Aplikacija za strateško planiranje i analizu hokejaških utakmica

Binder, Karla

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

APLIKACIJA ZA STRATEŠKO PLANIRANJE I ANALIZU HOKEJAŠKIH UTAKMICA

Završni rad

Karla Binder

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 14.09.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Karla Binder
Studij, smjer:	Računalno inženjerstvo
Mat. br. Pristupnika, godina upisa:	R 4321, 16.10.2020.
OIB Pristupnika:	89012656662
Mentor:	izv. prof. dr. sc. Ivica Lukić
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Aplikacija za strateško planiranje i analizu hokejaških utakmica
Znanstvena grana rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak završnog rad:	Omogućiti korisniku postavljanje strategije za hokejašku utakmicu. Pružiti korisniku analizu uspješnosti i efektivnosti napravljene strategije. Predstaviti mogućnosti poboljšanja kao npr. prepoznavanje prijetećih situacija, razjašnjavanje pogrešaka, pružanje pozitivnih povratnih informacija i imitiranje najbolje tehnike. Tema rezervirana za: Karla Binder
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	14.09.2023.
Datum potvrde ocjene od strane Odbora:	24.09.2023.

Potvrda mentora o predaji konačne verzije rada:	Mentor elektronički potpisao predaju konačne verzije.
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 25.09.2023.

Ime i prezime studenta:	Karla Binder
Studij:	Računalno inženjerstvo
Mat. br. studenta, godina upisa:	R 4321, 16.10.2020.
Turnitin podudaranje [%]:	11

Ovom izjavom izjavljujem da je rad pod nazivom**: Aplikacija za strateško planiranje i analizu hokejaških utakmica**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ivica Lukić

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Pratitelji nekog sporta, bilo da su najveći obožavatelji ili tek početni gledatelji vole predlagati razne strategije za koje smatraju da bi bile najučinkovitije za sportsku utakmicu. Postoje razne aplikacije koje omogućuju korisniku sudjelovanje u virtualnim sportskim ligama sastavljenih od profesionalnih sportaša iz stvarnog života. Ove aplikacije pružaju korisniku sastavljanje vlastitog tima, te postavljanje strategije za utakmice koje se trebaju odigrati. Međutim, mogućnosti provjere napravljene strategije za utakmice odigrane u stvarnom svijetu još nisu razvijene. Postoje razne stranice koje pružaju uvid u statistike igrača i sastava pomoću kojih bi gledatelji mogli uspoređivati svoje sastave, no takav način rada je poprilično nedostupan, pa čak i nerazumljiv početnim gledateljima. Cilj završnog rada je omogućiti korisniku jednostavnu i razumljivu analizu pretpostavljenih strategija hokejaških utakmica.

Strategija se može odnositi na puno stvari. Ova aplikacija se temelji na analizi sastavne strategije. Hokej na ledu je sport u kojem se tim sastoji od šest igrača: tri igrača napada, dva obrambena i vratar. Budući da u hokeju na ledu uglavnom postoji *glavni* vratar koji igra većinu utakmica, neće biti uključen u strategiju jer većinom ne postoji drugi vratar za usporedbu. Za postavljanje strategije korisnik će imati na izbor 4 tima iz NHL-a (engl. *National Hockey League*). Izabrani timovi su najbolji timovi svoje divizije za sezonu 2022-2023.

U sljedećem poglavlju opisana su slična postojeća rješenja aplikacije. U trećem poglavlju su navedene i ukratko objašnjene tehnologije i alati korišteni za izradu web aplikacije. Zatim, u četvrtom poglavlju objašnjena je implementacija tehnologija i konkretna izrada aplikacije. U petom poglavlju prikazan je način korištenja web aplikacije. Zaključak rada nalazi se u šestom poglavlju gdje je obuhvaćen cilj rada, te preporuke za poboljšanje rada i moguće nadogradnje.

1.1. Zadatak završnog rada

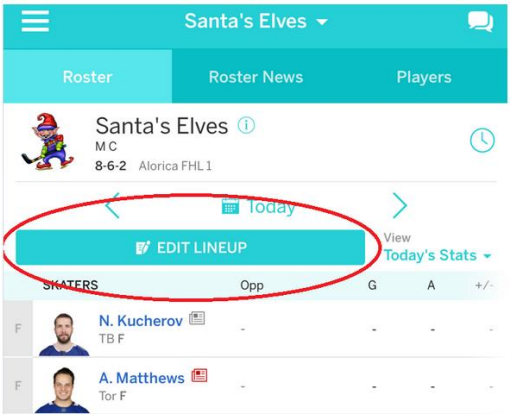
Omogućiti korisniku postavljanje strategije za hokejašku utakmicu. Pružiti korisniku analizu uspješnosti i efektivnosti napravljene strategije. Predstaviti mogućnosti poboljšanja kao npr. pružanje povratnih informacija i imitiranje najbolje tehnike.

2. SLIČNA RJEŠENJA

Jedna od najpopularnijih platforma za upravljanje postavama virtualnih timova je ESPN Fantasy Hockey. Postoje i igre za računala i za mobilne uređaje kao što su Franchise Hockey Manager 5 i Franchise Hockey 2022 koje rade na isti princip. Uz to, postoje i aplikacije za mobilne uređaje kao što su Hockey Lineup Manager i Hockey Coach Lineup and Roster. Ovi sustavi, uz mogućnost upravljanja postavama pružaju i dodatne informacije o timovima.

2.1. ESPN Fantasy Hockey

ESPN Fantasy Hockey [1] je online sportska platforma koja omogućuje korisniku upravljanje sastavom tima, pruža uvid u statistike pojedinih igrača, itd. Prije svake utakmice korisnik može izabrati početnu postavu. Postoji i zasebna stranica za tim koju je korisnik odabrao gdje su prikazani trenutni igrači,zatim je prikazan pregled njihovih izvedbi, te je omogućeno premještanje igrača između onih koji su u tom trenutku aktivni i onih koji se nalaze na klupi. Platforma omogućuje besplatno stvaranje tima, odabir igrača i upravljanje postavama, ali korištenje dodatnih značajki kao što su prikaz statistike tima i igrača, te dodatnih mogućnosti uređivanja zahtjeva plaćanje. Na slici 2.1. prikazan je izbor igrača za virtualni tim *Santa's Elves*.

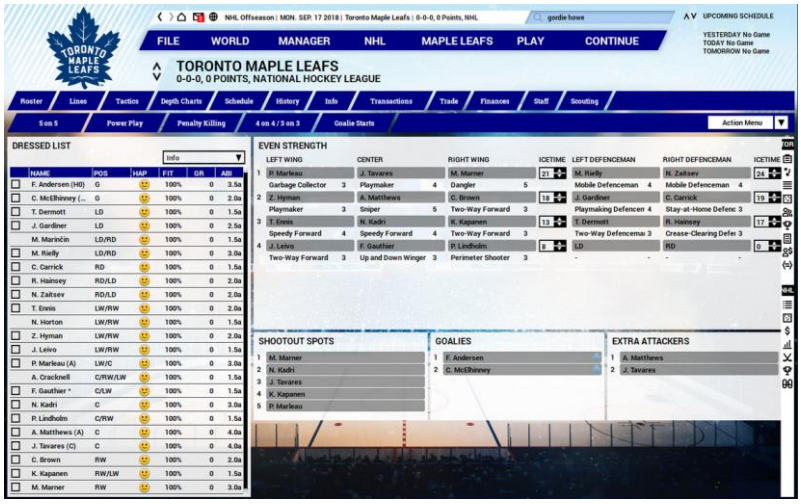


Slika 2.1. Odabir sastava u ESPN aplikaciji [1]

2.2. Franchise Hockey Manager 5

Franchise Hockey Manager 5 [2] je popularna simulacijska igra sportskog menadžmenta usmjerena na hokej na ledu. Omogućuje igračima preuzimanje uloge menadžera ili glavnog trenera profesionalnog tima u hokeju na ledu. Pruža korisnicima priliku donošenja strateških odluka,

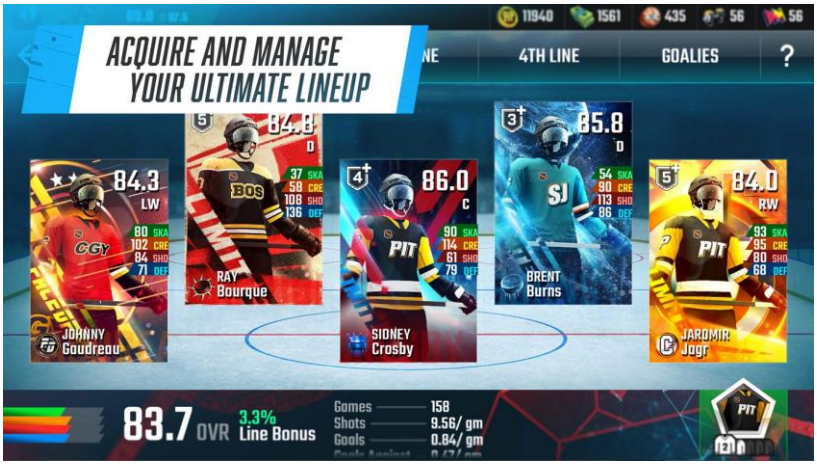
upravljanje timskim popisima, provođenje razmjene igrača, sastavljanje postava i donošenje taktičkih odluka kako bi svoj tim doveli do uspjeha. Slika 2.2. prikazuje izbor postave za hokejaški tim Toronto Maple Leafs.



Slika 2.2. Odabir postave u Franchise Hockey Manager 5 igri [2]

2.3. Franchise Hockey 2022

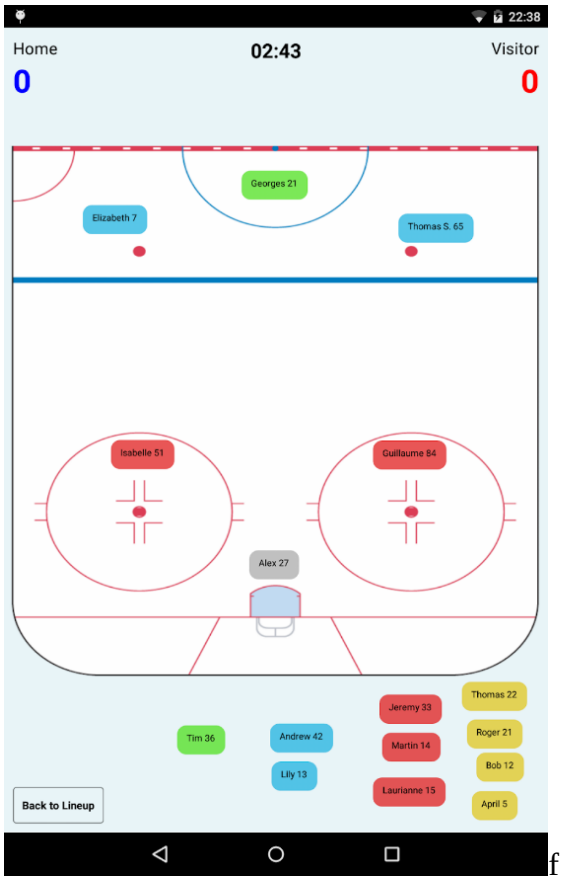
Franchise Hockey 2022 [3] je igra za mobilne uređaje koja omogućuje upravljanje hokejaškim klubom i izgradnju postava od dobivenih karata igrača. Nakon odabira željenog kluba, korisnik dobiva jednu kartu igrača, za ostale igrače korisnik mora kupiti paket gdje tek pri otvaranju toga paketa saznaje kojeg je igrača dobio. Kada prikupi dovoljno igrača može nastaviti sa sastavljanjem i treniranjem postave, a zatim igranjem utakmica. Slika 2.3. prikazuje sučelje za odabir igrača.



