1EYun82IryFikdJt7lxj6583zx5Rvr5HoreO9tWY6f2VhSxvK+48vYFSf4zFtX/t2ge62g=="
  crossorigin="anonymous" referrerpolicy="no-referrer" />
</head>
<body data-theme="light">
  <script src="https://cdnjs.cloudflare.com/ajax/libs/notyf/3.10.0/notyf.min.js"
  integrity="sha512-467grL09I/ffq86LVdwDzi86uaxuAhFZyjC99D6CC1vghMp1YAs+DqCgRvhEtZIKX+o9lR0F2bro6qniyeCMEQ=="
  crossorigin="anonymous" referrerpolicy="no-referrer"></script>
  <app-root></app-root>
</body>
</html>
```

Slika 1. Datoteka index.html

U glavnoj komponenti pozivamo „header“ i „footer“ komponente, jer njih želimo imat na svakoj stranici naše aplikacije, te „router-outlet“ komponentu koja preko „app-routing.module.ts“ datoteke učitava komponente ovisno o linku na kojem se nalazimo (vidi sliku 2.).

```html
<app-header></app-header>

<router-outlet></router-outlet>

<app-footer></app-footer>
```

Slika 2. Glavna app komponenta

Primjerice, u navigaciji imamo link „LEADERBOARDS“ koji nas vodi na „/leaderboards“. Komponenta „router-outlet“ traži ima li ikoja definirana ruta „leaderboards“ u „app.routing.module.ts“ datoteci (vidi sliku 3.) i ako je ima, zamjenjuje sebe sa definiranom komponentom za tu rutu koja je, u ovom slučaju, „LeaderboardsComponent“.

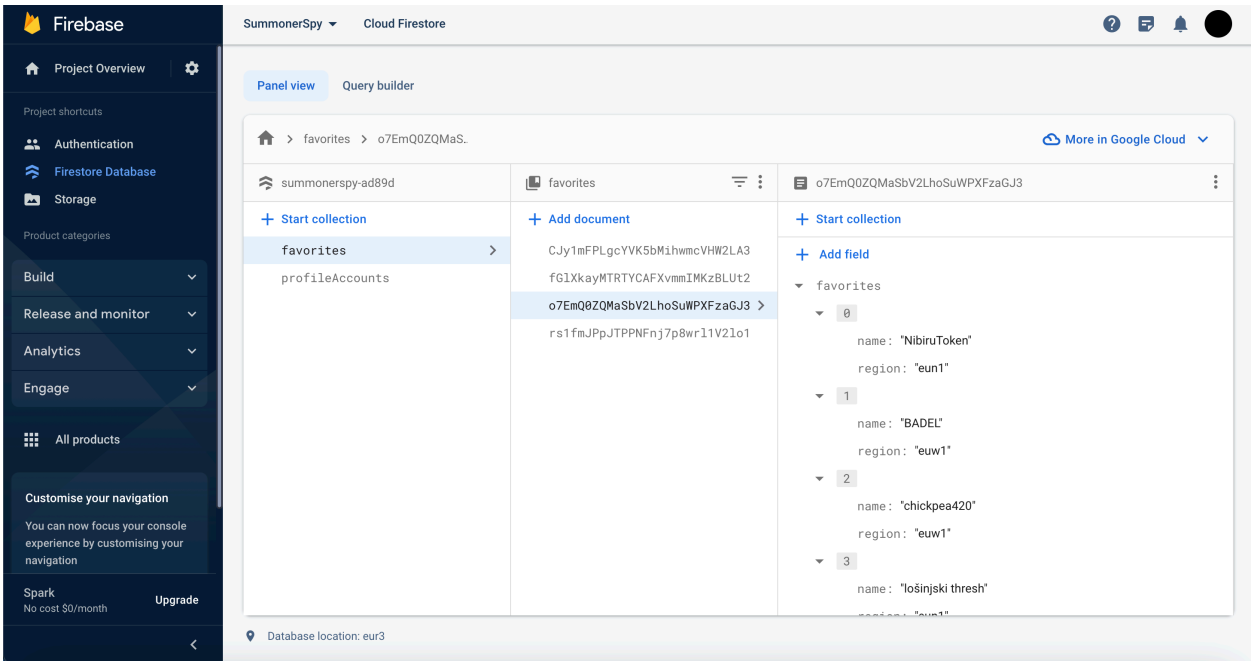
```
import ...

const routes: Routes = [
  { path: '', component: LandingPageComponent },
  { path: 'summoners/:regionCode/:summonerName', component: SummonerComponent },
  { path: 'leaderboards', component: LeaderboardsComponent },
  { path: 'free-rotation', component: FreeRotationComponent },
  { path: 'favorites', component: FavoritesComponent },
  { path: 'profile', component: ProfileComponent },
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  {path: '404', component: NotFoundComponent},
  {path: '**', redirectTo: '/404'}
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Slika 3. Routing vrijednosti za svaku komponentu

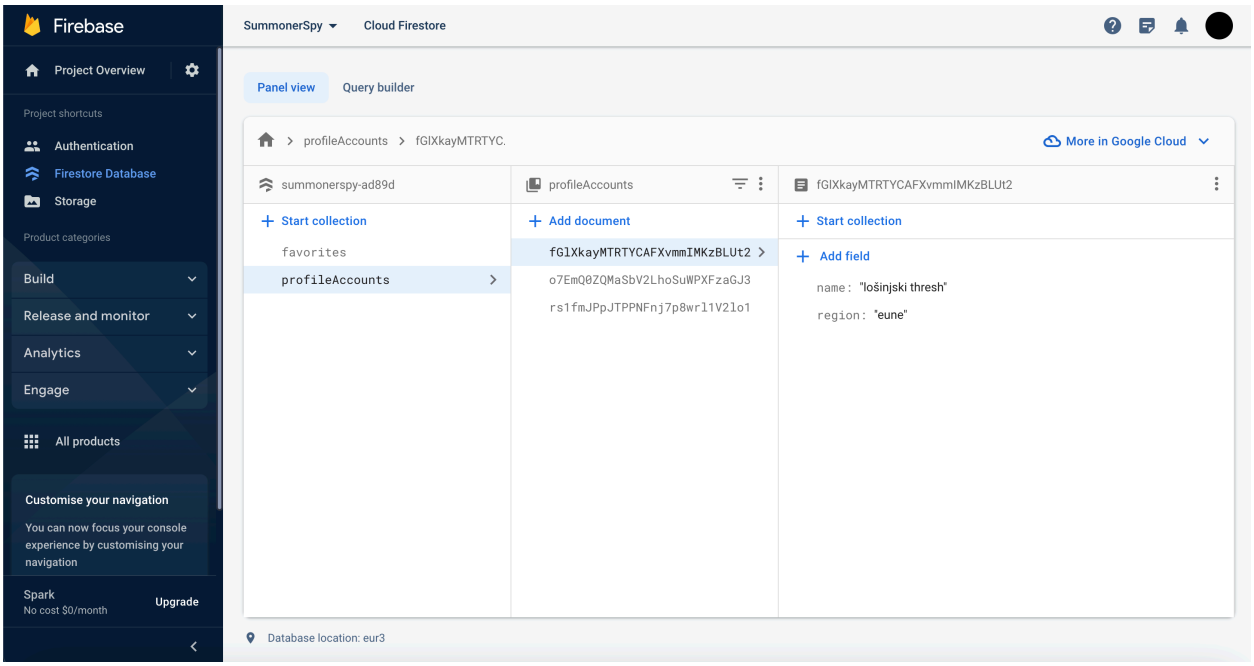
4.1. Struktura baze podataka

Baza podataka u Firebase-u sadrži dvije kolekcije: „favorites“ i „profileAccounts“. Svaka od ovih kolekcija sadrži dokumente čiji su identifikatori ID-ovi prijavljenih korisnika. U „favorites“ kolekciji, svaki dokument pohranjuje niz objekata koji se u Firebase-u nazivaju map elementi. Svaki map element unutar ovog niza, također nazvanog „favorites“, sastoji se od dva polja: "name" i "region". U polju "name" čuva se ime igrača koje je korisnik dodao u omiljene, dok se u polju "region" pohranjuje kôd regije u kojoj se taj igrač nalazi. Vizualnu reprezentaciju ovih podataka u Firebase-u su prikazani na slici 4. Ovi podaci, kombinirani s kôdom i imenom igrača, omogućuju dohvaćanje svih potrebnih informacija o tom igraču pomoću Riotovog API-ja. Takva struktura baze podataka osigurava učinkovitu organizaciju podataka o omiljenim igračima i korisničkim profilima te omogućava brz pristup i manipulaciju ovim informacijama.



Slika 4. Kolekcija favorites

Kolekcija „profileAccounts“ također sadrži u svom dokumentu map element s istim poljima, ali samo je jedan map element. Ne nalazi se u nizu jer je ideja da korisnik može imati samo jednog igrača koji spada pod „profileAccounts“ (vidi sliku 5.).



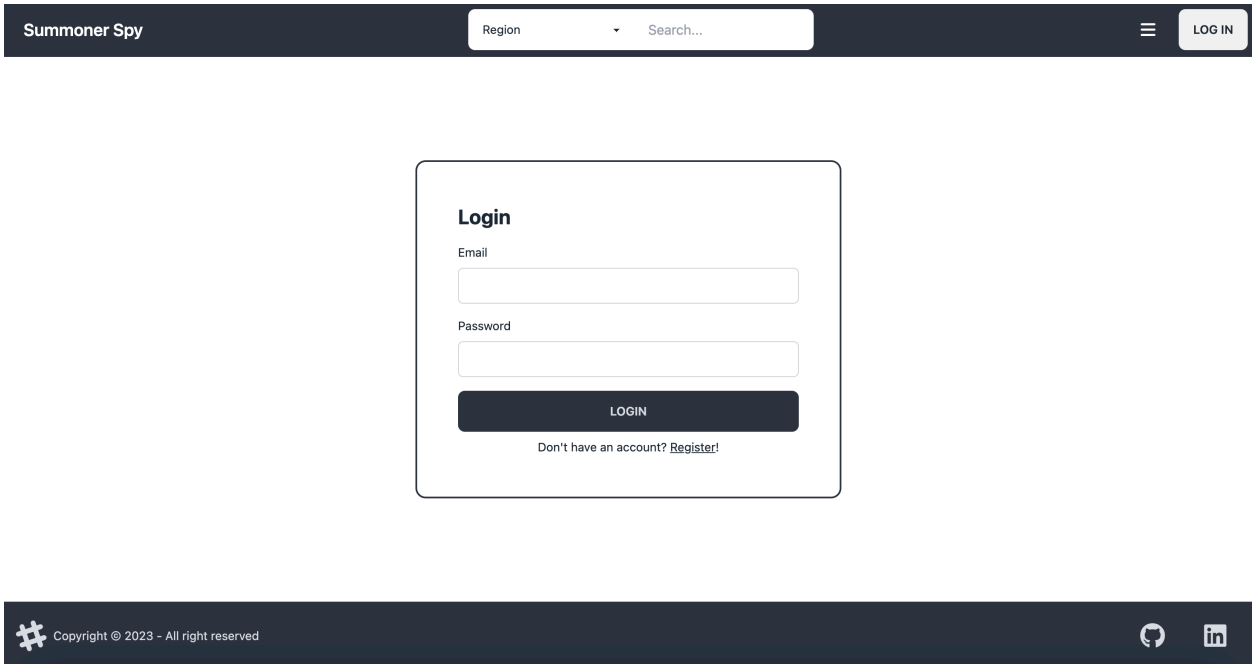
Slika 5. Kolekcija profileAccounts

4.2. Opis komponenti aplikacije i njihove funkcionalnosti

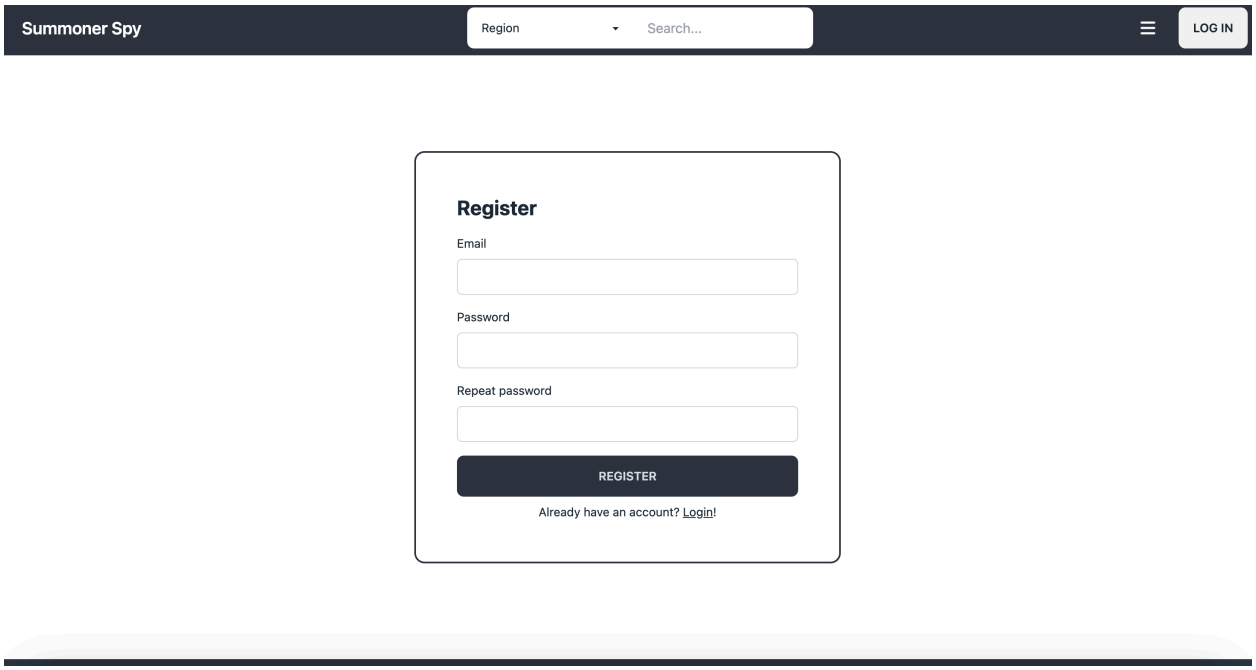
Ova aplikacija je složena mreža različitih dijelova, svaki sa svojom svrhom i ulogom u pružanju bogatog korisničkog iskustva. Od komponenti koje se bave korisničkom autentifikacijom do onih koje omogućavaju praćenje igračkih statistika, svaka komponenta doprinosi cjelokupnom funkcionalnom sklopu "Summoner Spy" aplikacije.

4.2.1. Login i Register komponente

Komponente za prijavu i registraciju omogućuju korisnicima stvaranje računa i prijavu u aplikaciju putem njihovih email adresa i lozinki. Ovdje se koristi tradicionalna metoda autentifikacije, koja osigurava siguran pristup aplikaciji. Korisnici mogu stvoriti vlastite račune i jednostavno se prijaviti radi pristupa svim funkcionalnostima aplikacije. Korisničko sučelje ovih komponenti je prikazano na slikama 6. i 7.