Slika 2.3. Odabir igrača u Franchise Hockey 2022 igri [3]

2.4. Hockey Lineup Manager

Hockey Lineup Manager [4] je aplikacija za mobilne uređaje namijenjena trenerima koji vode postave tima. Napredni grafički sustav upravljanja ove aplikacije omogućuje korisniku jednostavno rukovanje s nekoliko oblika postava. Igra omogućuje automatsko računanje vremena provedeno na ledu tijekom utakmice za svakog igrača pojedinačno, kao i prikaz statistika za svakog igrača. Također, aplikacija se može koristiti za informiranje igrača o strategiji. Slika 2.4. prikazuje postavljanje postave u mobilnoj aplikaciji.



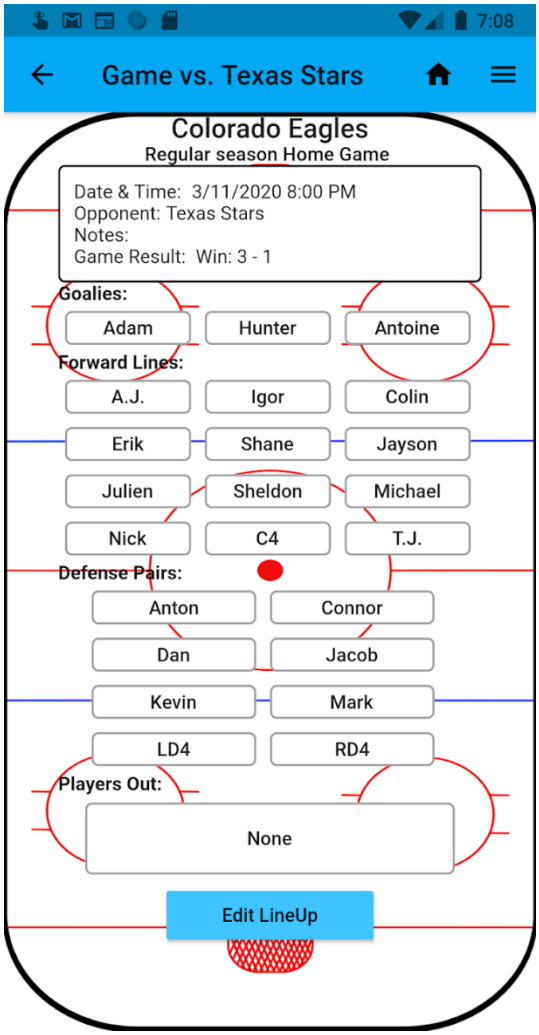
f

Slika 2.4. Postavljanje postave u Hockey Lineup Manager aplikaciji [4]

2.5. Hockey Coach Lineup and Roster

Hockey Coach Lineup and Roster [5] je aplikacija za mobilne uređaje koju su dizajnirali hokejaški treneri kako bi pomogli ostalim trenerima u upravljanju postava vlastitih timova. Aplikacija omogućuje praćenje i spremanje postava za svaki tim pojedinačno. Unutar svake igre omogućeno je praćenje svake postave zasebno. Korisnik klikom na ime igrača izmjenjuje igrače u

postavi. Ova aplikacija je u potpunosti besplatna za korištenje. Na slici 2.5. je prikazano sučelje za postavljanje postave za određenu utakmicu.



Slika 2.5. Postavljanje postave u Hockey Coach Lineup and Roster [5]

3. OPIS KORIŠTENIH TEHNOLOGIJA I ALATA

Aplikacija je razvijena u razvojnom okruženju Visual Studio Code. Razlog korištenja Visual Studio Code je zato što pruža intuitivno korisničko sučelje, sadrži dobro organiziranu bočnu traku, integrirani terminal i snažnu značajku pretraživanja i zamjene čime povećava produktivnost. Objašnjenje Visual Studio Code-a nalazi se u odjeljku 3.1.1.

Za razvoj web aplikacije korišten je razvojni okvir poznat po akronimu MERN. Više o njemu objašnjeno je u potpoglavlju 3.3. Razlog korištenja ovog razvojnog okvira je mogućnost implementacije *full-stack* web aplikacije korištenjem najvećim dijelom jednog jezika-JavaScript, više o JavaScriptu objašnjeno je u odjeljku 3.2.3. Također, ovaj razvojni okvir se široko koristi u razvoju web aplikacija, stoga postoji opsežna podrška zajednice, te široki raspon biblioteka, okvira i alata koje mogu poboljšati produktivnost razvoja.

Uz JavaScript, korišteni su HTML i CSS. Ova tri jezika baza su izgradnje web aplikacija, više o njima objašnjeno je u potpoglavlju 3.2.

Također, korištene su i razne biblioteke poput: Mongoose, Bootstrap i Axios. Više o njima nalazi se u potpoglavlju 3.4.

3.1. Razvojno okruženje

3.1.1. Visual Studio Code

Prema [6] Visual Studio Code je besplatan uređivač koda koji podržava pisanje koda u raznim jezicima kao što su Python, Java, C++, JavaScript, itd. Ističe ključne riječi u kodu različitim bojama radi lakšeg prepoznavanja obrasca kodiranja. Također, daje savjete za dovršenje linija koda i brza rješenja za učestale pogreške.

3.2. Jezici i formati

3.2.1. HTML

Prema [7] HTML (engl. *HyperText Markup Language*) je pristup temeljen na tekstu za opis strukture sadržaja unutar datoteke. Za prikaz sadržaja na web stranici koriste se posebne komponente poznate kao HTML oznake. HTML elementi uvijek imaju početnu oznaku, sadržaj u sredini i završnu oznaku. Ovo označavanje govori web pregledniku kako prikazati tekst, slike i druge oblike multimedije na web stranici. HTML podržavaju svi glavni web preglednici, uključujući web preglednike na stolnim računalima, kao i mobilne web preglednike. Prednosti

korištenja HTML-a uključuju to da ima čist i dosljedan izvorni kod, besplatan je za korištenje, te se može integrirati s drugim jezicima kao što su CSS i JavaScript.

3.2.2. CSS

Prema [8] CSS (engl. *Cascading Style Sheet*) je jezik korišten za uređivanje HTML stranica. Unutar CSS datoteke nalazi se naziv elementa za stiliziranje koji se naziva CSS selektor, zatim slijede vitičaste zagrade unutar kojih se različitim atributima kao što su veličina fonta i boja pozadine dodjeljuju vrijednosti. Ako se koristi CSS datoteka, potrebno ju je povezati s HTML datotekom pomoću HTML oznake *link*. CSS može kontrolirati kako su različiti dijelovi stranice, poput zaglavlja, podnožja, tijela i sadržaj odjeljaka raspoređeni na stranici. Ovo je iznimno korisno kada sadržaj mora biti postavljen na drugačiji način, npr. ovisno o tome gleda li se na stolnom računalu, tabletu ili pametnom telefonu.

3.2.3. JAVASCRIPT

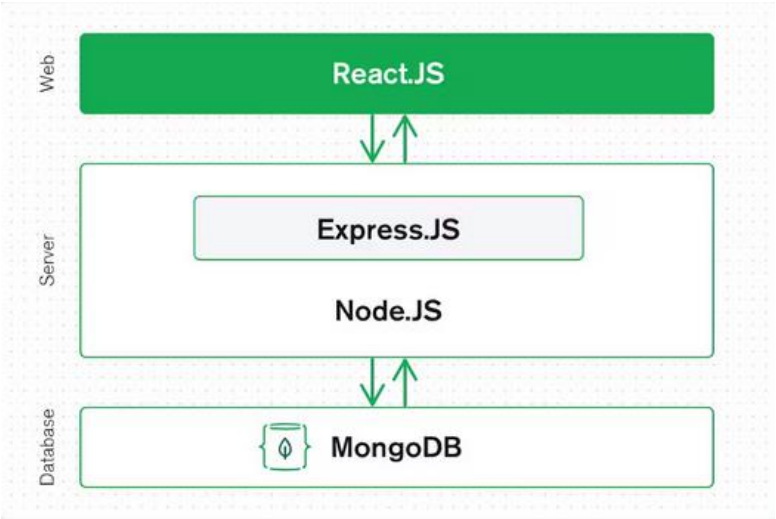
Prema [9] JavaScript je programski jezik čiji je glavni mehanizam osigurati da HTML web stranice učini interaktivnijima. JavaScript je interpretirani jezik, za razliku od jezika koji se prevode u strojni jezik, kao što su C++ i Java. HTML nudi posebnu oznaku <*SCRIPT*> koja programerima omogućuje ugradnju JavaScript-a unutar stranice. JavaScript je poznat kao *untyped* jezik, što znači da varijable mogu poprimiti različite oblike dok se program izvršava. JavaScript funkcije su analogne onome što bi drugi jezici nazvali metodama. Ove funkcije sadrže kod koji se može pokrenuti događajem na pregledniku, kao što je klik mišem, učitavanje stranice, podnošenje obrasca ili pritisak na tipku. Također je uobičajeno da jedna JavaScript funkcija pozove drugu za obavljanje dopunskog posla.

3.2.4. JSON

Prema [10] JSON (engl. *JavaScript Object Notation*) je tekstualni format za razmjenu podataka koji se koristi za razmjenu podataka između web klijenata i web poslužitelja. Format definira skup pravila strukturiranja za prikaz strukturiranih podataka. JSON se koristi kao alternativa XML-u (engl. *eXtensible Markup Language*). Format je izveden iz standarda programskog jezika JavaScript i slijedi sintaksu JavaScript objekta. JSON se sastoji od parova ime atributa: vrijednost atributa i interpunkcijskih znakova u obliku zagrada, točke sa zarezom i dvotočke.

3.3. MERN

Ova kolekcija tehnologija sastavljena je od četiri tehnologije: MongoDB, Express.js, React te Node.js. Troslojna arhitektura MERN-a [11] sastoji se od React-a koji se koristi za stvaranje dinamičkih aplikacija na klijentskoj strani, zatim Node.js i unutar njega Express.js koji se koristi za rukovanje HTTP zahtjevima i odgovorima na poslužiteljskoj strani, te MongoDB koji služi za spremanje podataka. Na slici 3.1. nalazi se prikaz strukture MERN-a.



Slika 3.1. Prikaz strukture MERN-a [11]

3.3.1. Front-end

Prema [12] *front-end* razvoj je vrsta razvoja specijalizirana za stvaranje i dizajn korisničkog sučelja (UI) i korisničkog iskustva (UX) web stranica i web aplikacija. Primarna odgovornost *front-end* razvoja je osigurati da su vizualni i interaktivni aspekti web stranice ili aplikacije jednostavni za korištenje, estetski ugodni i funkcionalno učinkoviti.

Prema [13] React je JavaScript biblioteka za stvaranje korisničkog sučelja. Omogućuje dizajniranje jednostavnih prikaza za svako stanje u vlastitoj aplikaciji. React učinkovito ažurira i prikazuje komponente kada se podaci promijene. Deklarativni pogledi čine kod predvidljivijim i lakšim za otklanjanje pogrešaka. Također omogućuje izgradnju komponenti koje upravljaju vlastitim stanjem, te daje mogućnost sastavljanja ih u složena sučelja. Budući da je logika komponente napisana u JavaScriptu umjesto u predlošcima, prosljeđivanje podataka kroz aplikaciju i zadržavanje stanja izvan DOM-a (engl. *Document Object Model*) je pojednostavljeno.

3.3.2. Back-end

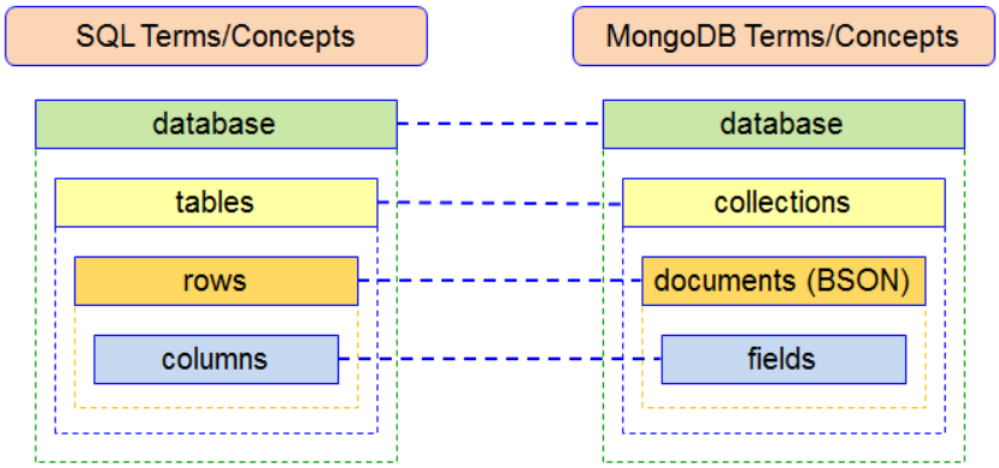
Prema [14] *back-end* razvoj pokriva logiku i integraciju web aplikacija na strani poslužitelja i aktivnosti, poput pisanja API-ja, stvaranja biblioteka i rada s komponentama sustava. *Back-end* programeri izrađuju kod koji omogućuje komunikaciju između baze podataka i aplikacije. Također, programeri održavaju pozadinu web stranice, uključujući baze podataka i servere.

Prema [15] Node.js je okruženje za izvršavanje JavaScript otvorenog koda na raznim platformama. Node.js pruža skup asinkronih ulazno-izlaznih operacija u svojoj standardnoj biblioteci koje sprječavaju blokiranje JavaScript koda. Biblioteke u Node.js su općenito napisane korištenjem ne blokirajućih modela, čineći ponašanje blokiranja iznimkom, a ne normom. Node.js ima jedinstvenu prednost jer velika količina *front-end* programera koji pišu u jeziku JavaScript na klijentskoj strani mogu pisati kod na strani poslužitelja bez upotrebe drugog jezika.

Prema [16] Express.js je minimalističan okvir (engl. *framework*) koji pridodaje funkcionalnosti Node.js web poslužitelja. Olakšava organizaciju funkcionalnosti aplikacije usmjeravanjem. Omogućuje definiranje puteva pomoću HTTP metoda i URL-ova. Uključuje niz međuprogramskih modula koji se mogu koristiti za izvršavanje dodatnih zahtjeva i odgovora.

3.3.3. Baza podataka

Prema [17] MongoDB je NoSQL (engl. *Structured Query Language*) program za upravljanje bazom podataka otvorenog koda. NoSQL se koristi kao alternativa tradicionalnim relacijskim bazama podataka. NoSQL baze podataka vrlo su korisne za rad s velikim skupovima distribuiranih podataka. MongoDB je alat koji omogućuje pohranjivanje ili dohvaćanje informacija iz dokumenta. Umjesto korištenja tablica i redaka kao u relacijskim bazama podataka, kao NoSQL baza podataka, MongoDB arhitektura sastoji se od zbirki i dokumenata. Dokumenti se sastoje od parova ključ-vrijednost. Zbirke, ekvivalent SQL tablicama, sadrže skupove dokumenata. Na slici 3.2 nalazi se usporedba SQL i NoSQL (na primjeri MongoDB) tipa baze podataka.



Slika 3.2. Prikaz usporedbe SQL-a i MongoDB-a [18]

3.4. Biblioteke

3.4.1. Mongoose

Prema [19] Mongoose je biblioteka ODM (engl. *Object Data Modeling*) za MongoDB koja omogućava strukturiranje i pristup vlastitim podacima na jednostavan način. Razlozi zašto programeri često rade s ovom bibliotekom su: kao pomoć u modeliranju, kod provedbe sheme, validacija modela i kod opće manipulacije podacima.

3.4.2. React Bootstap

Prema [20] React Boostrap je biblioteka u kojem je svaka komponenta izgrađena kao prava React komponenta, bez nepotrebnih ovisnosti kao što je jQuery. React-Bootstrap je kompatibilan s većinom Bootstrap tema. Budući da je rađena na modelu React komponente veća je kontrola nad oblikom i funkcijom svake komponente.

3.4.3. Axios

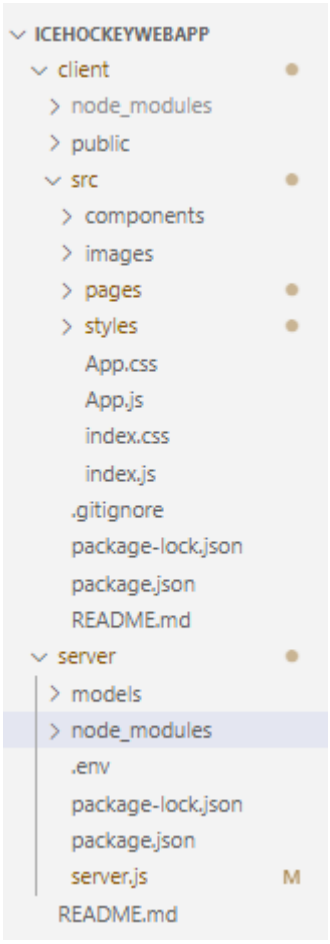
Prema [21] Axios je JavaScript biblioteka koja se obično koristi za izradu HTTP zahtjeva iz web preglednika ili Node.js-a. Pruža jednostavan i intuitivan API za slanje asinkronih HTTP zahtjeva poslužitelju i rukovanje odgovorom.

4. IZRADA WEB APLIKACIJE

Izrada svake MERN aplikacije počinje s postavljanjem klijentske i poslužiteljske strane, kao i uspostava i povezivanje baze podataka s aplikacijom. U potpoglavlju 4.2. opisana je izrada interaktivnog i intuitivnog korisničkog sučelja na klijentskoj strani, dok su u potpoglavlju 4.3. opisani HTTP zahtjevi i povezivanje s bazom podataka. U zadnjem potpoglavlju opisan je sadržaj baze podataka.

4.1. Postavljanje aplikacije

Aplikacija je podijeljena na dva direktorija: *client* i *server*. Slika 4.1. prikazuje sadržaj pojedinog direktorija.



Slika 4.1. Struktura web aplikacije

U *client* direktoriju je instaliran React tako što je u terminalu upisana naredba *npx create-react-app* kao što je prikazano na slici 4.2. *Create-next-app* je naredba u Next.js okviru (engl. *framework*) koja omogućuje brzo i jednostavno postavljanje projekta. Nakon pokretanja naredbe,

potrebno je navesti naziv projekta, nakon čega se generira osnovna struktura Next.js aplikacije, uključujući datoteke i direktorije za komponentne, stranice i stilove. Također se postavlja početna datoteka *package.json* s potrebnim ovisnostima i skriptama.

```
PS C:\Users\karla\Documents\IceHockeyWebApp\client> npx create-react-app.
```

Slika 4.2. Prikaz naredbe za postavljanje React-a

U *server* direktoriju su instalirani Node.js i Express.js, kao što je prikazano na slici 4.3. i slici 4.4. Instalacija Node.js-a je napravljena pomoću naredbe *npm init* u terminalu. Nakon pokretanja, prikazan je niz upita o projektu kao što su: ime projekta, verzija projekta, git repozitorij i sl. Na temelju odgovora generirat će se datoteka *package.json* (različita od one na klijentskoj strani) s navedenim informacijama.

```
PS C:\Users\karla\Documents\IceHockeyWebApp\server> npm init
```

Slika 4.3. Prikaz naredbe za postavljanje Node.js-a

Naredba *npm i express* dohvaća najnoviju verziju Express paketa i instalira ga u direktoriji *node_modules*. Uz to, ažurira datoteku *package.json* dodajući Express kao ovisnost (engl. *dependency*).

```
PS C:\Users\karla\Documents\IceHockeyWebApp\server> npm i express
```

Slika 4.4. Prikaz naredbe za postavljanje Express.js-a

4.2. Klijentska strana

Kao što je prikazano na slici 4.5. u App.js skripti postavljen je način usmjeravanja za različite komponente pomoću biblioteke *react-router-dom*. Svaka <*Route*> komponenta definira put i prikazuje odgovarajuću komponentu kada se pristupi tom putu. Također, unutar funkcije nalazi se varijabla stanja *isAuthenticated* kojom se pomoću tzv. udice (engl. *hook*) *useState* provjerava je li korisnik prijavljen ili nije. Kada se pristupi korijenskom putu, osim što se prikazuje *Home* komponenta, prosljeđuje se i stanje varijable.

```
function App() {
  const [isAuthenticated, setIsAuthenticated] = useState(false);

  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home isAuthenticated={isAuthenticated} setIsAuthenticated={setIsAuthenticated} />} />
        <Route path="/profile" element={<Profile />} />
        <Route path="/team-selection" element={<TeamSelection />} />
        <Route path="/carolina-hurricanes" element={<CarolinaHurricanes />} />
        <Route path="/boston-bruins" element={<BostonBruins />} />
        <Route path="/golden-knights" element={<GoldenKnights />} />
        <Route path="/colorado-avalanche" element={<ColoradoAvalanche />} />
      </Routes>
    </Router>
  );
}

export default App;
```

Slika 4.5. Usmjeravanje komponenti u web aplikaciji

Prevođenje (engl. *rendering*) korijenske komponente i njeno montiranje na DOM napravljeno je prema prikazu na slici 4.6. i ono se nalazi u index.js skripti. Metoda *createRoot* omogućuje stvaranje korijena (engl. *root*) za prikaz React komponenti unutar DOM-a. Prikaz React komponenti obavlja se pomoću metode *root.render*.