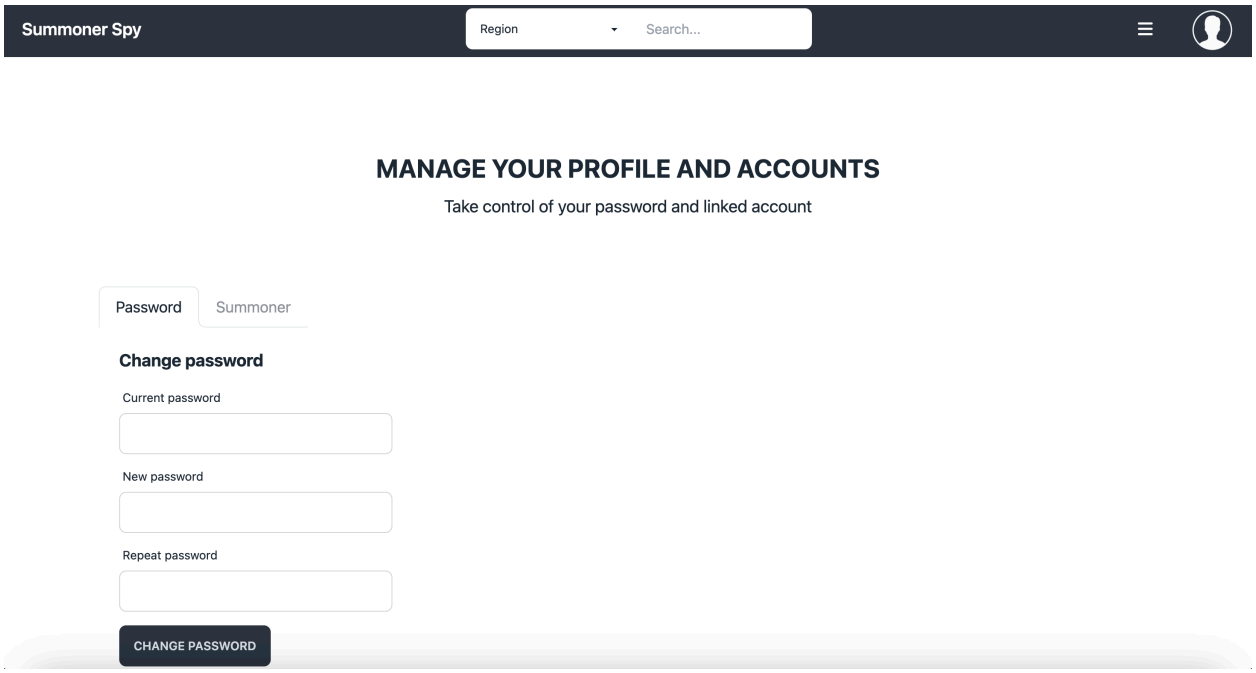
Slika 6. Login komponenta



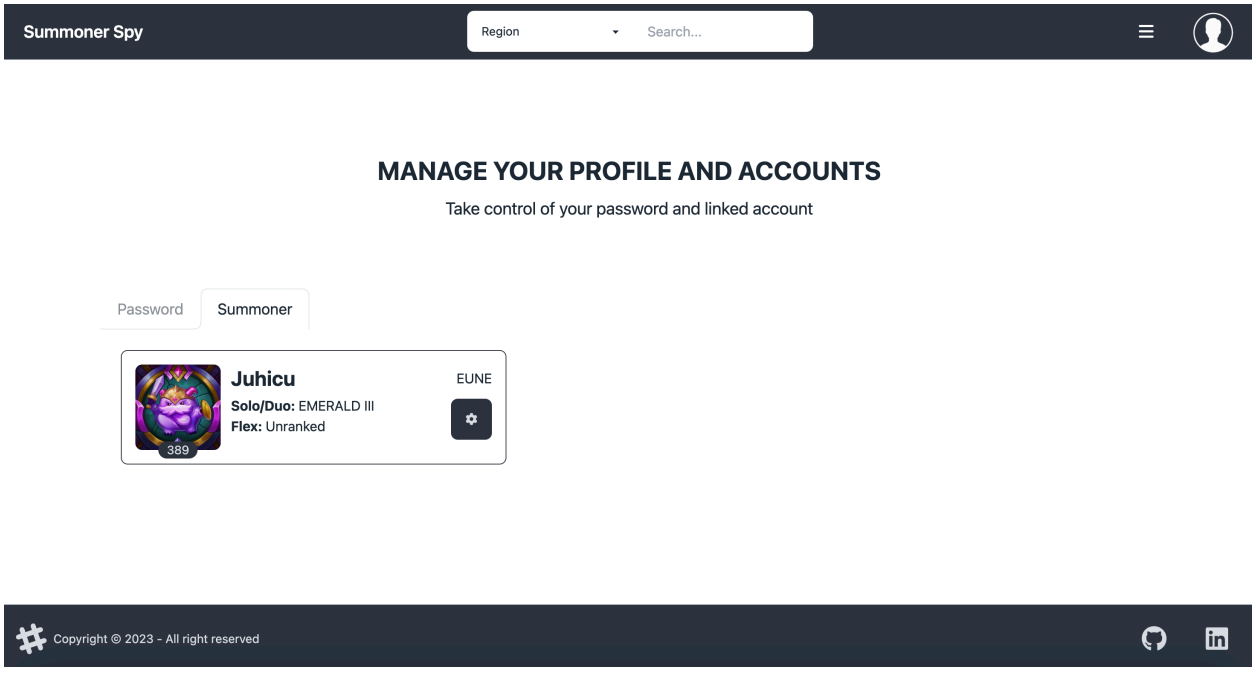
Slika 7. Register komponenta

4.2.2. Profile komponenta

Komponenta profila omogućuje korisnicima da pregledaju svoje korisničke informacije i promijene svoju lozinku (slika 8.). Također, korisnici mogu vidjeti svoj „accountProfile“ koji je označen kao glavni profil. Korisnik glavni profil, pomoću ove komponente, može ukloniti kao „accountProfile“ te ga dodati ili ukloniti iz liste favorita (slika 9.). Ova komponenta omogućuje korisnicima da upravljaju svojim osobnim podacima i postavkama.



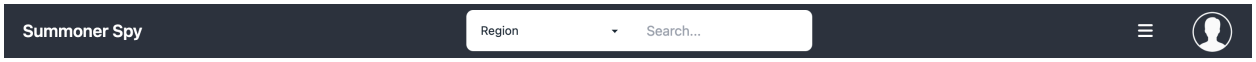
Slika 8. Profile komponenta - mijenjanje lozinke



Slika 9. Profile komponenta - upravljanje profilnim računom

4.2.3. Header i footer komponente

Komponente zaglavlja i podnožja prisutne su na svakoj stranici aplikacije te pružaju dosljedan dizajn i navigaciju. Zaglavlje (slika 10.) omogućuje korisnicima brz pristup važnim dijelovima aplikacije te „search“ komponentu tako da korisnik može u bilo kojem trenutku pretražiti nečiji profil jer to je glavna funkcionalnost ove web aplikacije. Podnožje sadržava dodatne informacije i linkove (vidi sliku 11.).

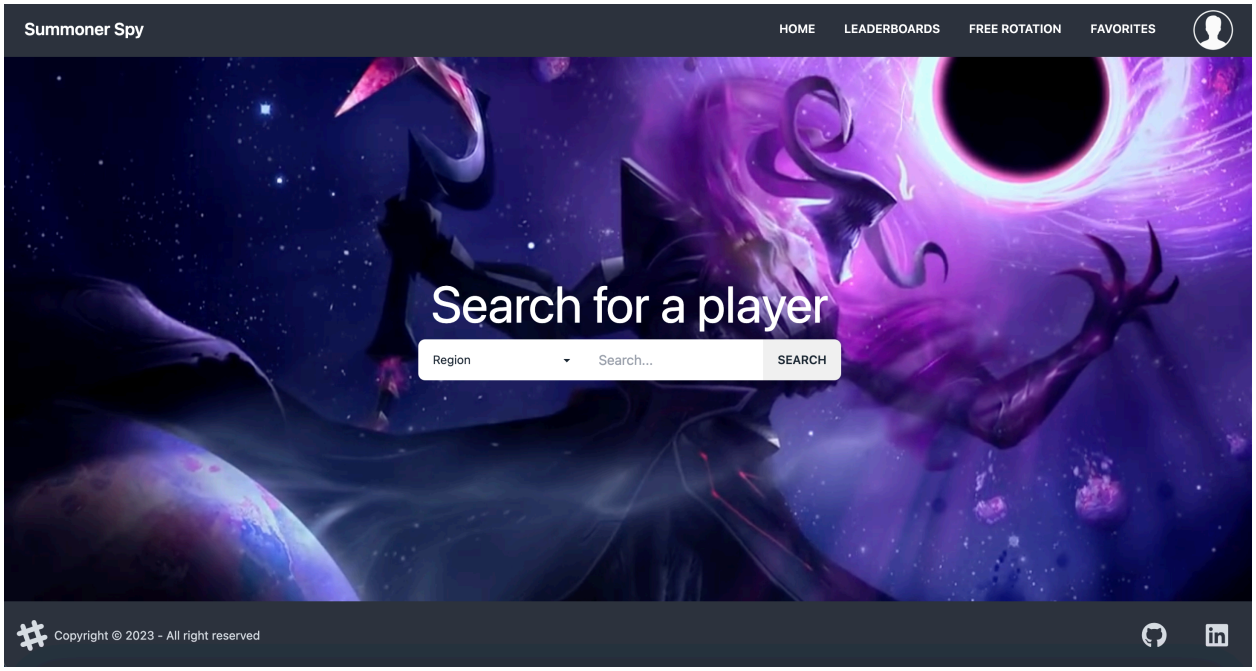


Slika 10. Header komponenta

Slika 11. Footer komponenta

4.2.4. Landing komponenta

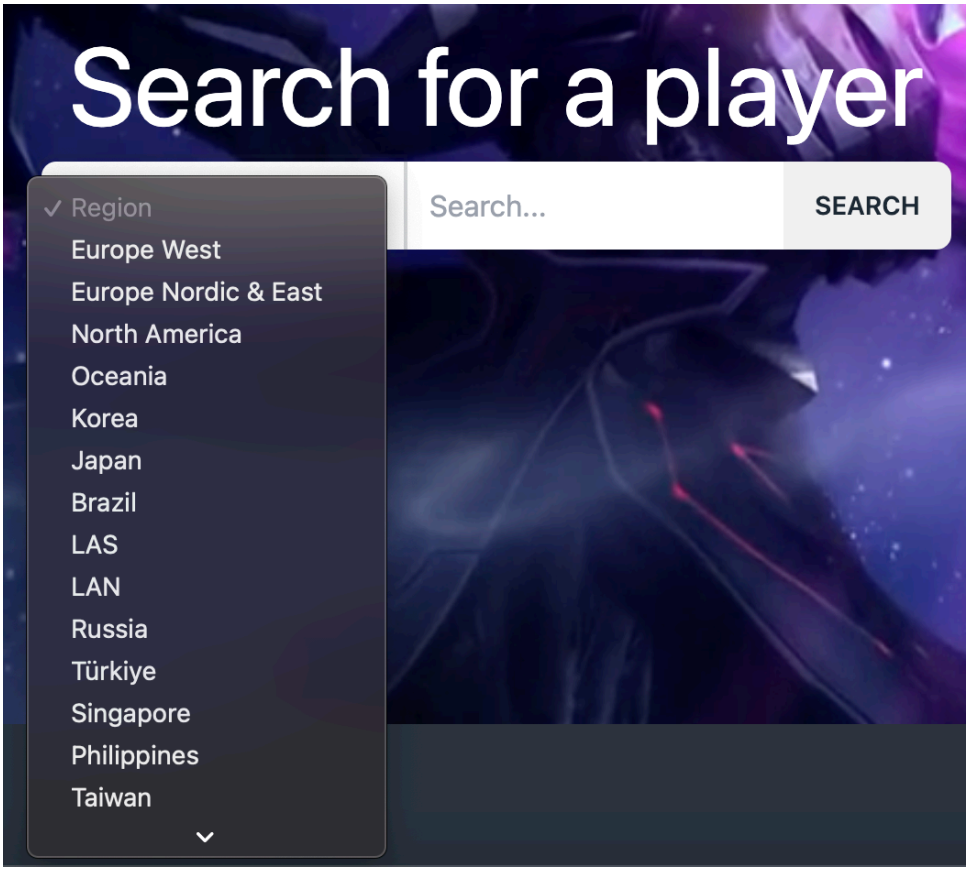
Landing komponenta, odnosno početna stranica, također kao i „header“ komponenta sadrži „search“ komponentu za pretragu koja omogućuje korisnicima da odaberu regiju i unesu ime igrača kojeg žele pregledati. Ova stranica omogućuje brz i jednostavan početak istraživanja igrača i njihovih podataka (slika 12.).