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <App/>
);
```

Slika 4.6. Montiranje korijenske komponente na DOM

4.2.1. Registracija i prijava korisnika

Pri pristupu početnoj stranici korisniku se pruža mogućnost prijave i registracije. Ako korisnik želi pristupiti ostalim dijelovima aplikacije mora biti prijavljen. Kod na slici 4.7. obrađuje podnošenje obrasca za registraciju. Pri upisu podataka za registraciju šalje se *post* zahtjev na poslužiteljsku stranu. Nakon toga, dolazi provjera odgovora poslužitelja kako bi se odredio ishod pokušaja registracije. Na temelju odgovora poslužitelja na korisničkom sučelju se prikazuje poruka je li registracija uspješna ili nije.

```
const handleRegistrationSubmit = async (e) => {
  e.preventDefault();

  try {
    const response = await axios.post(
      "http://localhost:5000/registration",
      userRegisterData
    );

    if (
      response.status === 201 &&
      response.data.message === "User registered successfully"
    ) {
      setRegistrationMessage("Registration successful");
    } else if (
      response.status === 200 &&
      response.data.message === "User already exists"
    ) {
      setRegistrationMessage("Username or email already exists");
    } else {
      console.log("Unexpected response:", response.data);
    }
  } catch (error) {
    console.error("Registration failed:", error.response.data.message);
  }
};
```

Slika 4.7. Kod za registraciju korisnika

Vrijednost podataka za registraciju nalazi se u tzv. udici *useState* sa stanjem *userRegisterData* i funkcijom koja postavlja to stanje *setUserRegisterData* kao što se može vidjeti na slici 4.8. Uz to, za rukovanje unosom tih podataka napravljena je funkcija *handleRegisterChange* koja ažurira stanje *userLoginData* kada se promjeni vrijednost u poljima za unos.

```
const [userRegisterData, setRegisterUserData] = useState({
  username: "",
  email: "",
  password: "",
});

const [registrationMessage, setRegistrationMessage] = useState("");

const handleRegisterChange = (e) => {
  const { name, value } = e.target;
  setRegisterUserData({
    ...userRegisterData,
    [name]: value,
  });
};
```

Slika 4.8. Kod za unos podataka za registraciju

Sličan princip ima prijava korisnika, no u slučaju prijave korisnika generira se token za provjeru korisničkih podataka, više o njemu nalazi se u odjeljku 4.3.4. Na klijentskoj strani, kao što se vidi na slici 4.9. ako je prijava uspješno izvršena stanje varijable *isAuthenticated* se stavlja na točno i token za provjeru korisničkih podataka se sprema u lokalno spremište, što je bitno za daljni prolazak kroz aplikaciju. Korisnika se šalje na početnu stranicu, no sada je prijavljen i može pristupiti ostalim dijelovima aplikacije.

```
const handleLoginSubmit = async (e) => {
  e.preventDefault();

  try {
    const response = await axios.post(
      "http://localhost:5000/login",
      userLoginData
    );

    if (response.status === 200 && response.data.token) {
      const { token, userId } = response.data;

      localStorage.setItem("authToken", token);
      localStorage.setItem("userId", userId);

      setIsAuthenticated(true);
      setLoginMessage("Login successful");
      navigate("/");
```

Slika 4.9. Kod za prijavu korisnika

Unos podataka za prijavu i funkcija za rukovanje unosom radi na isti princip kao i za registraciju korisnika, osim što kada korisnik unosi podatke, kao što se vidi na slici 4.10., za prijavu treba upisati samo korisničko ime i zaporku.

```
const [userLoginData, setUserLoginData] = useState({
  username: "",
  password: "",
});

const [loginMessage, setLoginMessage] = useState("");

const handleLoginChange = (e) => {
  const { name, value } = e.target;
  setUserLoginData({
    ...userLoginData,
    [name]: value,
  });
};
```

Slika 4.10. Kod za unos podataka za prijavu

4.2.2. Pristup korisničkom profilu i odjava korisnika

Nakon prijave, korisnik ima mogućnost pregleda prijašnjih pokušaja u svojem profilu, te mogućnost odjave. Odlazak na profil omogućen je pritiskom tipke koja sadrži korisničko ime. Korisničko ime se dohvaća slanjem *fetch* zahtjeva prema poslužiteljskoj strani kao što se vidi na slici 4.11.

```
useEffect(() => {
  const userId = localStorage.getItem("userId");
  fetch(`http://localhost:5000/getUsername?userId=${userId}`)
    .then((response) => response.json())
    .then((data) => {
      setProfileUsername(data.username);
    })
    .catch((error) => {
      console.error("Error fetching username:", error);
    });
}, []);
```

Slika 4.11. Kod za dohvaćanje korisničkog imena

Pregled prijašnjih pokušaja prikazan je na slici 4.12. gdje se generira *get* zahtjev i šalje se na poslužiteljsku stranu. uz jedinstveni identifikator koji korisnik dobije pri registraciji.

```
axios
  .get(
    `http://localhost:5000/getPreviousLineups?userId=${userId}`,
    axiosConfig
  )
  .then((response) => {
    setPreviousLineups(response.data);
    setCurrentIndex(0);
  })
  .catch((error) => {
    console.error("Error fetching previous lineups:", error);
  });
}, []);
```

Slika 4.12. Kod za dohvaćanje prijašnjih pokušaja

Osim što funkcija dohvaća podatke o prijašnjim pokušajima i postavlja ih u varijablu *previousLineups*, postavlja i stanje *currentIndex*, na nulu, što se može vidjeti na slici 4.12. Ovo je, uz funkcije *prevSlide* i *nextSlide* koje se nalaze na slici 4.13., korišteno za kontroliranje prikaza pokušaja. Ove funkcije korisniku omogućuju jednostavno prolaženje kroz listu pokušaja.

```
const prevSlide = () => {
  if (currentIndex > 0) {
    setCurrentIndex(currentIndex - 1);
  }
};

const nextSlide = () => {
  if (currentIndex < previousLineups.length - 1) {
    setCurrentIndex(currentIndex + 1);
  }
};
```

Slika 4.13. Kod funkcije za kontrolu prikaza

Ako se korisnik odluči odjaviti, token za provjeru korisničkih podataka se briše iz lokalnog spremišta i stanje za provjeru korisničkih podataka se postavlja na netočno, što znači da nije moguće provjeriti podatke korisnika i utvrditi je li korisnik važeći. Nakon toga korisnik više neće imati pristup ostalim dijelovima aplikacije. Način rada prikazan je na slici 4.14. Uz to, korisnika se automatski vraća na početnu stranicu, gdje se ako želi opet koristiti aplikaciju mora ponovno prijaviti.

```
const handleLogoutClick = () => {
  localStorage.removeItem("authToken");
  setIsAuthenticated(false);
  navigate("/");
};
```

Slika 4.14. Kod za odjavu korisnika

4.2.3. Provjera korisničkih podataka

Pri samom dolasku na početnu stranicu, korištenjem tzv. udice *useEffect*, prikazanoj na slici 4.15., provjerava se postoji li u lokalnom spremištu token za provjeru korisničkih podataka. Ako postoji važeći token stanje *isAuthenticated* se postavlja na točno i korisniku je omogućen daljnji pristup aplikaciji, ako ga nema, stanje *isAuthenticated* se postavlja na netočno i korisniku je odbijen daljnji pristup.

```
useEffect(() => {
  const authToken = localStorage.getItem("authToken");

  if (authToken) {
    setIsAuthenticated(true);
  } else {
    setIsAuthenticated(false);
  }
}, []);
```

Slika 4.15. Kod za provjeru korisničkih podataka

4.2.4. Pristup odabiru tima

Pristup za odabir tima kontroliran je tipkom, tj. funkcijom *handleButtonClick* koja korisnika šalje na stranicu za odabir tima samo ako je stanje za provjeru korisničkih podataka točno, tj. ako je potvrđeno postojanje tokena za provjeru korisničkih podataka koji se generira pri prijavi korisnika. Na slici 4.16. prikazan je kod za izvršavanje ove funkcije. Ako korisnik klikne tipku, a nije prijavljen tj. nije utvrđeno postojanje tokena za provjeru stanja, spustit će se izbornik za prijavu. Ovime se obavještava korisnika na prijavu ako želi pristupiti odabiru.

```
const handleButtonClick = () => {
  if (isAuthenticated) {
    navigate("/team-selection");
  } else {
    const loginDropdown = document.querySelector(".dropdown-content");
    if (loginDropdown) {
      loginDropdown.style.display = "block";
    }
  }
};
```

Slika 4.16. Kod za kontrolu pristupa daljnjem dijelu aplikacije

4.2.5. Dohvaćanje datuma utakmice

Korisnik za svaki tim ima na izbor 10 datuma utakmica za koje može složiti postavu. Datumi utakmica nalaze se u bazi i ti se podaci dohvaćaju *get* zahtjevom uz parametar *teamName* kao što je prikazano na slici 4.17. Razlog korištenja imena tima kao parametara za dohvaćanje datuma je kako bi se osiguralo dohvaćanje datuma samo izabranog tima jer se u bazi podataka nalazi više timova koji su odigrali utakmice na isti datum. Na slici 4.17. je primjer dohvaćanja datuma za tim Colorado Avalanche. Pristup je jednak za sve timove.

```
const fetchGameDates = async () => {
  try {
    const response = await axios.get("http://localhost:5000/gamedates", {
      params: {
        teamName: "Colorado Avalanche",
      },
    });
    const dates = response.data;
    setGameDates(dates);
    if (dates.length > 0) {
      setSelectedGameDate(dates[0]?.date || "");
    }
  } catch (error) {
    console.log(error);
  }
};
```

Slika 4.17. Kod za dohvaćanje datuma

4.2.6. Dohvaćanje igrača

Nakon odabira datuma, korisniku će se prikazati igrači koji su igrali utakmicu na taj datum. Kod za dohvaćanje igrača za napad nalazi se na slici 4.18. Princip dohvaćanja jednak je kao i za odabir datuma, samo uz parametar *teamName*, postoji i parametar *gameDate* kako bi se osiguralo dohvaćanje samo igrača koji su taj dan igrali. Također, dohvaćanje igrača za obranu radi na isti način.

```
const fetchForwardPlayers = async (selectedDate) => {
  try {
    const response = await axios.get("http://localhost:5000/forwardplayers", {
      params: {
        gameDate: selectedDate,
        teamName: "Colorado Avalanche",
      },
    });
    const players = response.data;
    setForwardPlayers(players);
  } catch (error) {
    console.log(error);
  }
};
```

Slika 4.18. Dohvaćanje igrača za napad

4.2.7. Kontrola prikaza igrača

Na slici 4.19. nalazi se funkcija za kontrolu prikaza igrača ovisno o odabranom datumu. Funkcija ažurira varijablu stanja *selectedGameDate*, te pokreće funkcije dohvaćanja igrača za napad i obranu na temelju odabranog datuma.

```
const handleGameDateChange = (e) => {
  const selectedDate = e.target.value;
  setSelectedGameDate(selectedDate);

  if (selectedDate) {
    fetchForwardPlayers(selectedDate);
    fetchDefensivePlayers(selectedDate);
  } else {
    setForwardPlayers([]);
    setDefensivePlayers([]);
  }
};
```

Slika 4.19. Kontrola prikaza igrača ovisno o datumu

Na slici 4.20. prikazana je funkcija koja kontrolira odabir igrača. Funkcija pravi kopiju prethodnog stanja (prethodnog odabira) i provjerava je li to isti igrač, ako je, briše igrača iz prethodnog odabira i ostavlja ga u trenutnom. Ovime se sprječava korisnikov odabir istog igrača u više polja. Isti princip vrijedi i za obrambene igrače.

```
const handleForwardPlayerChange = (e, index) => {
  const selectedPlayer = e.target.value;
  setSelectedForwardPlayers((prevPlayers) => {
    const updatedPlayers = [...prevPlayers];

    updatedPlayers.forEach((player, i) => {
      if (i !== index && player && player === selectedPlayer) {
        updatedPlayers[i] = "";
      }
    });

    updatedPlayers[index] = selectedPlayer;

    return updatedPlayers;
  });
};
```

Slika 4.20. Kontrola prikaza igrača za napad ovinso o prethodnom izboru

4.2.8. Spremanje strategije

Nakon što korisnik izabere igrače ima mogućnost spremanja izabrane strategije, na način prikazan slikom 4.21. Prilikom spremanja uzima se jedinstveni identifikator iz lokalnog spremišta kako bi bio spremljen u bazu zajedno sa strategijom. Ovime se omogućuje prikaz prijašnjih

pokušaja u korisnikovom profilu. *Post* zahtjevom na poslužiteljsku stranu, strategija se sprema u MongoDB bazu podataka pod kolekciju *lineups*.

```
try {
  await axios.post("http://localhost:5000/saveSelection", data, {
    headers,
  });
  console.log("Selection saved successfully");
  setNotificationMessage("Lineup saved successfully!");
  setTimeout(() => {
    setNotificationMessage("");
  }, 3000);
} catch (error) {
  console.log(error);
  setNotificationMessage("Failed to save lineup. Please try again.");
  setTimeout(() => {
    setNotificationMessage("");
  }, 3000);
}
};
```

Slika 4.21. Kod za spremanje korisnikov izbor strategije

4.2.9. Analiza strategije

Funkcija za analizu korisnikove strategije poziva nekoliko funkcija. Na slici 4.22. prikazan je kod koji se izvršava nakon klika na tipku. Kako bi se dohvatile informacije o postavama koje su odigrale utakmicu za odabrani datum šalje se *get* zahtjev na poslužiteljsku stranu. Zatim se uzimaju podaci o korisnikovom izboru postava i vraćaju imena odabranih igrača. Poslije toga se pozivaju funkcije za analizu i usporedbu, nakon čega se rezultati funkcija spremaju za prikaz u skočnom prozoru koji se otvara pomoću funkcije *openModel*.

```
const handleCheck = async () => {
  try {
    const response = await axios.get("http://localhost:5000/gamelineup", {
      params: {
        gameDate: selectedGameDate,
        teamName: "Colorado Avalanche",
      },
    });
    const lineup = response.data;

    const selectedForwardLineup = selectedForwardPlayers
      .filter((playerId) => playerId)
      .map((playerId) => {
        const player = forwardPlayers.find((p) => p._id === playerId);
        return player ? player.playerName : "";
      });

    const selectedDefensiveLineup = selectedDefensivePlayers
      .filter((playerId) => playerId)
      .map((playerId) => {
        const player = defensivePlayers.find((p) => p._id === playerId);
        return player ? player.playerName : "";
      });
```

Slika 4.22. Kod za provjeru uspješnosti izbora

Funkcija na slici 4.23. *findMostStatsPlayers* iz korisnikovog odabira igrača pronalazi tri vrste podataka: igrača koji je zabio najviše golova, igrača koji je imao najviše asistencija i igrača koji je najviše minuta proveo van igre zbog kazne. Ova funkcija daje korisniku bolji uvid u statistiku igrača koje je odabrao.

```
const findMostStatsPlayers = (selectedForwardPlayers, selectedDefensivePlayers,
  forwardPlayers,defensivePlayers
) => {
  for (const playerId of selectedForwardPlayers) {
    if (!playerId) continue;
    const player = forwardPlayers.find((p) => p._id === playerId);
    if (!player) continue;
    if (player.goals > mostGoals) {
      mostGoals = player.goals;
      mostGoalsPlayer = player.playerName;
    }
    if (player.assists > mostAssists) {
      mostAssists = player.assists;
      mostAssistsPlayer = player.playerName;
    }
    if (player.PIM > mostPenaltyMinutes) {
      mostPenaltyMinutes = player.PIM;
      mostPenaltyMinutesPlayer = player.playerName;
    }
  }
```

Slika 4.23. Kod za pronalaženje igrača sa najvišim vrijednostima

Funkcija na slici 4.24. *checkLineupMatch* obavlja analizu postave i daje savjete o odabiru prikladnijeg igrača, tj. igrača koji bi zabio više golova ili više pridonio igri svojim asistencijama. Postava se analizira povlačenjem podataka iz baze koji sadrže informacije o odigranim utakmicama, usporedbom tih podataka s korisnikovim unosom, te slanjem povratnih informacija. Postoje tri vrste postava u bazi podataka, a to su: najbolja postava, osrednja postava, te najgora postava. U kodu prikazanog na slici 4.25. uspoređuju se korisnikova napadačka postava i postave koje su odigrale utakmicu. Postave koje su odigrale utakmicu su povučene iz baze podataka. Ako se korisnikova postava podudara s bilo kojom od onih iz baze, korisniku se prikazuje koji je tip postave pogodio. Isti princip je i za obrambene igrače.

```
const checkLineupMatch = (
  lineup,
  selectedForwardLineup,
  selectedDefensiveLineup
) => {

  for (const item of lineup) {
    if (
      item.position === "forward" &&
      isSameLineup(selectedForwardLineup, item.lineup)
    ) {
      forwardMatch = true;
      forwardMatchType = item.type;
    }
  }
```

Slika 4.24. Kod funkcije za analizu postave

```
  if (forwardMatch && defensiveMatch) {
    setLineupMatchup(
      `Forward lineup matches ${forwardMatchType}. Defensive lineup matches ${defensiveMatchType}.`
    );
  } else if (forwardMatch) {
    setLineupMatchup(`Forward lineup matches ${forwardMatchType}.`);
  } else if (defensiveMatch) {
    setLineupMatchup(`Defensive lineup matches ${defensiveMatchType}.`);
  } else {
    setLineupMatchup("");
  }
};
```

Slika 4.25. Kod za ispis rezultata usporedbe

Na slici 4.26. je prikazana funkcija koja obavlja usporedbu. Funkcija provjerava jesu li dva polja jednaka, tj. jesu li igrači unutar postave isti. Ako postoji podudaranje za svako polje u sastavu, funkcija vraća točno, u suprotnome netočno.

```
const isSameLineup = (lineup1, lineup2) => {
  if (
    !Array.isArray(lineup1) ||
    !Array.isArray(lineup2) ||
    lineup1.length !== lineup2.length
  ) {
    return false;
  }

  for (const player of lineup1) {
    const normalizedPlayer = normalize(player);
    if (!lineup2.some((player2) => normalize(player2) === normalizedPlayer)) {
      return false;
    }
  }

  for (const player of lineup2) {
    const normalizedPlayer = normalize(player);
    if (!lineup1.some((player1) => normalize(player1) === normalizedPlayer)) {
      return false;
    }
  }

  return true;
};
```

Slika 4.26. Kod za usporedbu postava

Uz to pri usporedbi funkcija *normalize* na slici 4.27. miče razmak između riječi i postavlja sve na mala tiskana slova. Ovo se radi kao bi se osiguralo točno podudaranje i ispravna usporedba postave.

```
const normalize = (player) => player.replace(/\s/g, "").toLowerCase();
```

Slika 4.27. Kod za funkciju *normalize*

Uz tu usporedbu, korisniku se pružaju i savjeti za unapređenje odabrane postave. Slika 4.28. prikazuje kod za provjeru je li korisnikova odabrana postava jednaka ili nije s jednom vrstom postave iz baze. Provjera se radi samo kada korisnik nije pogodio sva tri igrača bilo koje vrste postave. Ako nije pogodio sva tri igrača provjerava se je li pogodio jednog ili dva igrača iz jedne od postava iz baze. Korisniku se također daje savjet koji bi igrači upotpunili postavu za bolju izvedbu.