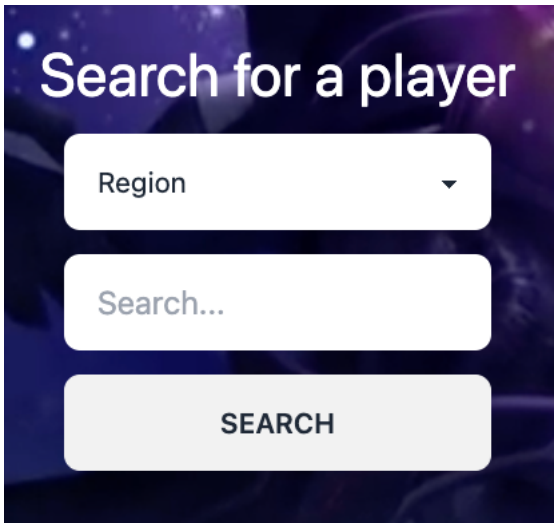
Slika 12. Landing komponenta

4.2.5. Search komponenta

Već objašnjena komponenta za pretragu omogućuje korisnicima da odaberu regiju i upišu ime igrača koje žele pregledati. Ova komponenta inicira traženje igračevih informacija i prelazak na stranicu profila igrača. Desktop i mobilni prikaz ove komponente na početnoj stranici možemo vidjeti na slikama 13. i 14.



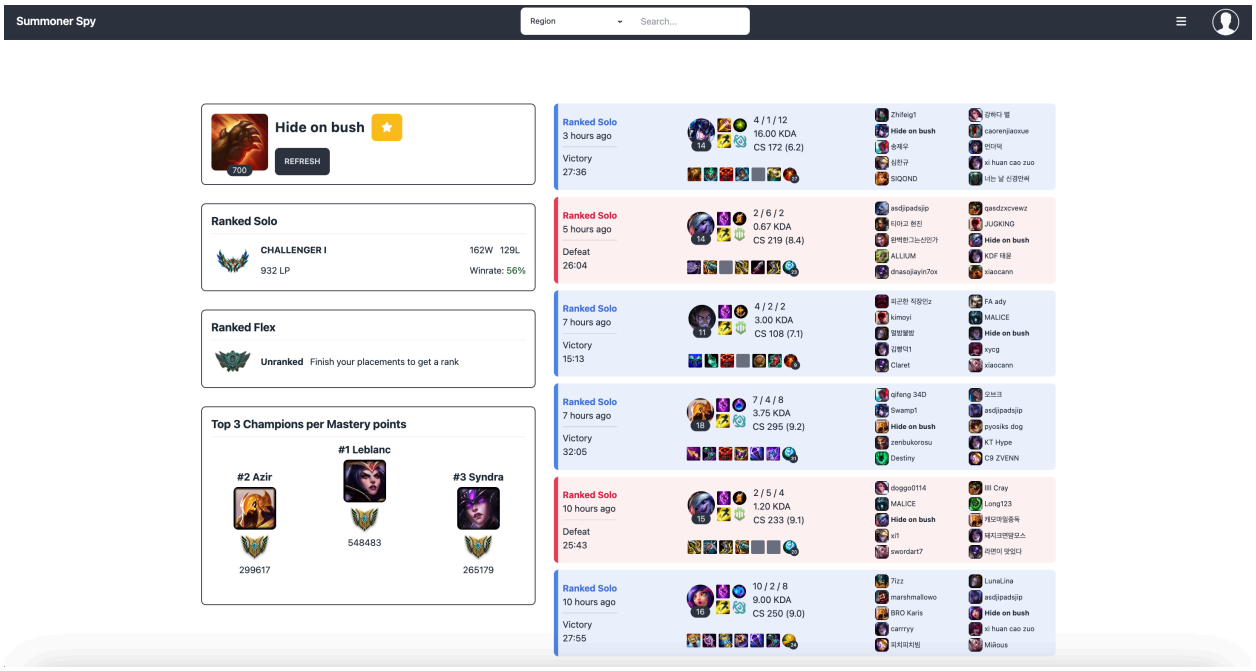
Slika 13. Search komponenta - desktop prikaz



Slika 14. Search komponenta - mobilni prikaz

4.2.6. Summoner komponenta

Summoner stranica prikazuje osnovne informacije o profilu igrača u igri "League of Legends". Korisnici mogu vidjeti podatke o igračevom ranku u različitim modovima, povijest odigranih mečeva i njegove top 3 heroje po ostvarenim bodovima (slika 15.).



Slika 15. Summoner komponenta

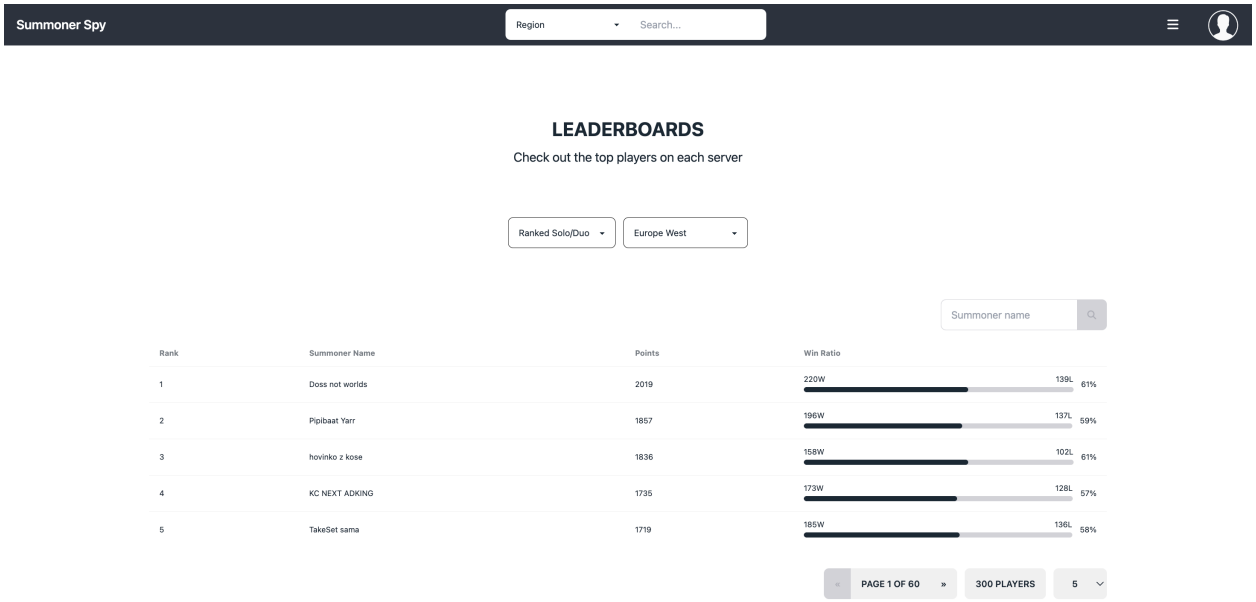
4.2.7. Match komponenta

Match komponenta prikazuje detalje svakog individualnog odigranog meča, a poziva se u „summoner“ komponenti. Detalji odigranog meča uključuju rezultat, trajanje, gamemode, heroj kojeg je igrač igrao, njegov rezultat, farmu, spellove, rune i kupljene iteme. Također, prikazuju se sortirane liste igrača po timovima i pozicijama. Ovi podaci su izdvojeni u zasebnu komponentu iz razloga što zahtjevaju dosta puno logike kojom bi se pretrpala „summoner“ komponenta, a sučelje komponente je prikazano na slici 16.

Slika 16. Match komponenta

4.2.8. Leaderboards komponenta

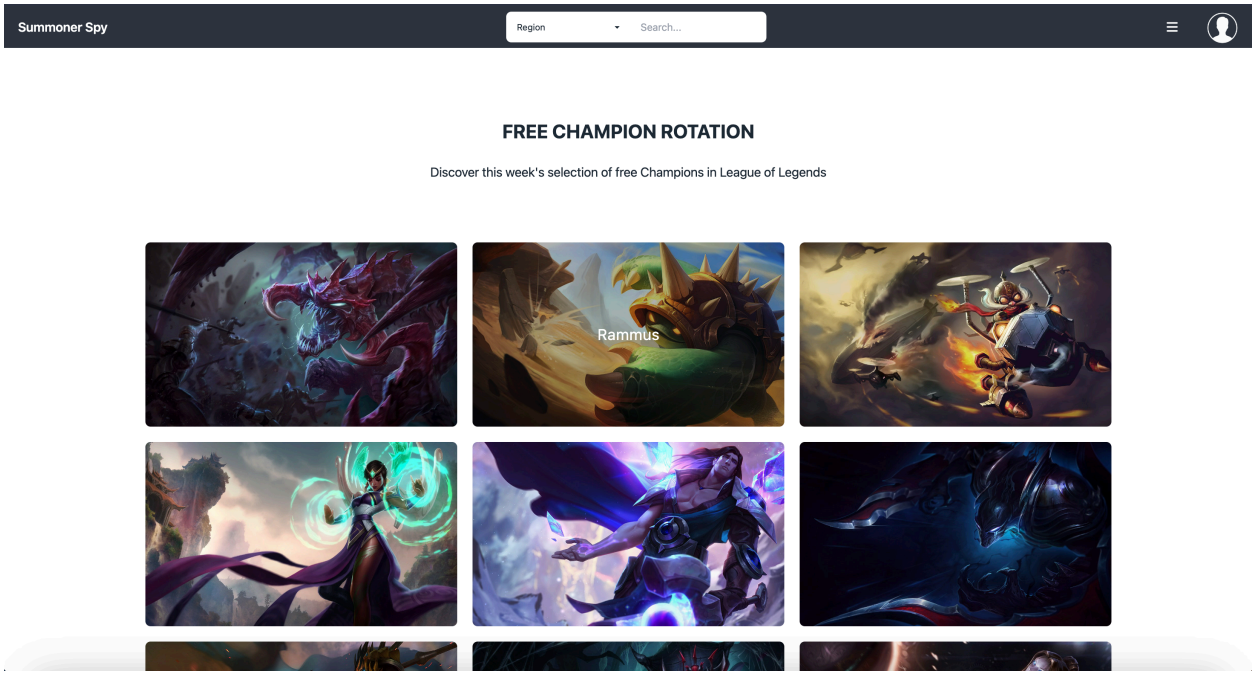
Leaderboards komponenta prikazuje tablicu najboljih igrača za određenu regiju i rang mod (Solo/Duo ili Flex). Ovdje korisnici mogu pratiti tko se ističe u određenim modovima (slika 17.).