```
if (!isGoodLineup && !isBestLineup && !isWorstLineup) {
  missingBestPlayers = bestForwardPlayersNormalized.filter(
    (player) => !normalizedSelectedForwardLineup.includes(player)
  );
  missingGoodPlayers = goodForwardPlayersNormalized.filter(
    (player) => !normalizedSelectedForwardLineup.includes(player)
  );
  missingWorstPlayers = worstForwardPlayersNormalized.filter(
    (player) => !normalizedSelectedForwardLineup.includes(player)
  );
  if (missingBestPlayers.length === 2) {
    advice += `You have one player from the best forward lineup. Consider adding: ${missingBestPlayers
      .map((player) => displayOriginalName(player, bestForwardPlayers))
      .join(", ")}. `;
  } else if (missingBestPlayers.length === 1) {
    advice += `You have two players from the best forward lineup. Consider adding: ${missingBestPlayers
      .map((player) => displayOriginalName(player, bestForwardPlayers))
      .join(", ")}.  `;
  } else if (missingGoodPlayers.length === 2) {
    advice += `You have one player from the good forward lineup. Consider adding: ${missingGoodPlayers
      .map((player) => displayOriginalName(player, goodForwardPlayers))
      .join(", ")}. `;
  } else if (missingGoodPlayers.length === 1) {
    advice += `You have two players from the good forward lineup. Consider adding: ${missingGoodPlayers
      .map((player) => displayOriginalName(player, goodForwardPlayers))
      .join(", ")}. `;
  } else if (missingWorstPlayers.length > 0) {
    advice += `You also have players from the worst forward lineup. Consider choosing from the best lineup: ${bestForwardPlayers.join(
      ", "
    )}. `;
  }
}
if (isWorstLineup) {
  advice += `You have players from the worst forward lineup. Consider choosing from the best lineup: ${bestForwardPlayers.join(
    ", "
  )}.  `;
}
```

Slika 4.28. Kod za davanje savjeta

U dijelu koda za davanje savjeta nalazi se nekoliko funkcija za provjeru i dohvaćanje podataka. Prvi primjer funkcije je prikazan na slici 4.29. pod nazivom *getBestForwardPlayers* koja pronalazi igrače koji pripadaju najboljoj postavi. Isti princip vrijedi i za osrednju i najgoru postavu, kao i za obrambene igrače.

```
const getBestForwardPlayers = (lineup) => {
  const bestForwardLineup = lineup.find(
    (item) => item.position === "forward" && item.type === "best lineup"
  );
  return bestForwardLineup ? bestForwardLineup.lineup : [];
};
```

Slika 4.29. Kod za dohvaćanje igrača najbolje postave

Drugi primjer funkcije je *displayOriginalName* prikazana na slici 4.30. čija je svrha vratiti naziv igrača u originalno stanje jer se za davanje savjeta korisniku ispisuju igrači koje nije odabrao. Ova funkcija je korištena kako bi se korisniku pružilo bolje korisničko iskustvo budući da *normalize* funkcija, koja je korištena za ispravan pronalazak imena, promjeni izgled imena i prezimena.

```
const displayOriginalName = (normalizedName, originalNamesArray) => {
  const originalName = originalNamesArray.find(
    (name) => normalize(name) === normalizedName
  );
  return originalName || normalizedName;
};
```

Slika 4.30. Kod vraćanje originalnog imena

4.2.10. Prikaz analize strategije

Klikom na tipku *check* otvara se skočni prozor u kojem su prikazani rezultati analize. Skočni prozor je napravljen pomoću komponente *react-modal*. Sama funkcija prima parametar *isOpen* koji služi za otvaranje i zatvaranje skočnog prozora. Ostali parametri su stanja varijabli koje sadrže rezultate funkcija za analizu unosa korisnikove strategije. Kod je prikazan na slici 4.31.

```jsx
const ResultModal = ({ isOpen, closeModal,lineupMatchup,mostGoalsPlayer, mostAssistsPlayer,mostPenaltyMinutesPlayer,advice }) => {
  return (
    <Modal isOpen={isOpen} onRequestClose={closeModal} className="modal">
      <div className="modal-content">
        <h2>Lineup Match Result</h2>
        {lineupMatchup&& <p>Lineup Matchup: {lineupMatchup}</p>}
        {advice &&<p>Advice: {advice}</p>}
        {mostGoalsPlayer&&<p>Player with the most goals: {mostGoalsPlayer}</p>}
        {mostAssistsPlayer &&<p>Player with the most assists: {mostAssistsPlayer}</p>}
        {mostPenaltyMinutesPlayer &&<p>Player with the most penaly minutes: {mostPenaltyMinutesPlayer}</p>}
        <button onClick={closeModal}>Close</button>
      </div>
    </Modal>
  );
};
```

Slika 4.31. Kod za prikaz skočnog prozora

4.3. Poslužiteljska strana

Na poslužiteljskoj strani nalazi se server.js skripta u kojoj se postavlja express poslužitelj, uspostavlja se veza s MongoDB bazom podataka i definira nekoliko ruta za dohvaćanje i spremanje podataka.

4.3.1. Uspostavljanje veza

Na slici 4.32. prikazano je uspostavljanje veze s bazom podataka pomoću biblioteke Mongoose. Za povezivanje korištena je ugrađena funkcija *connect* s parametrom *uri* (engl. *Uniform Resource Identifier*) i dvjema opcijama *useNewUrlParser* i *useUnifiedTopology* koje su korištene za konfiguraciju veze. Funkcija *app.listen* se koristi za povezivanje i slušanje (engl. *listen*) veza na navedenoj pristupnoj točki (engl. *port*). Ako je veza uspješno uspostavljena poslužitelj je spreman za obradu dolaznih HTTP zahtjeva.

```jsx
mongoose.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => {
    console.log('Connected to MongoDB Atlas');
    app.listen(port, () => {
      console.log(`Example app listening on port ${port}`);
    });
  })
  .catch((error) => {
    console.error('Error connecting to MongoDB Atlas:', error);
  });
```

Slika 4.32. Kod za povezivanje sa bazom podataka

4.3.2. Post zahtjevi

Jedan od razloga korištenja *post* zahtjeva je za slanje podataka na poslužitelj za stvaranje ili ažuriranje resursa (engl. *resource*).

Primjer rukovanja *post* zahtjevom objašnjen je na spremanju korisnikovog izbora postave, iako se u aplikaciji još nalaze *post* zahtjevi za spremanje registriranog korisnika u bazu i prijava korisnika.

Za spremanje korisnikove postave je potreban model prikazan slikom 4.33. Model sadrži polja koja trebaju odgovarati strukturi dokumenata u kolekciji lineup.

```
const lineupSchema = new mongoose.Schema({
  teamName: {
    type: String,
    required: true,
  },
  gameDate: {
    type: String,
    required: true,
  },
  forwardLineup: {
    type: [String],
    required: true,
    validate: {
      validator: (value) => value.length === 3,
      message: 'Forward lineup must have exactly 3 players',
    },
  },
  defensiveLineup: {
    type: [String],
    required: true,
    validate: {
      validator: (value) => value.length === 2,
      message: 'Defensive lineup must have exactly 2 players',
    },
  },
  userId: {
    type: String,
    required: true,
  },
});

const Lineup = mongoose.model('Lineup', lineupSchema);
```

Slika 4.33. Model za spremanje korisnikove postave

Kod koji definira rutu za rukovanje *post* zahtjevom prikazan je na slici 4.34. Funkcija *app.post* postavlja rutu za rukovanje *post* zahtjevom na krajnjoj točki */SaveSelection.* Kod unutar funkcije će se izvršiti kada se s klijentske strane pošalje *post* zahtjev prema krajnjoj točki.

Funkcija *verifiyToken* osigurava spremanje podataka korisnika čiji su podaci provjereni i važeći, kao i blokiranje spremanja podataka korisnika čiji podaci nisu važeći. Model postave koji se nalazi na slici 4.34. koristi se za stvaranje novog dokumenta u bazi podataka. Objekt *lineupData* se koristi za popunjavanje dokumenta.

```
app.post('/saveSelection', verifyToken, async (req, res) => {
  try {
    const userId = req.userId;
    console.log('User ID:', userId);

    console.log(req.body);

    const lineupData = {
      ...req.body,
      userId: userId,
    };

    const lineup = await Lineup.create(lineupData);
    console.log(lineup);

    res.status(200).json(lineup);
  } catch (error) {
    console.log(error.message);
    res.status(500).json({ message: error.message });
  }
});
```

Slika 4.34. Kod za POST zahtjev

4.3.3. Get zahtjevi

Jedan od razloga korištenja *get* zahtjeva je za dohvaćanje podataka iz određenog resursa (engl. *resource*).

Primjer rukovanja *get* zahtjevom objašnjen je na zahtjevu za dohvaćanje datuma utakmica, iako se u aplikaciji nalaze *get* zahtjevi za: dohvaćanje korisničkog imena, dohvaćanje igrača za napad, dohvaćanje igrača za obranu, dohvaćanje već postojećih sastava za usporedbu i dohvaćanje prijašnjih pokušaja.

Za dohvaćanje datuma utakmica također je potreban model koji je prikazan na slici 4.35.

```
const gamedateSchema=mongoose.Schema(
    {
        date:{
            type: String,
            required:[true]
        },
    }
)
```

Slika 4.35. Model za datume utakmica

U ovom slučaju obrađuje se *get* zahtjev prema krajnjoj točki */gamedates*. Dohvaćaju se datumi utakmica iz kolekcije *gamedates* na temelju navedenog parametra *teamName* i šalju se kao JSON podaci u odgovoru. Razlog postojanja upita u zahtjevu je za uzimanje datuma specifičnih za taj hokejaški tim. Ovaj primjer prikazan je na slici 4.36.

```
app.get('/gamedates', (req, res) => {
  const { teamName } = req.query;
  const query = teamName ? { teamName } : {};
  Gamedate.find(query)
    .then((dates) => {
      res.json(dates);
    })
    .catch((error) => {
      console.error('Error fetching game dates:', error);
      res.status(500).json({ error: 'Failed to fetch game dates' });
    });
});
```

Slika 4.36. Kod za *get* zahtjev

4.3.4. Token za provjeru korisničkih podataka

Provjera korisničkih podataka se koristi za provjeru identiteta korisnika prije odobravanja pristupa određenim resursima. U ovoj aplikaciji je potrebna provjera kako bi se kontrolirao pristup odabira tima, a samim time i odabiranje igrača za postavu što je glavna funkcionalnost aplikacije.

U *post* zahtjevu za prijavu nalazi se ugrađena funkcija za generiranje tokena iz biblioteke JSON Web Token. Prikaz funkcije nalazi se na slici 4.37. Ova funkcija sadrži i vremensko ograničenje za trajanje tokena. Kada token istekne, više neće moći biti korišten za provjeru.

```
const token = jwt.sign({ userId: user._id }, secretKey, {
  expiresIn: '1h',
});
```

Slika 4.37. Metoda za generiranje tokena

Na slici 4.38. je funkcija za provjeru korisničkih podataka na temelju JWT tokena. Funkcija provjerava prisutnost važećeg tokena i izdvaja korisnički ID za daljnju obradu, dopuštajući ovlaštenim korisnicima pristup zaštićenim rutama.

```
const verifyToken = (req, res, next) => {
  const token = req.headers.authorization && req.headers.authorization.split(' ')[1];

  if (!token) {
    return res.status(401).json({ message: 'Unauthorized' });
  }

  jwt.verify(token, secretKey, (err, decoded) => {
    if (err) {
      return res.status(401).json({ message: 'Invalid token' });
    }
    req.userId = decoded.userId;
    console.log('Token verification successful. User ID:', req.userId);
    next();
  });
};
```

Slika 4.38. Metoda za provjeru kosrisničkih podataka

4.4. Baza podataka

Baza podataka se sastoji od 6 kolekcija: defensiveplayers, forwardplayers, gamedates, gamelineups, lineups i users.

U kolekciji defensiveplayers nalaze se sljedeći podaci: ID (zaseban za svakog igrača), naziv tima, ime igrača, pozicija igrača, broj postignutih golova, broj postignutih asistencija, broj udaraca na gol, datum utakmice, te kaznena minutaža. Prikaz podataka igrača u kolekciji se nalazi na slici 4.39. Kolekcija forwardplayers sadrži jednake podatke.

```
_id: ObjectId('64959a4f01574758537c1cf3')
teamName: "Colorado Avalanche"
playerName: "Devon Toews"
playerPosition: "D"
goals: 1
assists: 1
gameDate: "14-04-2023"
PIM: 0
shots: 3
```

Slika 4.39. Primjer dokumenta kolekcije defensiveplayers

Na slici 4.40. prikazana je kolekcija gamedates koja sadrži ID, naziv tima i datum utakmice.

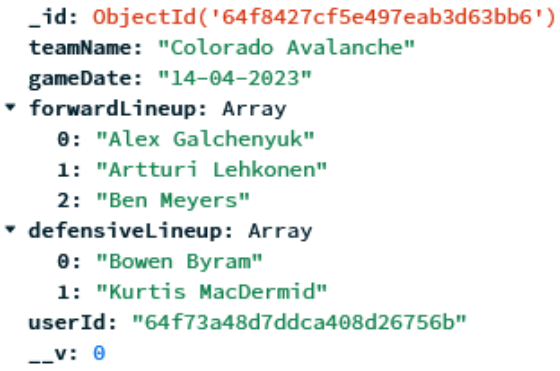
```
_id: ObjectId('6495f2a0151e51c41df1a072')
teamName: "Colorado Avalanche"
gameDate: "14-04-2023"
```

Slika 4.40. Primjer dokumenta kolekcije gamedates

Kolekcija gamelineups, kao što je prikazano na slici 4.41. sadrži podatke o postavama igrača iz utakmice s podacima: ID, datum utakmice, imena igrača u postavi, polje CF% (engl. *Corsi for percentage*) koje predstavlja postotak udaraca prema golu, polje GA (engl. *Goals against*) koje predstavlja ukupan broj golova koji je tim primio, polje GF (engl. *Goals for*) koje predstavlja ukupan broj golova koji je tim zabio, naziv tima, tip postave i pozicija.

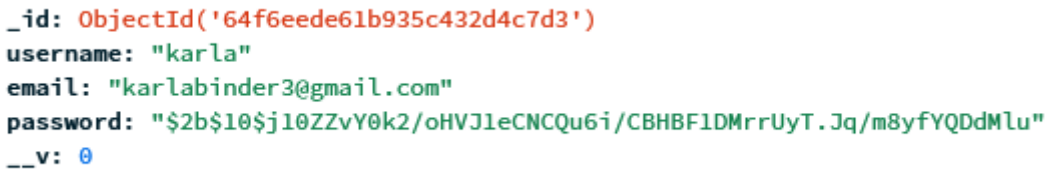
Slika 4.41. Primjer dokumenta kolekcije gamelineups

Kolekcija lineups sadrži podatke o spremljenim korisnikovim izborom sastava s podacima: ID, naziv tima, datum utakmice, imena igrača u postavi za napad, imena igrača u postavi za obranu, korisnički ID. Primjer dokumenta je na slici 4.42.



Slika 4.42. Primjer dokumenta kolekcije lineups

U zadnjoj kolekciji pod nazivom users nalaze se podaci o registriranim korisnicima s podacima: korisničko ime, e-mail i zaporka. Primjer se nalazi na slici 4.43.



Slika 4.43. Primjer dokumenta kolekcije users

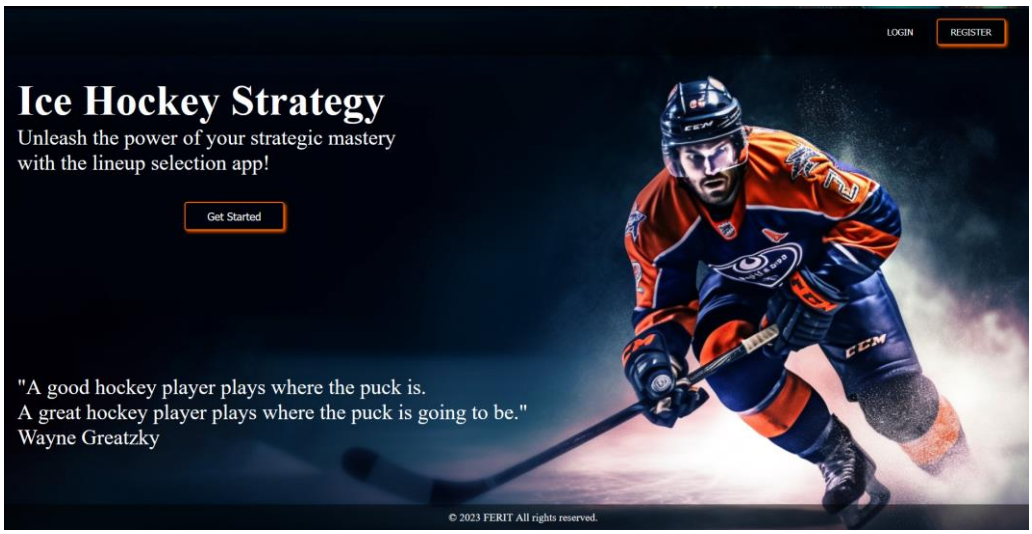
Podaci o postavama igrača i njihovim rezultatima za usporedbu prikupljeni su s web stranice Naturalstattrick [22].

5. NAČIN KORIŠTENJA WEB APLIKACIJE

U narednom poglavlju opisano je korištenje web aplikacije i prikazan je njezin izgled.

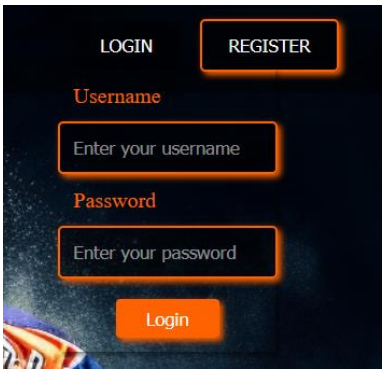
5.1. Početna stranica

Pri dolasku na početnu stranicu prikazanu na slici 5.1. korisniku se na lijevoj strani stranice prikazuje tipka *Get Started* čijim klikom prelazi na stranicu za odabir tima. Na desnoj strani zaglavlja nalaze se tipke za prijavu i registraciju.



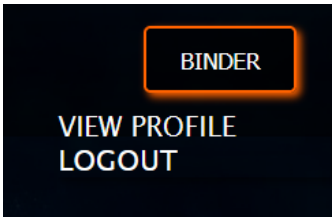
Slika 5.1. Početna stranica aplikacije

Klikom tipke za prijavu ili tipke za registraciju prikazuje se padajući izbornik. Unutar padajućeg izbornika za prijavu kao što je prikazano na slici 5.2. nalaze se polja za unos korisničkog imena i zaporke, Klikom na tipku *Login*, ako su podaci točni korisnik se uspješno prijavljuje. Ako podaci nisu jednaki onima u bazi podataka, korisniku se prikazuje tekst ispod tipke *Login* da prijava nije uspješna.



Slika 5.2. Padajući izbornik za prijavu

Ako je prijava uspješna zaglavlje početne stranice se mijenja i umjesto tipki za prijavu i registraciju, pojavljuje se tipka s korisničkim imenom, kao na slici 5.3. Klikom tipke s korisničkim imenom se također prikazuje padajući izbornik gdje korisnik može birati želi li posjetiti svoj profil u kojem se nalaze prijašnje spremljeni pokušaji klikom na *View profile* ili se može odjaviti klikom na *Logout*.



Slika 5.3. Padajući izbornik za prikaz profila ili odjavu

Tipka *Get Started* uvodi korisnika u stranicu za biranje tima, no samo pod uvjetom da je korisnik prijavljen. Ako korisnik nije prijavljen i klikne na tipku *Get Started*, neće dobiti pristup stranici za biranje tima i spustit čemu se izbornik za prijavu, navodeći ga da je za nastavak potrebna prijava.

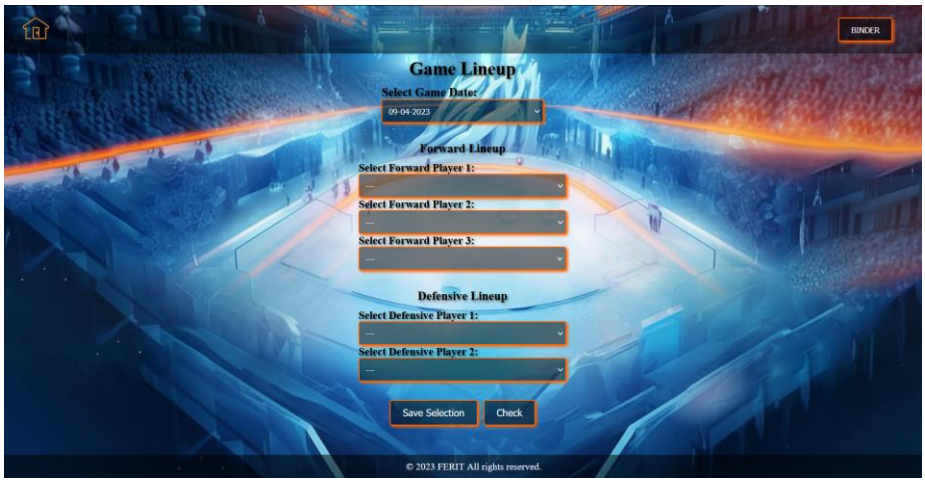
5.2. Stranica za odabir hokejaškog tima

Nakon uspješne prijave i pritiskom tipke *Get Started* korisnik prelazi na stranicu za odabir hokejaškog tima. Korisnik ima mogućnost odabira tima klikom na sliku igrača koji predstavlja tim. Ako korisnik ne može prepoznati tim preko slike, prelaskom mišem preko slike prikazuje se naziv tima. Na stranici se također nalazi ikona kućice, a klikom na nju korisnika se vraća na početnu stranicu. Uz to, nalazi se i tipka s korisničkim imenom. Izgled stranice za odabir hokejaškog tima nalazi se na slici 5.4.

Slika 5.4. Stranica za odabir hokejaškog tima

5.3. Stranica za odabir postave igrača

Klikom na sliku igrača prelazi se na stranicu za odabir postave igrača. Na stranici korisniku je omogućen odabir datuma utakmice za koji želi složiti postavu, nakon čega mu je omogućen izbor igrača napada i igrača obrane koji su bili prisutni na utakmici toga datuma. Prikaz stranice nalazi se na slici 5.5. Nakon odabira igrača korisniku se pružaju dva izbora, pritiskom tipke *Save Selection* sprema se odabrana postava u bazu podataka, dok klikom na tipku *Check* provjerava se uspješnost odabira najkvalitetnije postave.

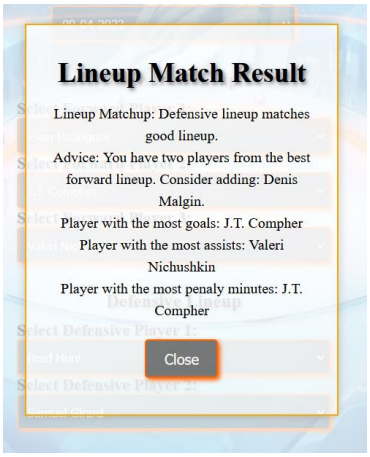


Slika 5.5. Stranica za odabir postave igrača

5.4. Skočni prozor analize postave

Ako korisnik odluči provjeriti kvalitetu vlastite postave, klikom na tipku *Check*, prikazuje se skočni prozor s analizom. Primjer analize u skočnom prozoru nalazi se na slici 5.6. Podudaranje

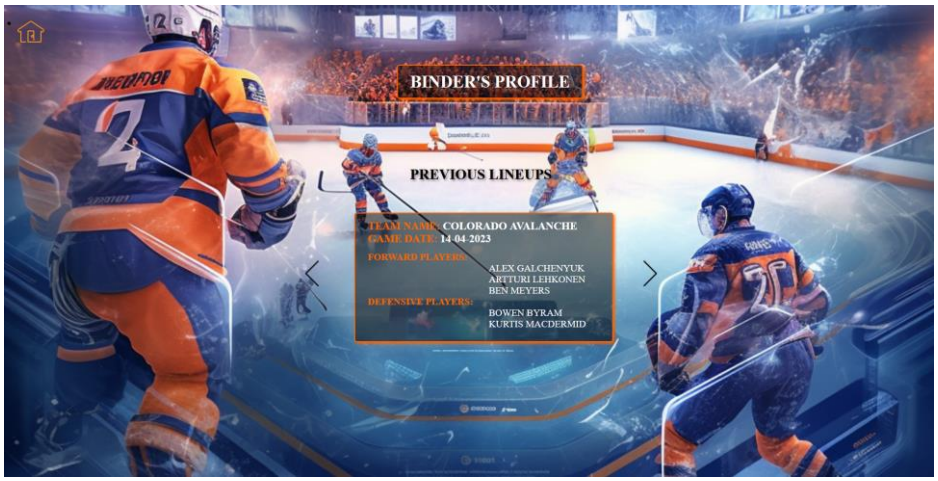
korisnikove postave s jednim tipom postave iz baze prikazuje se pod *Lineup Matchup*. Nakon toga prikazani su savjeti za dodavanje kvalitetnijih igrača u postavu. Zadnje je prikazan uvid u statistike najboljih igrača koje je korisnik izabrao.



Slika 5.6. Skočni prozor analize postave

5.5. Stranica za pregled spremljenih prijašnjih pokušaja