Slika 17. Leaderboards komponenta

4.2.9. Free rotation komponenta

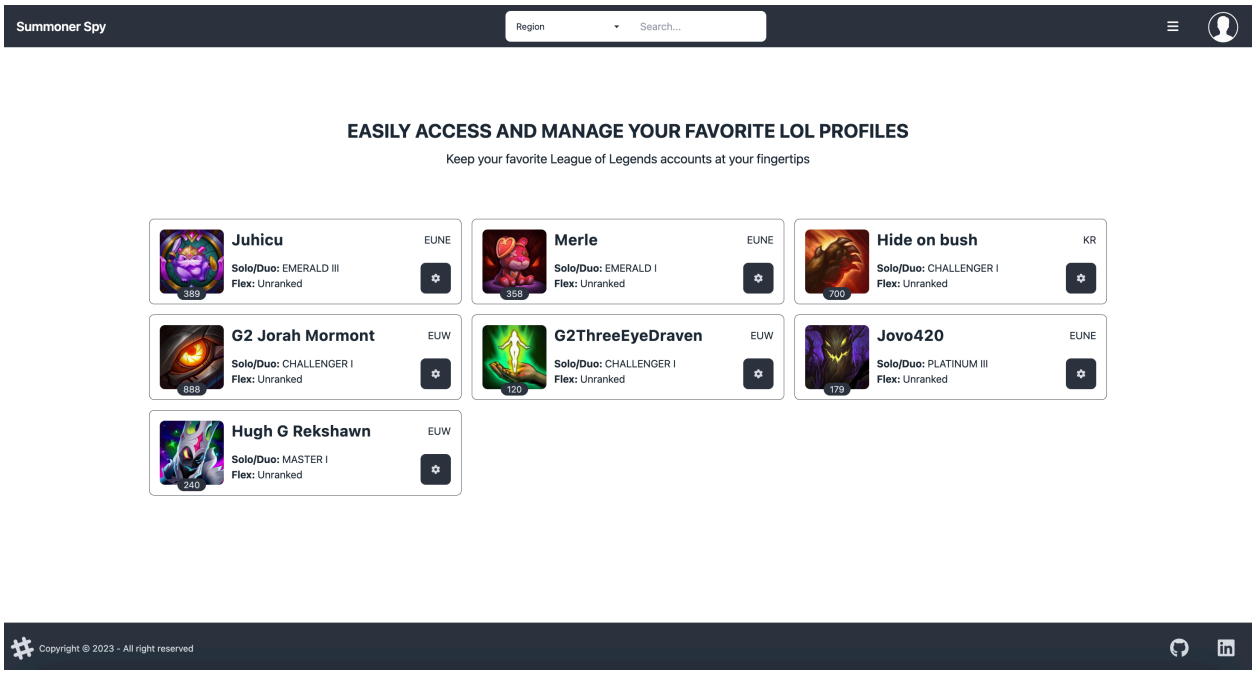
Free Rotation komponenta prikazuje slike i nazive heroja koji su besplatni za igranje tijekom određenog tjedna. Korisnici mogu brzo saznati koji heroji su trenutno dostupni za isprobavanje (slika 18.).



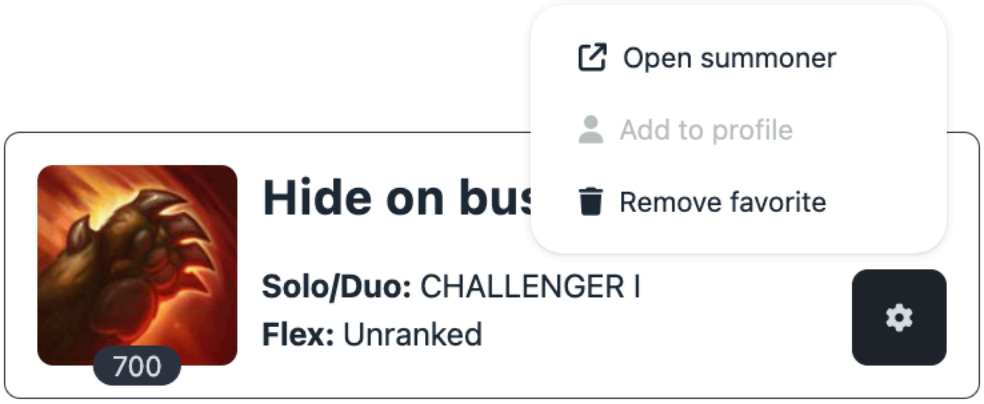
Slika 18. Free rotation komponenta

4.2.10. Favorites komponenta

Favorites komponenta prikazuje listu favoriziranih profila igrača (slika 19.). Korisnici mogu posjetiti ove profile, dodavati ili uklanjati ih kao "accountProfile" te upravljati listom favorita (slika 20.).



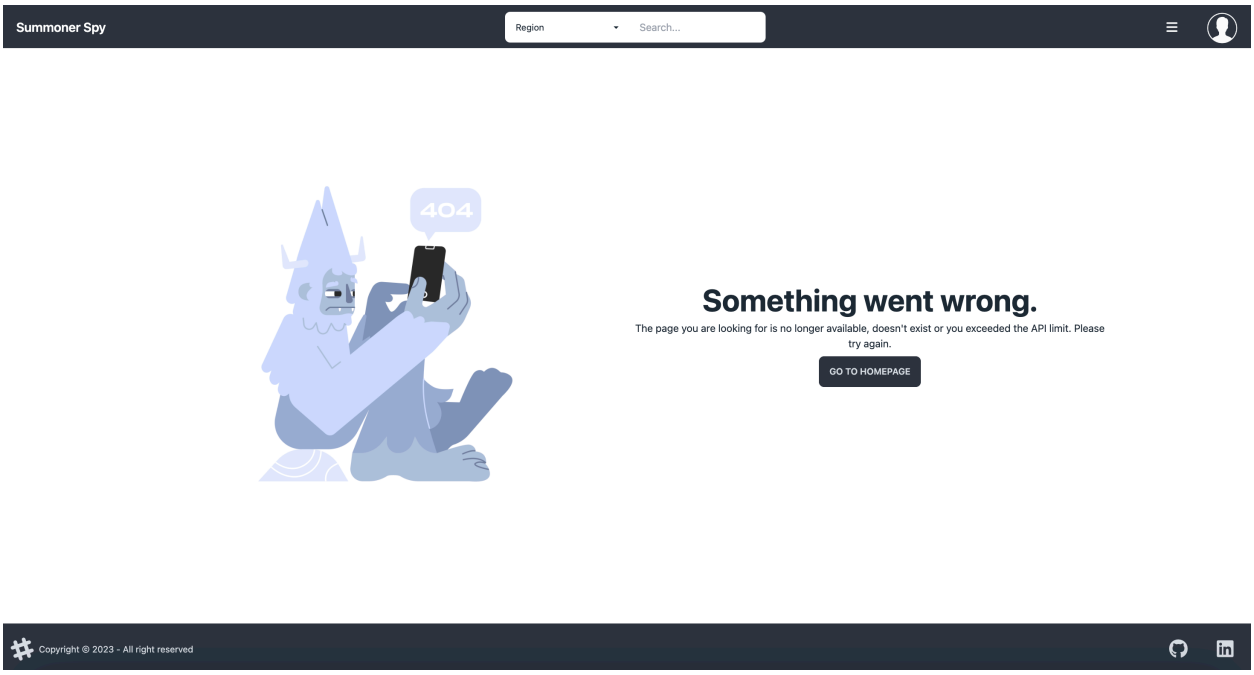
Slika 19. Favorites komponenta



Slika 20. Menu s akcijama za svaki profil

4.2.11. Not found komponenta

Not Found komponenta prikazuje se kada korisnik pokuša pristupiti nepostojećoj stranici, osiguravajući bolje korisničko iskustvo kroz jasne poruke o greškama (vidi sliku 21.).



Slika 21. Not found komponenta

5. Implementacija aplikacije

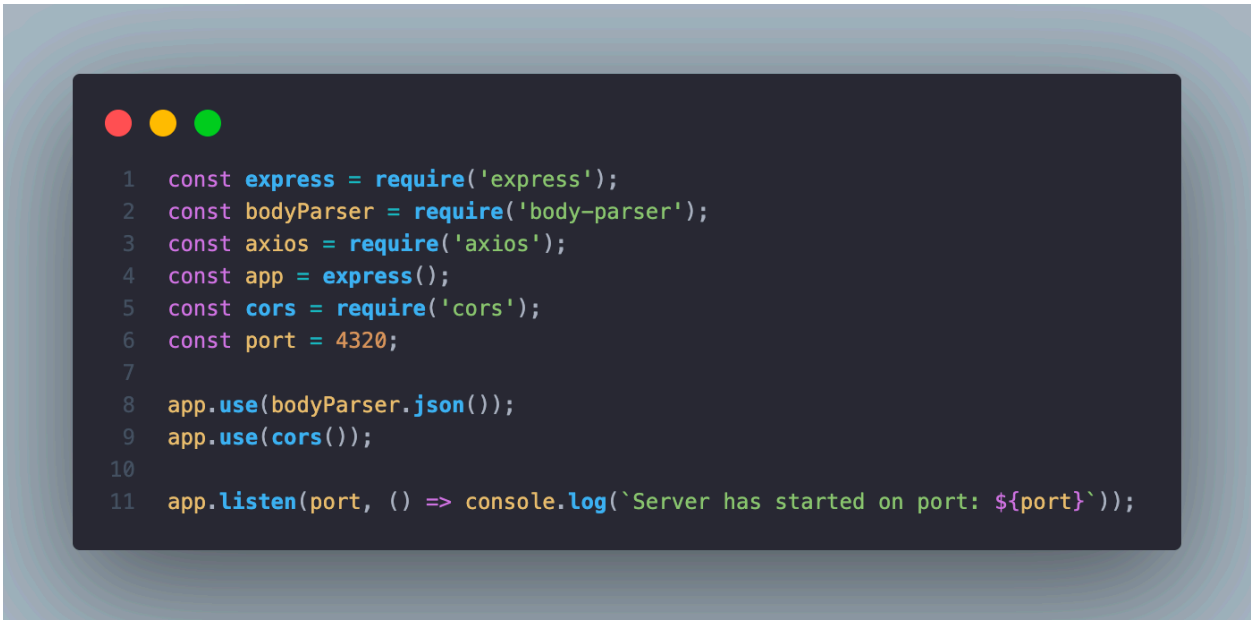
U ovom poglavlju istražit ćemo detalje implementacije "Summoner Spy" aplikacije te ćemo se usredotočiti na tehničke aspekte razvoja koristeći Angular i Node.js. Otkrit ćemo kako su ove tehnologije korištene za stvaranje glavnih funkcionalnosti aplikacije, te kako su rješavani izazovi koji su se pojavili tijekom razvojnog procesa.

5.1. Opis implementacije aplikacije

Aplikacija "Summoner Spy" razvijena je kombiniranjem različitih tehnologija kako bi se postigla visoka razina korisničke interakcije i pouzdanosti. Angular, kao front-end framework, bio je ključan za izgradnju reaktivnih komponenata i kvalitetnog korisničkog iskustva. S druge strane, Node.js je poslužio kao osnova za back-end implementaciju, dok je Firebase pružio potrebne alate za autentifikaciju i upravljanje podacima. U okviru back-end dijela, Express.js je korišten za smještaj svih API poziva prema "Riot" servisima kako bi se dobili relevantni podaci o igračima, mečevima i rangovima. Također, Firebase je odigrao ključnu ulogu u autentifikaciji korisnika, omogućavajući sigurnu prijavu, odjavu i registraciju. Također, korišten je i za upravljanje omiljenim igračima i profilima te mijenjanje korisničkih podataka. Front-end razvoj u Angularu obuhvatio je korištenje HTTP zahtjeva prema back-endu radi dohvaćanja podataka. Kroz integraciju s Express.js-om, korisnički zahtjevi odrađivani su brzo i učinkovito, omogućavajući prikaz važnih informacija o igračima i njihovim performansama na sučelju aplikacije.

5.2. Detaljan opis razvoja glavnih funkcionalnosti aplikacije korištenjem Angular-a i Node.js-a

Aplikacija „Summoner Spy“ organizirana je u dva odvojena repozitorija. Ona se sastoji od „SummonerSpy-client“ repozitorija koji čini front-end dio i „SummonerSpy-server“ repozitorija koji čini back-end dio aplikacije. Proces razvoja aplikacije započeo je s inicijalizacijom oba repozitorija. Prvo smo inicijalizirali back-end kao Node.js aplikaciju upotrebom naredbe „npm init“ u terminalu. Ova naredba generira „package.json“ dokument koji sadrži metapodatke koji opisuju naš Node.js projekt. Sljedeći korak bio je instaliranje potrebnih paketa pomoću naredbe npm install. Instalirali smo pakete kao što su „express“, „cors“, „axios“ i „nodemon“. „Express“ je Node.js framework koji omogućava izradu API-ja za slanje podataka putem API zahtjeva. „Cross-origin resource sharing“ (CORS) predstavlja mehanizam za integraciju aplikacija. CORS definira način na koji web klijentske aplikacije učitane s jedne domene mogu komunicirati s resursima na drugoj domeni. Ova karakteristika je važna kako bi back-end mogao uspješno komunicirati s front-end dijelom. „Axios“ je JavaScript library pomoću kojeg izvodimo HTTP zahtjeve. Korištenjem „Axios“-a dohvaćali smo podatke s Riot-ovog API-ja. „Nodemon“ je paket koji se koristi isključivo tijekom razvoja aplikacije, te je instaliran kao „dev Dependency“. Bez „Nodemon“-a, za svaku promjenu kôda morali bismo ručno zaustaviti i ponovno pokrenuti server kako bi promjene bile učitane. „Nodemon“ automatski ponovno pokreće server svaki put kad se izmjene spreme u datoteci koju smo pokrenuli, čime ubrzava i olakšava razvoj back-end dijela [6]. Nakon što su temeljni paketi bili instalirani, stvorili smo „server.js“ datoteku unutar istog repozitorija te dodali osnovni kôd za pokretanje aplikacije na unaprijed definiranom portu. Prikaz osnovnog kôda vidljiv je na slici 22.



```
1  const express = require('express');
2  const bodyParser = require('body-parser');
3  const axios = require('axios');
4  const app = express();
5  const cors = require('cors');
6  const port = 4320;
7
8  app.use(bodyParser.json());
9  app.use(cors());
10
11 app.listen(port, () => console.log(`Server has started on port: ${port}`));
```

Slika 22. Osnovni kod za express.js aplikaciju

Naknadno smo u „server.js“ datoteci implementirali prvi zahtjev prema Riot-ovom API-ju. Ovaj zahtjev prihvaća dva parametra - „region“, koji označava odabranu regiju, te „summonerName“, koji označava ime igrača kojeg korisnik želi pretražiti. Koristeći ove parametre, dinamički konstruiramo URL za Riot-ov API i koristimo Axios za slanje GET zahtjeva i dohvaćanje traženih podataka. U slučaju da ne uspijemo dohvatiti podatke, šaljemo prazan objekt na front-end. Na taj način, front-end dobiva obavijest o neuspješnoj pretrazi i prikazuje odgovarajuće poruke pogreške kako bi korisnicima pružili jasne informacije. Cijeli kôd kojeg smo upravo opisali prikazan je na slici 23.

```
app.get('/summoner/:region/:summonerName', (req, res) => {
    const { region, summonerName } = req.params;
    axios.get(`https://${region}.api.riotgames.com/lol/summoner/v4/summoners/by-name/${summonerName}`)
        .then(response => {
            res.status(200).send(response.data);
        })
        .catch(error => {
            console.log(error);
            res.send({});
        });
});
```

Slika 23. Dohvaćanje osnovnih podataka o igraču u backe-end-u

Nakon uspješnog izvršenja prvog zahtjeva za podacima, uslijedila je inicijalizacija Angular projekta. Početno smo instalirali Angular CLI (Command Line Interface), alat koji pruža developerima jednostavno i interaktivno sučelje za izvršavanje raznih zadataka, kao što su stvaranje projekta, izrada komponenti te lokalno pokretanje projekta. Instalacija Angular CLI-a izvršava se koristeći naredbu „npm install -g @angular/cli“, a ispravnost instalacije može se provjeriti s „ng version“ naredbom [7].

Prvi korak u stvaranju projekta bio je izvođenje naredbe „ng new SummonerSpy-client“, koja je osmislila osnovnu strukturu projekta. Nakon što smo osigurali da se projekt može pokrenuti naredbom „ng serve“, integrirali smo Tailwind CSS slijedeći njihovu dokumentaciju te uključili DaisyUI kao dodatak. Tako smo postigli brz i dosljedan stilizacijski okvir za daljnji razvoj. Nadalje, kreirali smo prvu komponentu, nazvanu „search“, koristeći naredbu „ng generate component search“, ili kraće „ng g c search“. Ova komponenta je dizajnirana s dva input polja: select i tekstualno polje te gumb koji pokreće funkciju pretraživanja (vidi sliku 24. i 25.). Korisnici kroz select biraju regiju, dok kroz tekstualno polje unose ime igrača.

```
<div class="join">
      <select class="select join-item w-auto" (change)="onRegionSelect($event.target.value)">
        <option disabled selected>Region</option>
        <option *ngFor="let region of regionsList" [value]="region.name">{{region.name}}</option>
      </select>
      <div>
        <input class="input join-item w-auto"
               type="text"
               placeholder="Search..."
               [(ngModel)]="summonerName"
               (keyup.enter)="search()">
      </div>
      <button *ngIf="!isHeader" class="join-item btn" (click)="search()">
        Search
      </button>
    </div>
```

Slika 24. HTML kod search komponente

```
search() {
    if (this.selectedRegion && this.summonerName.length > 0) {
      const url: string = `summoners/${this.selectedRegion.shorthand}/${this.summonerName}`;
      this.router.navigateByUrl(url).then(r => console.log(r));
    }
  }
```

Slika 25. Funkcija search

U ovom trenutku, primijetili smo dosta podataka povezanih s regijama. Bio nam je potreban puni naziv svake regije koja će biti prikazana u select-u. Također, trebale su nam skraćenice za svaku regiju (server) koje će se koristiti u URL-ovima profila igrača, kao i kodovi regija koje je Riot koristio za dohvaćanje igračevih podataka. Nadalje, bilo nam je nužno imati i glavnu regiju za svaki server. Kako bismo se nosili s ovim potrebama, osmislili smo model podataka pod nazivom „Region“. Ovaj model je koncipiran kao objekt koji sadrži sve relevantne atribute za regije, obuhvaćajući naziv, skraćenicu, kôd i glavnu regiju. Ovaj pristup omogućio nam je organiziranje i pravilno upravljanje svim važnim aspektima vezanim za regije unutar naše aplikacije, a na slici 26. je vidljivo kako je taj model podataka izveden u kôdu.

Slika 26. Region model

Ovaj model smo implementirali unutar „constants.ts“ datoteke. U toj datoteci smo izvezli varijablu nazvanu „regions“, koja je definirana kao niz elemenata tipa „Region“. Unutar ovog niza smo pohranili pojedinačne objekte za svaki dostupni server u igri (slika 27.).

```
import {Region} from "../models/region.model";

export const regions: Region[] = [
  {
    name: 'Europe West',
    shorthand: 'euw',
    code: 'euw1',
    majorRegion: 'europe',
    leaderboardRegion: true
  },
  {
    name: 'Europe Nordic & East',
    shorthand: 'eune',
    code: 'eun1',
    majorRegion: 'europe',
    leaderboardRegion: true
  }, ...];
```

Slika 27. Varijabla regions

Izvođenjem ovog pristupa, varijabla „regions“ postaje ključni izvor podataka vezanih za specifične regije u bilo kojoj komponenti unutar aplikacije. Prvi primjer primjene ovog koncepta je vidljiv u „summoner“ komponenti. Kada se izvrši funkcija „search“, otvara se nova stranica čiji URL sadrži skraćenicu regije i ime igrača koje je korisnik unio. Unutar Angularove funkcije „OnInit“, koja se poziva nakon inicijalizacije „summoner“ komponente, izvršavamo pretplatu na promjene rute stranice. Također, izvučemo parametre iz rute i pohranjujemo ih u odvojene varijable kako bismo ih mogli upotrijebiti prilikom dohvaćanja podataka. U situacijama kada pretražujemo igrača koji se nalazi na „Europe Nordic & East“ serveru, trenutno bismo imali samo „eune“ kao vrijednost regije. Međutim, budući da posjedujemo varijablu koja sadrži podatke za sve regije, možemo pristupiti objektu koji sadrži sve važne informacije. Ovo postižemo korištenjem Javascript funkcije „find“, koja nam omogućava identifikaciju odgovarajućeg objekta na temelju odabrane skraćenice regije. Objašnjeni kôd je vidljiv na slici 28.

```
async ngOnInit() {
    this.route.params.subscribe(async params => {
      this.summonerName = params['summonerName'];
      this.selectedRegion = regions.find(region => region.shorthand === this.route.snapshot.params['regionCode']);
    });
  }
```

Slika 28. Dohvaćanje imena igrača i regije u summoner komponenti

Kada je dostupan osnovni set podataka za izradu prvog GET zahtjeva prema back-end-u, možemo kreirati funkciju unutar front-end-a koja će obavljati ovu zadaću. Budući da ćemo ovu funkcionalnost koristiti ne samo u jednoj komponenti, već i u drugim dijelovima aplikacije, smisleno je izdvojiti je iz komponente i organizirati kao zaseban entitet. Ovako nešto se postiže stvaranjem servisa, posebne datoteke koju možemo pozivati iz različitih dijelova aplikacije prema potrebi. Kroz ovakav pristup, servis „pamti“ logiku i funkcionalnosti koje se mogu koristiti iz raznih komponenata, čime se smanjuje dupliranje kôda i olakšava održavanje. Servis u Angular-u predstavlja način za centralizirano upravljanje raznim funkcionalnostima i poslovnom logikom. Kroz servise, razdvajamo funkcionalnosti od komponenata te omogućujemo ponovno korištenje istih funkcija na više mjesta unutar aplikacije. Struktura servisa uključuje definiranje funkcija i metoda koje obavljaju specifične zadatke. Te funkcije se zatim mogu uključiti u različite komponente kroz injektiranje servisa u konstruktor komponenata. Dakle, u „summoner“ komponenti, nakon njene inicijalizacije, dobivamo odabranu regiju i ime igrača, importamo naš novi servis te ga deklariramo u konstruktoru komponente. Zatim, pozovemo funkciju „getSummoner()“ iz servisa (kod funkcije vidljiv na slici 30.) koja nam vraća osnovne podatke o igraču (dobiveni podaci prikazani na slici 31.) te njih spremimo u varijablu čije vrijednosti ćemo prikazati na našoj stranici (cijeli proces prikazan na slici 29.).

```
import ...
import {ApiService} from "../services/api.service";

@Component({
  selector: 'app-summoner',
  templateUrl: './summoner.component.html',
  styleUrls: ['./summoner.component.css']
})
export class SummonerComponent implements OnInit{
  isLoading: boolean = false;
  summonerName: string;
  selectedRegion: Region;
  summoner: any;

  constructor(private api: ApiService) { }

  async ngOnInit() {
    this.route.params.subscribe(async params => {
      this.isLoading = true;

      this.summonerName = params['summonerName'];
      this.selectedRegion = regions.find(region => region.shorthand === this.route.snapshot.params['regionCode']);
      let response = await this.api.getSummoner(this.selectedRegion.code, this.summonerName);

      this.isLoading = false;
    });
  }
}
```

Slika 29. Pozivanje funkcije iz servisa

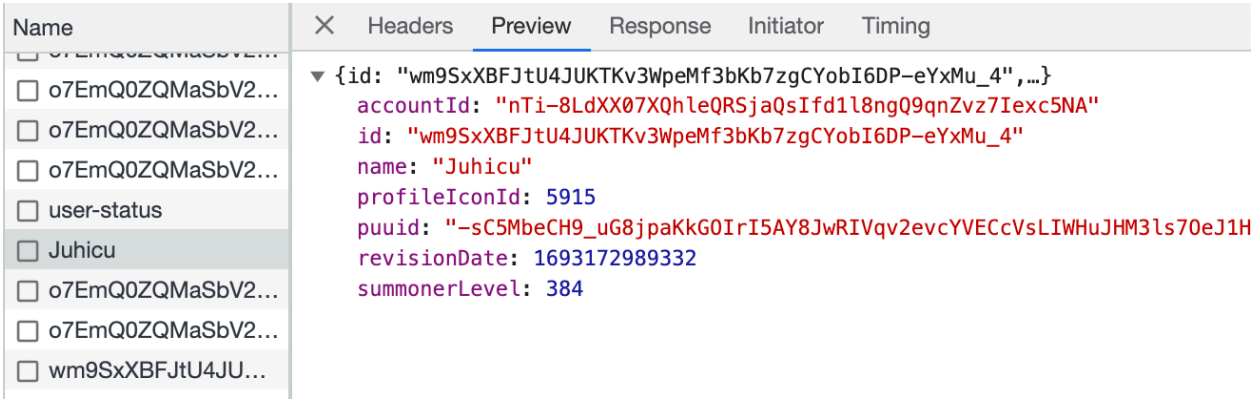
```
import { Injectable } from '@angular/core';
import {HttpClient} from '@angular/common/http';
import {Subscription} from "rxjs";
import {endpoint} from "../utils/constants";

@Injectable({
  providedIn: 'root'
})
export class ApiService {

  constructor(private http: HttpClient) {}

  getSummoner(selectedRegion: string, summonerName:string): Promise<any> {
    return new Promise(resolve => {
      const a = this.http.get(`${endpoint}/summoner/${selectedRegion}/${summonerName}`).subscribe((data) => {
        resolve(data);
      });
    });
  }
}
```

Slika 30. Funkcija getSummoner u servisu



Slika 31. Podaci primljeni sa back-end-a

Na isti način, razvili smo mehanizme dohvaćanja i obrade svih ostalih podataka koji su nam bili ključni za realizaciju ovog projekta. To je obuhvatilo dohvaćanje informacija o rangu i bodovima, broju bodova za tri najbolja heroja, te povijesti odigranih igara. U tu svrhu, napravili smo funkciju koja omogućava dohvaćanje identifikatora igara (match ID-jeva) pomoću jednog API poziva. Ovaj poziv je konfiguriran prema našim potrebama, omogućujući nam dohvaćanje željenog broja identifikatora putem parametara „start“ i „count“. Nakon što smo prikupili ID-jeve odigranih igara, koristimo drugi API poziv kako bismo dohvatili sve relevantne podatke za svaku pojedinu igru. Za ovaj poziv, osim globalnog parametra za regiju, moramo pružiti i svaki pojedini ID igre. To smo postigli pozivajući funkciju za dohvaćanje tih podataka iz backend-a unutar petlje (vidi sliku 32.). Ovaj proces nam omogućava da izvučemo sve važne informacije o određenoj igri.

```
async loadMatches() {
    this.matchIds = await this.api.getAllMatches(this.selectedRegion.majorRegion, this.summoner.puuid, this.start.toString(), this.count.toString());

    for (let i = 0; i < this.matchIds.length; i++) {
      const singleMatch: any = await this.api.getMatch(this.selectedRegion.majorRegion, this.matchIds[i]);
      for (let participant of singleMatch.info.participants) {
        if (participant.puuid === this.summoner.puuid) {
          singleMatch.info.isWin = participant.win;
        }
      }
      this.matches.push(singleMatch);
    }
  }
```

Slika 32. Dohvaćanje podataka svake igre

Dalje, izradili smo dodatne API pozive vezane uz samu igru, kao što je dohvaćanje informacija o besplatnim herojima i najboljim igračima. Posljednji ključni korak u ovom procesu je implementacija važnog servisa koji se bavi komunikacijom s Firebase-om preko backend-a.

Ovim servisom obuhvaćen je proces autentifikacije korisnika te omogućava dodavanje i uklanjanje igrača iz liste favorita, kao i dodavanje i uklanjanje profilnog računa. Primjer korištenja ovog servisa možemo vidjeti kod dodavanja i micanja profila igrača iz favorita. Ova funkcionalnost je dostupna jedino ako je korisnik ulogiran, te je ta provjera također primjer Firebase funkcionalnosti. Prilikom učitavanja stranice igrača, poziva se funkcija „userStatus()“ (slika 33.). Ona vraća podatke o trenutno ulogiranom korisniku, ako ga ima. Ukoliko nema, botun za dodavanje u favorite nije prikazan.

Slika 33. Slanje statusa o trenutnom korisniku

Dodavanje igrača u favorite započinje pretraživanjem igrača i dolaskom na njegovu stranicu. Uz ime igrača i njegovu profilnu sliku, nalazi se botun kojim započinjemo proces dodavanja igrača u favorite. Klikom se pokreće funkcija „toggleFavorite()“ u TypeScript datoteci ove komponente (slika 34.).

Slika 34. Dodavanje / micanje igrača iz favorita na front-end-u

Kada se funkcija pozove, prvo se provjerava trenutno stanje varijable „isFavorite“. Ako je ova varijabla postavljena na „false“, to znači da igrač trenutno nije označen kao favorit korisnika. U tom slučaju, funkcija mijenja vrijednost varijable „isFavorite“ na „true“, što označava da će igrač biti dodan u listu favorita. Nakon što je status varijable „isFavorite“ ažuriran, koristi se „await“ ključna riječ kako bi se izvršila asinkrona operacija dodavanja igrača u listu favorita.

Poziva se metoda „addToFavorites“ iz Firebase servisa, pri čemu se prenose ključni podaci kao argumenti, uključujući identifikator trenutno prijavljenog korisnika, kôd odabrane regije te ime igrača. Na back-end-u dobijemo u parametrima identifikator ulogiranog korisnika tako da možemo pronaći njegov zapis favorita u Firebase kolekciji (slika 35.).

Slika 35. Dohvaćanje favorites kolekcije

```
app.post('/add-favorites/:userId/:region/:name', async(req, res) => {
    const { userId, region, name } = req.params;
    try {
        const snapshot = await favoritesCollection.doc(userId).get();

        if (!snapshot.exists) {
            await favoritesCollection.doc(userId).set({
                favorites: [{
                    region: region,
                    name: name
                }]
            });
            return res.status(200).json({ message: 'Document created' });
        }

        const documentData = snapshot.data();
        const favorites = documentData.favorites;

        favorites.push({
            region: region,
            name: name
        });

        await favoritesCollection.doc(userId).update({
            favorites: favorites
        });

        return res.status(200).json({ message: 'Favorite added' });
    } catch (error) {
        console.error('Error getting document:', error);
        return res.status(500).json({ message: 'Internal server error' });
    }
});
```

Slika 36. Dodavanje igrača u favorite u back-end-u

Unutar try-catch bloka, počinjemo s dohvaćanjem dokumenta unutar kolekcije „favorites“, koji je identificiran korisničkim ID-om. U slučaju da takav dokument ne postoji, stvara se novi dokument koji u sebi sadrži niz „favorites“ te se u to polje dodaje novi „map“ element s atributima „region“ i „name“. U slučaju da dokument već postoji, novi „map“ element se jednostavno dodaje u postojeći niz „favorites“ (slika 36.).

6. Testiranje aplikacije

Testiranje aplikacije igra ključnu ulogu u osiguravanju njenog kvaliteta i funkcionalnosti prije nego što bude puštena u produkciju. U ovom poglavlju opisujemo pristup i metode testiranja aplikacije "Summoner Spy", kao i rezultate testiranja te analizu dobivenih rezultata.

6.1. Opis testiranja aplikacije

Cilj testiranja aplikacije "Summoner Spy" bio je potvrditi da aplikacija ispravno obavlja svoje osnovne funkcionalnosti te da pruža korisnicima očekivano iskustvo prilikom interakcije s njom. Za potrebe testiranja, koristili smo metodu funkcionalnog testiranja koje uključuje realne korisničke scenarije.

6.2. Metode testiranja

Testiranje je provedeno tako da je korisnicima dostavljen link prema aplikaciji, zajedno s nizom zadataka koje su trebali izvršiti. Prvi zadatak bio je da korisnici pronađu vlastiti profil igrača ukoliko igraju igru League of Legends. U suprotnom, navedeno je bilo ime igrača koji je služio kao ogledni primjer. Nakon toga, zadatak je obuhvaćao registraciju s vlastitim email-om te dodavanje zadnjeg pretraženog igrača u favorite i kao „profileAccount“. Dodatno, korisnici su bili upućeni da pronađu prvog rangiranog igrača u gamemode-u Ranked Solo/Duo na Europe West serveru te da od trenutno besplatnih heroja odaberu kojeg bi željeli igrati.

6.3. Rezultati testiranja i njihova analiza

Tijekom testiranja aplikacije "Summoner Spy", identificirano je nekoliko bugova koji su utjecali na funkcionalnost i vizualni izgled aplikacije. Većina ovih bugova bila je povezana s dizajnerskim aspektima, kao što su problema s indeksima na sidebar izborniku, što je rezultiralo preklapanjem teksta na stranici za besplatne heroje s sidebar-om. Važno je napomenuti da su ovi dizajnerski nedostaci brzo prepoznati i ispravljeni kako bi osigurali bolje korisničko iskustvo. Međutim, najznačajniji problem koji smo primijetili tijekom testiranja bio je vezan uz dolazak novog gamemode-a, Arene. Prilikom učitavanja komponente za odigranu igru Arene na stranici "summoner", primijetili smo da se aplikacija ponaša nepredvidljivo i da dolazi do različitih problema s prikazom podataka. Ovaj bug predstavlja značajnu poteškoću u ispravnom funkcioniranju aplikacije, budući da ne pruža dosljedno iskustvo korisnicima.

7. Zaključak

Ovaj rad rezultirao je stvaranjem funkcionalne web aplikacije "Summoner Spy" koja omogućuje korisnicima praćenje igrača, njihovih statistika i odigranih igara u League of Legends. Cijeli proces razvoja obuhvatio je kreiranje kompletne web aplikacije. Koristili smo Angular za front-end, Node.js za back-end, te Firebase za autentifikaciju i bazu podataka. Kombinacija ovih tehnologija omogućila je stvaranje funkcionalne i pouzdane aplikacije. Implementirali smo složenu funkcionalnost za dohvaćanje podataka o igračima i njihovim odigranim igrama putem Riot-ovog API-ja.

Ovo je omogućilo korisnicima praćenje statistika igrača u igri League of Legends. Pružili smo korisnicima mogućnost registracije, prijave i upravljanja vlastitim profilima. Ovo uključuje dodavanje igrača u listu favorita i postavljanje profila, što doprinosi boljem korisničkom iskustvu. Kroz proces razvoja, analizirali smo zahtjeve aplikacije. Također smo proveli testiranje s korisnicima kako bismo identificirali i ispravili pronađene bugove i poboljšali stabilnost i funkcionalnost aplikacije. Unatoč postignućima, "Summoner Spy" ima potencijal za daljnje unaprjeđenje. To uključuje proširenje funkcionalnosti praćenja igrača dodavanjem dodatnih statistika poput prošlih rangova, grafova o kretanju u rangu, dodatnim podacima o pojedinoj odigranoj igri i sl.

8. Popis literature

[1] R. Games, »How To Play - League of Legends,« Riot Games, [Mrežno]. Available: https://www.leagueoflegends.com/en-us/how-to-play/. [Pokušaj pristupa 9 8 2023].

[2] Google, »Angular - Introduction to the Angular docs,« Google, [Mrežno]. Available: https://angular.io/docs. [Pokušaj pristupa 9 8 2023].

[3] S. D. Roy, »freeCodeCamp,« 12 9 2022. [Mrežno]. Available: https://www.freecodecamp.org/news/what-is-tailwind-css-a-beginners-guide/. [Pokušaj pristupa 9 8 2023].

[4] daisyUI, »daisyUI - Tailwind CSS components,« daisyUI, [Mrežno]. Available: https://daisyui.com/. [Pokušaj pristupa 9 8 2023].

[5] Souvik, »Famous Companies Using MEAN Tech Stack for Development,« 2 2 2022. [Mrežno]. Available: https://www.rswebsols.com/mean-tech-stack-development/. [Pokušaj pristupa 10 8 2023].

[6] A. L. Morre, »Firebase Realtime Chat | Build and deploy with Firebase, NextJS and Chat Engine (Best UI 😁),« 20 2 2023. [Mrežno]. Available: https://www.youtube.com/watch?v=y3iWpRRt_sI. [Pokušaj pristupa 15 8 2023].

[7] Google, Google, Angular - Setting up the local enviroment and workspace. [Mrežno]. Available: https://angular.io/guide/setup-local. [Pokušaj pristupa 15 8 2023].

[8] C. S. Mullins, »The Complete Guide to DBA Practices and Procedures,« u *Database Administration*, New Jersey, 2013, pp. 642-674.

9. Popis slika

10. Prilozi

Poveznica na aplikaciju: https://summoner-spy.vercel.app/