Uvid u spremljene prijašnje pokušaje nalazi se u profilu korisnika. Korisniku je omogućen pristup profilu klikom na *View profile*. Prolazak kroz listu pokušaja omogućen je klikom na lijevu ili desnu strelicu. Stranica je prikazana na slici 5.7.



Slika 5.7. Stranica za pregled spremljenih prijašnjih pokušaja

6. ZAKLJUČAK

Hokej na ledu je sport koji je vrlo poznat u državama kao što su Sjedinjene Američke Države, Kanada i Skandinavske države. Međutim, na područjima kao što su Hrvatska i okolne države hokej na ledu nije tako široko praćen i popularan. Iz toga razloga kreirana je ova aplikacija čija je svrha na zabavan i interaktivan način upoznati korisnike s timovima i igračima najveće hokejaške lige na svijetu.

Osim pružanja informacija o postavama i igračima, cilj je omogućiti prostor za razvitak vlastitih strategija. Hokej na ledu je dinamičan sport s velikim brojem taktičkih elemenata stoga je idealan za analiziranje i razvijanje strategija.

Ovaj rad također je primjer razvoja web aplikacije kolekcijom tehnologija poznatom pod akronimom MERN. Korištenjem ovih tehnologija omogućeno je kreiranje interaktivne i funkcionalne web aplikacije.

Aplikaciju je moguće poboljšati dodavanjem novih timova i datuma utakmica, kao i detaljnija analiza postava kao npr. analiza izvedbe postave u usporedbi sa suparničkim timom. Još jedno moguće rješenje je mogućnost pretpostavke najboljeg sastava prije utakmice, te analiza toga sastava u stvarnom vremenu.

LITERATURA

[1] Setting Your Lineup [online], ESPN, dostupno na: https://support.espn.com/hc/en-us/articles/360000070892-Setting-Your-Lineup- [20.6.2023.]

[2] Out of the Park Developments, Franchise Hockey Manager 5 [online], STEAM, dostupno na:

https://store.steampowered.com/app/880380/Franchise_Hockey_Manager_5/ [20.6.2023.]

[3] CBS Interactive, Franchise Hockey 2022 [online], Google Play, dostupno na:

https://play.google.com/store/apps/details?id=com.atomic.moguls.hockey2018&hl=en [20.6.2023.]

[4] M. Forget, Hockey Lineup Manager [online], Google Play, dostupno na:

https://play.google.com/store/apps/details?id=com.teamsportlineuphockey&hl=en_US [14.9.2023.]

[5] EJ App Development, Hockey Coach Lineup and Roster [online], Google Play, dostupno na:

https://play.google.com/store/apps/details?id=com.ejappdev.hockey_coach_assistant&hl=en&gl=US [14.9.2023.]

[6] Learn to code with Visual Studio Code [online], Visual Studio Code, dostupno na: https://code.visualstudio.com/learn [1.6.2023.]

[7] B. Lutkevich, HTML (Hypertext Markup Language) [online], TheServerSide, 2023., dostupno na: https://www.theserverside.com/definition/HTML-Hypertext-Markup-Language [1.6.2023.]

[8] C. McKenzie, CSS (cascading style sheets) [online], TheServerSide, 2021., dostupno na: https://www.theserverside.com/definition/cascading-style-sheet-CSS [1.6.2023.]

[9] C. McKenzie, JavaScript [online], TheServerSide, 2018., dostupno na: https://www.theserverside.com/definition/JavaScript [1.6.2023.]

[10]A. S. Gillis, JSON (JavaScript Object Notation) [online], TheServerSide, 2022. dostupno na: https://www.theserverside.com/definition/JSON-Javascript-Object-Notation [4.6.2023.]

[11] MERN Stack Explained [online], MongoDB, dostupno na: https://www.mongodb.com/mern-stack [5.6.2023.]

[12] What Does a Front-End Developer Do? Complete Guide to the Front-End Developer Profession [online], Frontend Masters, dostupno na: https://frontendmasters.com/guides/front-end-handbook/2018/what-is-a-FD.html [5.6.2023.]

[13] React [online],React,dostupno na: https://legacy.reactjs.org/ [5.6.2023.]

[14] C. Deshpande, What is Backend Development? Skills, Salary, Roles & More [online], simplilearn, 2023., dostupno na: https://www.simplilearn.com/tutorials/programming-tutorial/what-is-backend-development [5.6.2023.]

[15] Introduction to Node.js [online], nodejs, dostupno na: https://nodejs.dev/en/learn/ [6.6.2023.]

[16] Express.js [online], geeksforgeeks, dostupno na: https://www.geeksforgeeks.org/express-js/ [20.9.2023.]

[17] A. Gillis, B. Botelho, MongoDB [online], TechTarget, 2023., dostupno na: https://www.techtarget.com/searchdatamanagement/definition/MongoDB [7.6.2023.]

[18] Lesson 4:Comparing MongoDB vs SQL Conceepts [online], Studio 3T, dostupno na:

https://studio3t.com/academy/topic/mongodb-vs-sql-concepts/ [7.6.2023.]

[19] J. Hall, Getting Started with MongoDB & Mongoose [online], MongoDB, 2022., dostupno na: https://www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-and-mongoose/ [12.6.2023.]

[20] React Bootstrap [online], React Bootstrap, dostupno na: https://react-bootstrap.netlify.app/ [12.6.2023.]

[21] What is Axios? [online], dostupno na: https://axios-http.com/docs/intro [12.6.2023.]

[22] NaturalStatTrick [online], NaturalStatTrick, dostupno na: https://www.naturalstattrick.com/

[20.6.2023.]

SAŽETAK

Ovaj rad prikazuje način izrade web aplikacije koja korisniku pruža unos postave za hokejašku utakmicu za odabrani hokejaški tim. Glavna funkcionalnost aplikacije je pružanje analize korisnikove postave i pružanje savjeta u vezi poboljšanja odabrane postave. Aplikacija je rađena u Visual Studio Code, gdje je za klijentski dio korišten React, za poslužiteljski dio Node.js i Express js. i za bazu podataka je korišten MongoDB. Jezici korišteni za izradu aplikacije su: HTML, CSS i JavaScript.

Ključne riječi: analiza strategije, hokej na ledu, javascript, MERN, sastavljanje postave

ABSTRACT

Application For Strategic Planning and Analysis of Hockey Matches

This paper presents how to create a web application that allows the user to input a lineup for a hockey game for the selected hockey team. The main functionality of the application is to provide an analysis of the user lineup, as well as provide advise on improving the selected lineup. The application was created in Visual Studio Code, where React was used for the client part, Node.js and Express.js were used for the server part and MongoDB was used for a database. Languages used to create the applications are HTML, CSS and JavaScript.

Keywords: ice hockey, javascript, lineup setup, MERN, strategy analysis

ŽIVOTOPIS

Karla Binder, rođena je 4.7.2023. u Osijeku. Početak obrazovanja započela je u osnovnoj školi Ivana Filipovića u Osijeku. Svoje srednjoškolsko obrazovanje stekla je u II. gimnaziji Osijek, koja je još poznata pod nazivom jezična gimnazija. Trenutno studira na trećoj godini preddiplomskog sveučilišnog studija računarstva, izborni blok računalno inženjerstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek.