Sustav za praćenje medicinskih podataka profesionalnih sportaša

Frankić, Marin

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Preddiplomski sveučilišni studij računarstva

Završni rad

Sustav za praćenje medicinskih podataka profesionalnih sportaša

Rijeka, rujan 2022.

Marin Frankić
0069082674

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Preddiplomski sveučilišni studij računarstva

Završni rad

Sustav za praćenje medicinskih podataka profesionalnih sportaša

Mentor: prof. dr. sc. Ivan Štajduhar

Rijeka, rujan 2022.

Marin Frankić
0069082674

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 8. ožujka 2021.

Zavod: **Zavod za računarstvo**
Predmet: **Uvod u umjetnu inteligenciju**
Grana: **2.09.06 programsko inženjerstvo**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Marin Frankić (0069082674)**
Studij: Preddiplomski sveučilišni studij računarstva

Zadatak: **Sustav za praćenje medicinskih podataka profesionalnih sportaša / Monitoring system for tracking professional athletes' medical data**

Opis zadatka:

Zadatak je izraditi web aplikaciju koja će omogućiti liječniku jednostavan unos informacija dobivenih kod izrade standardiziranih medicinskih testova. Aplikacija se treba sastojati od poslužiteljskog i klijentskog dijela. Pristup aplikaciji moguć je samo za ovlaštene korisnike. Sučelje mora biti sastavljeno na način da je korisniku lako i jednostavno dobiti informacije o pregledima te na temelju istih izvući potrebne informacije. Omogućiti pretraživanje i filtriranje prema često korištenim parametrima. Izraditi testove za ključne funkcionalnosti sustava te time osigurati kvalitetu.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

M. Frankić

Zadatak uručen pristupniku: 15. ožujka 2021.

Mentor:

Izv. prof. dr. sc. Ivan Štajduhar

Predsjednik povjerenstva za završni ispit:

Izv. prof. dr. sc. Kristijan Lenac

Izjava o samostalnoj izradi rada

Izjavljujem da sam završni rad pod naslovom „Sustav za praćenje medicinskih podataka profesionalnih sportaša“ izradio samostalno.

Rijeka, rujan 2022.

Marin Frankić

Zahvala

Zahvaljujem svim prijateljima, kolegama, mentoru i obitelj na podršci tijekom studija i izrade ovog rada.

Sadržaj

1 Uvod

Ovaj završni rad će se fokusirati na razvoj web sustava za praćenje medicinskih podataka profesionalnih sportaša. Nakon što je korisnik kreiran kroz administratorsko sučelje sustava, može započeti proces unosa podataka. Budući da ne postoji proces registracije korisnika, administrator ima punu kontrolu nad osobama koje mogu vidjeti podatke o sportašu.

Korisnik će se moći prijaviti u sustav nakon što primi svoje pristupne podatke. Moći će pristupiti popisu svih sportaša unutar sustava, kao i podacima o svakom pojedinačnom sportašu, informacijama vezanim uz prethodne liječničke preglede i statistikama. Korisnik zatim ima mogućnost unosa novih informacija, promjene dijela informacija i brisanja informacija.

Ovakav sustav, kada bi se razradio u potpunosti, uvelike bi pomogao sportašima, klubovima i klupskim liječnicima u pronalaženju uzroka nekih bolesti i povreda te samim time i bolji uvid u načine zaštite od vanjskih faktora.

2 Tehnologije izrade programskog rješenja

U ovom poglavlju opisuju se tehnologije korištene tijekom izrade programskog rješenja.

2.1 HTML

HTML (engl. *Hyper Text Markup Language*) [1] standardni je jezik za označavanje za *hipertekst*[1] dokumente namijenjene prikazivanju u web pregledniku. To je najosnovnija građevna komponenta weba. Web preglednici prihvaćaju HTML dokumente s web poslužitelja ili iz lokalne pohrane i pretvaraju ih u multimedijske web stranice. HTML logički uspostavlja strukturu web stranice i značenje inicijalno zadanih parametara za izgled dokumenta.

HTML elementi služe kao temelj za HTML stranice. Slike i drugi objekti, kao što su interaktivni obrasci, mogu se ugraditi u proizvedenu stranicu pomoću HTML elemenata. HTML omogućuje izradu organiziranih dokumenata izražavanjem strukturne semantike za tekst kao što su zaglavlja, odlomci, popisi, poveznice, citati i druge stavke.

HTML element razlikuje se od drugog sadržaja u dokumentu pomoću oznaka (engl. *tags*), koje se sastoje od naziva elementa u uglastim zagradama. Naziv elementa unutar oznake ne razlikuje velika i mala slova. Odnosno, može se pisati veli-

[1]skup dokumenata u elektroničkom obliku, pretežno tekstovnog i slikovnoga sadržaja, međusobno povezanih elektroničkim uputnicama, poveznicama

kim, malim slovima ili kombinacijom oba. Međutim, norma i predložena praksa je pisanje oznaka malim slovima.

2.2 CSS

CSS (engl. *Cascading Style Sheets*) [2] je jednostavan stilski jezik koji se koristi kako bi web stranice bile vizualno privlačnije. CSS se može koristiti za određivanje boje teksta, stila fonta, razmaka između odlomaka, načina na koji su stupci skalirani i organizirani, koje se pozadinske slike ili boje koriste, razlika u prikazu za različite uređaje i veličine zaslona te mnoštvo drugih efekata.

CSS ima jednostavnu sintaksu u kojoj selektori odlučuju na koji se dio *markupa* odnosi odgovarajući skup pravila navedenih u bloku deklaracije.

Informacije o prezentaciji dokumenta mogu se dobiti iz raznih izvora. Izvori mogu biti preglednik, korisnik i autor. Svaki izvor ima drugačiji prioritet, a kaskadno se odnosi na prikaz izvora s najvećim prioritetom.

2.3 TypeScript

TypeScript [3] proširuje *JavaScript* opcionalnim tipovima (kodni isječak 2.1) koji pružaju alate za razvoj velikih *JavaScript* aplikacija za bilo koji preglednik, poslužitelj ili operativni sustav. *TypeScript* se prevodi (engl. *compile*) u čitljiv standardizirani *JavaScript*. Budući da je *JavaScript* podskup *TypeScripta*, postojeći *JavaScript* programi također su valjani *TypeScript* programi.

TypeScript se može koristiti za stvaranje *JavaScript* aplikacija koje se mogu izvršavati i na klijentskoj i na poslužiteljskoj strani. Za prevođenje *TypeScripta* u *JavaScript* može se upotrijebiti zadani *TypeScript* prevoditelj ili poznati Babel.

TypeScript prihvaća definicijske datoteke koje pružaju informacije o tipovima iz postojećih *JavaScript* knjižnica. To omogućuje drugim programima da koriste vrijednosti definirane u datotekama na isti način na koji bi koristili *TypeScript* entitete sa statički upisanim vrijednostima.

```
interface FormValues {
    player_id?: number;
    disease: string;
    date: string;
}

interface IVaccinationForm {
    initialValues: FormValues;
}

const VaccinationForm: FC<IVaccinationForm> = ({ initialValues
   }) => {
...
```

Kodni isječak 2.1 Primjer definiranja tipova podataka u *TypeScriptu*

2.3.1 Yarn

Upravitelj paketa (engl. *package manager*) je alat koji automatski upravlja ovisnostima (engl. *dependencies*) projekta. Upravitelj paketa može se koristiti za instalaciju, deinstalaciju, ažuriranje i nadogradnju paketa, izmjenu postavki projekta, pokretanje skripti.

Yarn (engl. *Yet Another Resource Negotiator*) [4] je upravitelj paketa koji je Facebook objavio 2016. kao alternativu za npm (engl. *Node Package Manager*). Izvorna svrha Yarna bila je riješiti nedostatke npm-a kao što su problemi s performansama i sigurnošću. Yarn se brzo etablirao kao sigurno, brzo i pouzdano rješenje za upravljanje ovisnostima *JavaScripta*.

2.4 React

React [5] je knjižnica za izgradnju korisničkih sučelja (engl. *user interfaces*). *React* nije radni okvir (engl. *framework*), niti je ograničen samo na web. Koristi se zajedno

s drugim knjižnicama za prikazivanje u određenim okruženjima. *React Native*, na primjer, može se koristiti za izradu mobilnih aplikacija.

Programeri koriste *React* u kombinaciji s *ReactDOM* knjižnicom za izradu web aplikacija. Spoj *Reacta* i *ReactDOM-a* često je ono na što se odnosi pojam *React framework*.

Glavna je svrha *Reacta* smanjiti broj grešaka koje nastaju kada programeri stvaraju korisnička sučelja. To postiže korištenjem komponenti, koje su samostalni, logički dijelovi koda koji opisuju segment korisničkog sučelja. Ove se komponente mogu kombinirati kako bi se formiralo potpuno korisničko sučelje, a *React* apstrahira velik dio napora potrebnog za prikazivanje, dopuštajući programeru da se usredotoči na dizajn korisničkog sučelja.

2.4.1 Redux

Redux [6] je *JavaScript* spremnik stanja (engl. *store*) s predvidljivim ponašanjem. *Redux* olakšava razvoj pouzdanih, višeplatformskih (klijentskih, poslužiteljskih i izvornih) aplikacija koje je jednostavno testirati.

React-Redux knjižnica je koja nudi službene *React* poveznice (*bindings*) za *Redux* kao i dodatke koji omogućuju komponentama da komuniciraju s *Redux* spremnikom stanja. Programer određuje kako dohvatiti vrijednosti koje su potrebne komponenti iz *Reduxa*, a komponenta se automatski ažurira prema potrebi.

Redux Toolkit knjižnica je koja sadrži alate za pojednostavljenje uobičajenih slučajeva upotrebe kao što su postavljanje spremnika stanja, konstruiranje *reducera* (kodni isječak 2.2) i logika ažuriranja nepromjenjivih tipova. Sadrži najčešće korištene *Redux* dodatke (engl. addons) i nudi prihvatljive početne postavke za postavljanje spremnika stanja.

```
...
const initialState: AuthState = {
    data: <AuthPayloadData>{},
    isSubmitting: false,
    hasLoaded: false,
};
```

```
const extraReducers = {
    [authActions.login.pending.type]: (state: AuthState) => ({
        ...state,
        isSubmitting: true,
    }),
    [authActions.login.fulfilled.type]: (
        state: AuthState,
        { payload }: PayloadAction<AuthPayload>,
    ) => {
        localStorage.setItem('accessToken', payload.data.token)
   ;

        return {
            ...state,
            data: payload.data,
            isSubmitting: false,
            hasLoaded: true,
        };
    },
    [authActions.login.rejected.type]: (state: AuthState) => ({
        ...state,
        isSubmitting: false,
    }),
};
const auth = createSlice({
    name: 'auth',
    initialState,
    reducers: {},
    extraReducers,
});
export default auth.reducer;
```

Kodni isječak 2.2 Primjer reducera koristeći *Redux Toolkit*

2.4.2 Axios

Axios [7] je sličan *Fetch API-ju*, koji dolazi s *JavaScriptom*, po tome što proizvodi *Promise* objekt, ali ima mnogo snažnije mogućnosti. Moguće je poništiti zahtjev i postaviti vremensko ograničenje odgovora (engl. *response*) koristeći *Axios*. Osim toga, podržava napredak učitavanja (engl. *upload*), ima ugrađenu CSRF sigurnost i automatski transformira JSON (engl. *JavaScript Object Notation*) podatke. Bitna značajka *Axiosa* je njegova izomorfna priroda, koja omogućuje rad iste baze koda u *Node.js* aplikacijama i preglednika i poslužitelja.

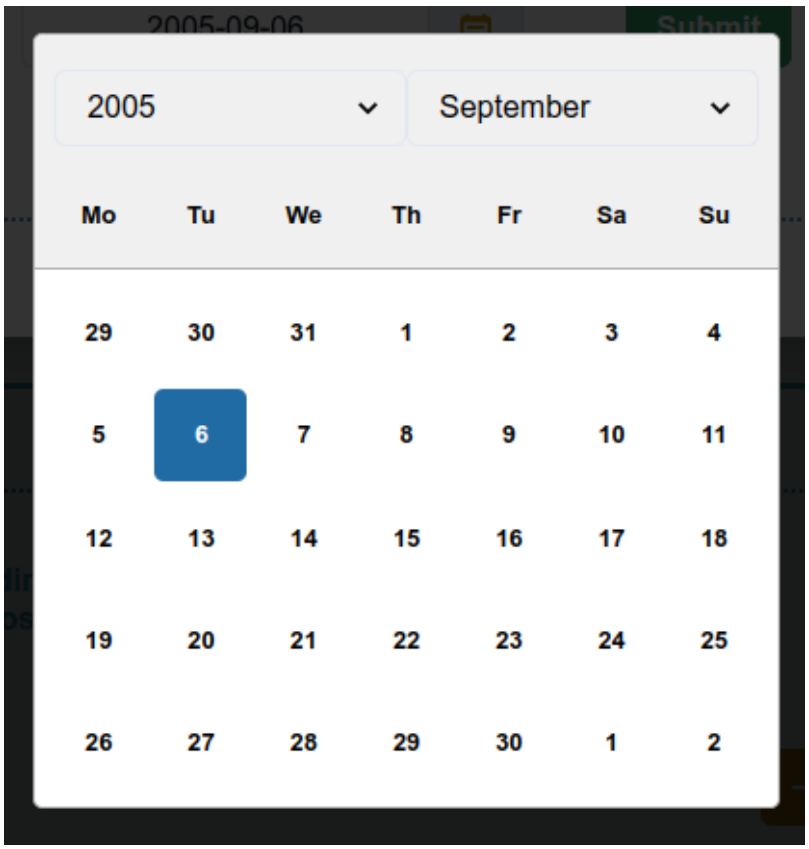
2.4.3 Chart.js

Chart.js [8] je *JavaScript* knjižnica za konstrukciju grafikona. Jednostavan je za korištenje, ali dovoljno robustan za stvaranje kompliciranih vizualizacija. U potpunosti je napisan u *JavaScriptu* i ne ovisi o dodatnim knjižnicama. Osim toga, nudi alate za animiranje ažuriranja podataka i uključivanje interakcije u grafikone. Za korištenje s *Reactom* također je potrebna knjižnica *react-chartjs-2* [9].

2.4.4 React Datepicker

Birač datuma (engl. *date picker*) je komponenta grafičkog korisničkog sučelja koja korisniku omogućuje odabir datuma iz kalendara i/ili vremena iz vremenskog raspona. Standardna praksa je da se uključi polje tekstualnog okvira koje, kada se klikne za unos datuma, prikazuje kalendar pored ili ispod polja, omogućujući korisniku da unese odgovarajući datum.

React Datepicker [10] učinkovita je komponenta za višekratnu upotrebu za prikazivanje datuma u dijalogu kalendara. Datum i vrijeme mogu se dodati pomoću *HTML* okvira za unos, budući da paket omogućuje jednostavnu prilagodbu. Izgled se može vidjeti na slici 2.1.



Slika 2.1 *React Datepicker*

2.4.5 React Table

React Table [11] skup je *React Hookova* i dodataka čiji je cilj pomoći programerima u sastavljanju logičkih aspekata kompliciranih mreža podataka u jedan učinkovit, proširiv API (engl. *Application Programming Interface*).

React Table ne renderira niti pruža komponente korisničkog sučelja tablice prema zadanim postavkama. Programer je odgovoran za renderiranje *markupa* tablice koristeći stanje i *callback* zadanih *React Table hookova*. Koristi se kada tablica zahtijeva standardne funkcije kao što su sortiranje, filtriranje i označavanje stranica, prilagođeni dizajn tablice bez utjecaja na funkcionalnost. Funkcionalnost tablice jednostavno je proširiva novim svojstvima.

2.4.6 Chakra UI

Chakra UI [12] sofisticirana je knjižnica *React* komponenti. Uključuje kolekciju *React* komponenti koje se mogu ponovno koristiti i sastaviti za izgradnju web aplikacija. *Chakra UI* jednostavna je, modularna i pristupačna. Omogućuje izradu pristupačnih aplikacija i ubrzava proces razvoja. Komponente je lako modificirati, proširiti i prilagoditi. Komponente su male i jednostavne za sastavljanje u veće konstrukcije. Izmjena između različitih kombinacija boja, poput svijetlih i tamnih, ili bilo kojeg drugog asortimana boja, iznimno je jednostavna.

2.5 Python

Python [13] je programski jezik koji se često koristi za izradu web stranica i aplikacija, automatizaciju zadataka i analizu podataka. *Python* je programski jezik opće namjene, što znači da se može koristiti za razvoj niza različitih programa i nije specijaliziran za rješavanje određenih problema.

2.6 Django REST Framework

Django [14] je web okvir temeljen na *Pythonu* koji omogućuje brz i siguran razvoj dugotrajnih web stranica. Otvorenog je koda (engl. *open-source*) i besplatan je. Prilagodljiv je, stoga se može koristiti za razvoj bilo koje vrste web stranice. Može komunicirati s bilo kojim okvirom na strani klijenta i može prenositi podatke u praktički bilo kojem formatu (JSON, HTML, XML, itd.). Interno daje izbor za nekoliko potrebnih funkcionalnosti za izradu web stranice, a osim toga, može se poboljšati korištenjem vanjskih komponenti.

Django REST Framework [15] omotač je normalnog *Django* okvira koji se koristi za stvaranje raznih vrsta API-ja. Prije uspostave API-ja koji koristi okvir REST (engl. *REpresentational State Transfer*), moraju se dovršiti tri faze: pretvaranje podataka modela u JSON/XML format (serijalizacija), prikaz podataka u *viewu* i određivanje URL-a za mapiranje u *viewset*.

REST je arhitekturalni stil za pružanje standarda između računalnih sustava na webu, čime se olakšava međusobna komunikacija sustava. Sustavi usklađeni s REST-om, često poznati kao *RESTful* sustavi, karakterizirani su svojim *statelessnessom* i odvajaju klijentske i poslužiteljske probleme.

Postoje 4 osnovne HTTP (engl. *Hyper Text Transfer Protocol*) metode koje koristimo u zahtjevima za interakciju s resursima u REST sustavu:

- GET – dohvaća resurs (prema id-u) ili skup resursa
- POST – kreira novi resurs
- PUT – ažuriraj navedeni resurs (prema id-u)
- PATCH – ažurira određena svojstva resursa na temelju njegovog id-a
- DELETE – uklanja resurs po id-u

Serializeri (kodni isječak 2.3) omogućuju kompliciranim podacima, kao što su skupovi upita (engl. *querysets*) i instance modela, da se prevedu u izvorne *Python* tipove podataka, koji se kasnije mogu jednostavno prikazati u JSON, XML ili drugom formatu. Nakon validacije dolaznih podataka, *serializeri* također omogućuju deserijalizaciju, dopuštajući da se raščlanjeni podaci transformiraju natrag u komplicirane tipove.

```
1  class PlayerSerializer(serializers.ModelSerializer):
2      general_medical_overview = GeneralMedicalOverviewSerializer
       (
3          required=False, many=True)
4      medical_anamnesis = MedicalAnamnesisSerializer(required=
       False, many=True)
5      medicine = MedicineSerializer(required=False, many=True)
6      vaccination = VaccinationSerializer(required=False, many=
       True)
7
8      class Meta:
9          model = Player
10         fields = '__all__'
```

Kodni isječak 2.3 Primjer serijalizera

2.7 PostgreSQL

PostgreSQL [16] besplatan je sustav za upravljanje relacijskim bazama podataka koji omogućuje brzo i jednostavno postavljanje te upravljanje bazama podataka. Navedeni sustav otvorenog koda nastao je 1996. godine te je kroz više od 20 godina razvoja postao jedan od najpopularnijih opcija za rad s bazama podataka u industriji. Glavne karakteristike su: arhitektura, pouzdanost, skalabilnost te performansa. *PostgreSQL*dostupan je na svim operacijskim sustavima te time pruža dodatnu razinu fleksibilnosti. Također, dostupan je veliki broj dodataka poput *PostGIS* sustava za rad s prostornim podacima koji proširuju osnovne mogućnosti baze podataka.

2.7.1 pgAdmin

pgAdmin najpopularnije je grafičko sučelje za razvoj i administraciju *PostgreSQL* baza podataka. Otvorenog je koda te je dostupan na svim operacijskim sustavima. Također, pruža veliki broj alata koji znatno pojednostavljuju i ubrzavaju rad s bazom podataka. Omogućuje administratoru brojne opcije za praćenje te grafički prikaz analitičkih podataka.

2.8 Alati korišteni za razvoj aplikacije

Pri razvoju aplikacije korišteni su različiti alati kako bi se ubrzao razvoj, poboljšala kvaliteta koda te upravljanje izvornim kodom.

2.8.1 Visual Studio Code

Visual Studio Code [17] popularni je uređivač teksta osmišljen za ubrzanje razvojnog ciklusa. Pruža korisniku veliki broj alata i dodataka koji olakšavaju razvoj aplikacija. Dodatno, olakšava uklanjanje pogrešaka te kontrolu verzije koda pomoću lakog povezivanje s *Git-om*. *Visual Studio Code* je besplatan te je dostupan na svim operacijskim sustavima.

2.8.2 Git

Git [18] je besplatan softver za upravljanje verzijom koda. Otvorenog je koda te predstavlja industrijski standard pri timskom radu na projektima neovisno o njihovoj veličini. Jednostavnost korištenja, brojni dodaci te izrazito brza performansa su glavne karakteristika koje ga izdvajaju od ostalih sustava za kontrolu verzije. Dodatno, dostupan je veliki broj besplatnih grafičkih sučelja koji dodatno olakšavaju razvoj aplikacija.

2.8.3 Redux DevTools i React Developer Tools

Redux DevTools [19] je ekstenzija za *Google Chrome* koja olakšava praćenje promjene stanja aplikacije te ispravljanje grešaka. Otvorenog je koda te uz *Redux* može se koristiti za bilo koju arhitekturu koja koristi stanje.

React Developer Tools [20] je ekstenzija za *Google Chrome* koja se koristi za uklanjanje grešaka pri razvoju aplikacija pomoću *React* knjižnice. Otvorenog je koda te olakšava rad s komponentama koje su glavna karakteristika *React* aplikacija. Dodatno, omogućuje pregled komponenti, njihovi izmjenu te prikaz njihove hijerarhije.

3 Modeliranje baze podataka

U ovom poglavlju detaljno je objašnjena analiza modela baze podataka.

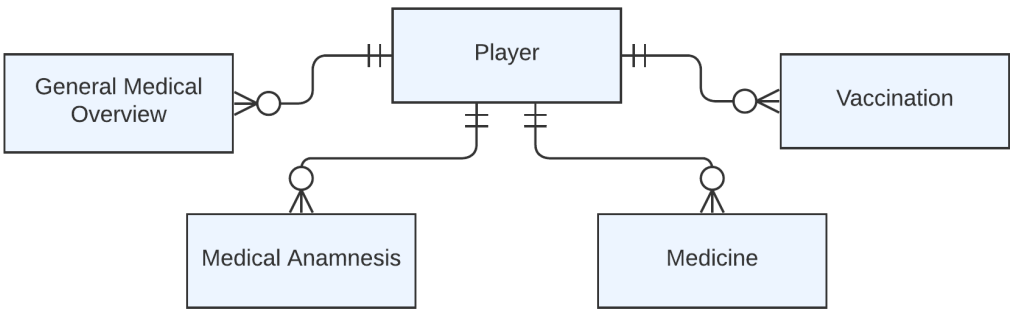
3.1 Specifikacija odnosa entiteta

Opis i objašnjenje veza unutar baze podataka.

1. igrač (engl. *Player*) - generalni medicinski pregled (engl. *General Medical Overview*)
    - Svaki generalni medicinski pregled mora se vezati na samo jednog igrača. Svaki igrač može imati više generalnih medicinskih pregleda, ali ne mora imati nijedan.
2. igrač - medicinska anamneza (engl. *Medical Anamnesis*)
    - Svaka medicinska anamneza mora se vezati na samo jednog igrača. Svaki igrač može imati više medicinskih anamneza, ali ne mora imati nijednu.
3. igrač - lijek (engl. *Medicine*)
    - Svaki lijek mora se vezati na samo jednog igrača. Svaki igrač može imati više lijekova, ali ne mora imati nijedan.
4. igrač - cjepivo (engl. *Vaccination*)
    - Svako cjepivo mora se vezati na samo jednog igrača. Svaki igrač može imati više cjepiva, ali ne mora imati nijedno.

3.2 Konceptualno modeliranje

Nakon specificiranja internih veza sustava, sljedeći korak je konstruiranje početnog modela odnosa entiteta (ER model) (slika 3.1) koji odgovara specifikaciji veza između različitih entiteta. Ne postoji jamstvo da će prvi ER model odgovarati konačnom rezultatu modeliranja baze podataka. Dijagram odnosa entiteta (ER) prikazuje entitete i njihove odnose. Kardinalnost se koristi za karakterizaciju svakog odnosa, a može ili ne mora biti potrebna ovisno o prirodi veze. Za neke veze dan je glagolski opis kako bi se razjasnila priroda interakcije između brojnih zasebnih entiteta.



Slika 3.1 Konceptualni ER model baze podataka

3.3 Izrada logičkog modela

Logički model, poput konceptualnog modela, služi kao koristan nacrt za dizajn sustava i buduće reference. Također pruža solidnu osnovu za stvaranje baze podataka. Logički modeli također se mogu koristiti za proširenje svojstava svakog objekta naznakom je li atribut broj, cijeli broj, niz ili niz alfanumeričkih znakova. Veze modela između entiteta također se mogu specificirati više nego u konceptualnom modelu podataka.

Logički model, poput konceptualnog modela, služi kao koristan nacrt za dizajn sustava i buduće reference. Također pruža solidnu osnovu za stvaranje baze poda-

taka. Logički modeli također se mogu koristiti za proširenje svojstava svakog objekta naznakom je li atribut broj, cijeli broj, niz ili niz alfanumeričkih znakova. Veze modela između entiteta također se mogu specificirati više nego u konceptualnom modelu podataka.

Korištenjem tehnike normalizacije pokušavaju se eliminirati suvišnosti iz baze podataka. Normalne forme čine normalizaciju. Svaka normalna forma eliminira određenu vrstu funkcionalne ovisnosti i osigurava ispunjenje svih prethodnih normalnih formi. Kako bi se uklonio rizik od anomalija, obično je dovoljno provesti relacije kroz prve tri normalne forme.

3.3.1 Funkcijske zavisnosti

Kako bi se konstruirao logički model, potrebno je identificirati koje odgovarajuće funkcijske veze postoje između karakteristika. Ključ kandidat je bilo koja grupa atributa koja određuje preostale atribute na jedinstven način. Prilikom uspostavljanja logičkog modela, mora se odabrati samo jedan primarni ključ kandidat ako relacija sadrži više ključeva kandidata. Također postoji mogućnost korištenja surogat (zamjenskih) ključeva kao primarnih ključeva. Surogat ključevi proizvoljni su identifikatori koji se ne generiraju iz podataka.

Primjenom definicije funkcijskih zavisnosti na podacima pronađene su sljedeće zavisnosti:

- $\{player\_id\}$ —> $\{name,\ position,\ leg,\ date\_of\_birth,\ games,\ image\}$
- $\{player\_id\}$ —> $\{general\_medical\_overview\_id,\ time\_made,\ height,\ weight,\ BP\}$
- $\{player\_id\}$ —> $\{medical\_anamnesis\_id,\ time\_made,\ family\_anamnesis,\ player\_anamne$ $symptoms\}$
- $\{player\_id\}$ —> $\{medicine\_id,\ time\_made,\ name,\ TUE\}$
- $\{player\_id\}$ —> $\{vaccination\_id,\ time\_made,\ disease,\ date\}$

3.3.2 Preslikavanje veza

Pravila o preslikavanju ER dijagrama koja definiraju na koji se način preslikavaju entiteti veze prema vrsti veze.

- veze 1:m - preslikavaju se tako da se primarni ključ entiteta na strani veze 1 dodaje kao strani ključ u entitet na strani veze m.
- veze 1:1
    1. veza je obavezna na oba kraja - atribut manje važnog entiteta postaje običan atribut, duplikati se izbacuju, a primarni ključ ostaje ključ važnijeg entiteta. Entiteti mogu ostati i razdvojeni te onda primarni ključ jednog entiteta postaje strani kluč drugog entiteta.
    2. veza je na jednom kraju obavezna, a na drugom opcionalna - primarni ključ entiteta na obaveznoj strani postaje strani ključ entiteta na opcionalnoj strani veze
    3. veza je na oba kraja opcionalna - entiteti ostaju razdvojeni, primarni ključ jednog entiteta koristi se kao strani ključ u drugom entitetu.
- veze m:n - kreira se entitet, primarni ključevi obaju entiteta koriste se kao strani ključevi u tom entitetu, a zajedno tvore složeni primarni ključ
- usporedne veze - za svaku vezu stvara se poseban strani ključ, prema osnovnim pravilima preslikavanja

Prema ovim pravilima dobivaju se sljedeće relacije:

- player(<u>id</u>, name, position, leg, date_of_birth, games, image)
- general_medical_overview(<u>id</u>, *player_id*, time_made, height, weight, BP)
- medical_anamnesis(<u>id</u>, *player_id*, time_made, family_anamnesis, player_anamnesis, symptoms)
- medicine(<u>id</u>, *player_id*, time_made, name, TUE)
- vaccination(<u>id</u>, *player_id*, time_made, disease, date)

3.3.3 Prva normalna forma

Prva faza tehnike normalizacije i temeljni zahtjev za rad s podatkovnim tablicama baze podataka. Ako tablica u bazi podataka nije u stanju proizvesti prvu normalnu formu, dizajn baze smatra se neadekvatnim. Prva normalna forma pruža skalabilni dizajn tablice koji se može brzo proširiti kako bi se znatno pojednostavio postupak dohvaćanja podataka. Svaki atribut tablice ne može sadržavati više vrijednosti. Za svaku zbirku povezanih podataka izrađuje se tablica, a vrijednost svakog skupa podataka prepoznaje se primarnim ključem specifičnim za skup podataka. Gore navedene relacije, prema tome, već zadovoljavaju uvjete.

3.3.4 Druga normalna forma

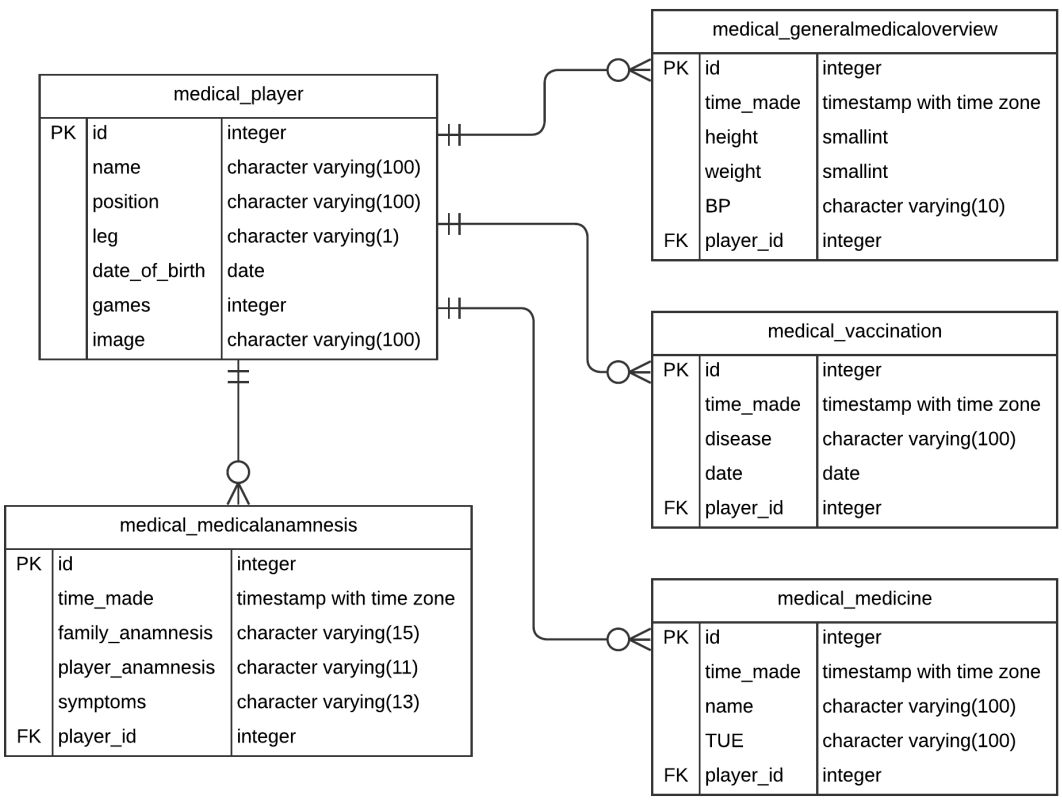
Relacija mora biti u prvoj normalnoj formi i ne smije imati djelomične ovisnosti. Relacije već zadovoljavaju i ovaj uvjet.

3.3.5 Treća normalna forma

Treća normalana forma osigurava smanjuje dupliciranje podataka, smanjuje anomalije podataka i čini model informativnijim. Tablica bi trebala biti u drugoj normalnoj formi i ne bi trebala imati nikakve tranzitivne ovisnosti. Navedne relacije zadovoljavaju sve uvjete, što znači da su u trećoj normalnoj formi.

3.3.6 Logički model

Kada su relacije provedene kroz sve tri normalne forme, moguće je kreirati logički model baze podataka 3.2.



Slika 3.2 Logički model baze podataka

4 Implementacija programskog rješenja

Sustav je razvijen korištenjem opisanih tehnologija (poglavlje 2). Za izradu korisničkog sučelja (engl. *frontend*) korišteni su *React* i *Chakra UI*, dok je za izradu poslužiteljskog dijela sustava (engl. *backend*) korišten je okvir *Django* uz *Django REST Framework*.

Na temelju logičkog modela razvoj poslužiteljskog dijela sustava započeo je razvojem fizičkog modela baze podataka prema logičkom modelu, pomoću modela razreda (engl. *class model*). Ti se razredi nalaze u zasebnoj datoteci *models.py*. Svaki model odgovara jednoj tablici u bazi podataka. Nakon izrade modela moraju se definirati pogledi i serijalizatori. Na temelju URL zahtjeva, svaki pogled provodi specijaliziranu logiku sustava. Datoteka *urls.py* sadrži popis svih URL-ova koje koristi sustav. Najznačajnija prednost *Django REST Framework ViewSeta* je da se generiranjem URL-a upravlja automatski (*router* klasom). Također je definirano na koji *ViewSet* su mapirani. *ViewSetovi* se nalaze u datoteci *views.py*, pri čemu je svaki *ViewSet* određen zasebnom klasom. Unutar ovih klasa, oni komuniciraju s modelima uz upotrebu serijalizatora kako bi dobili podatke iz baze ili ih spremili u bazu. *ViewSetovi* pružaju informacije u JSON formatu.

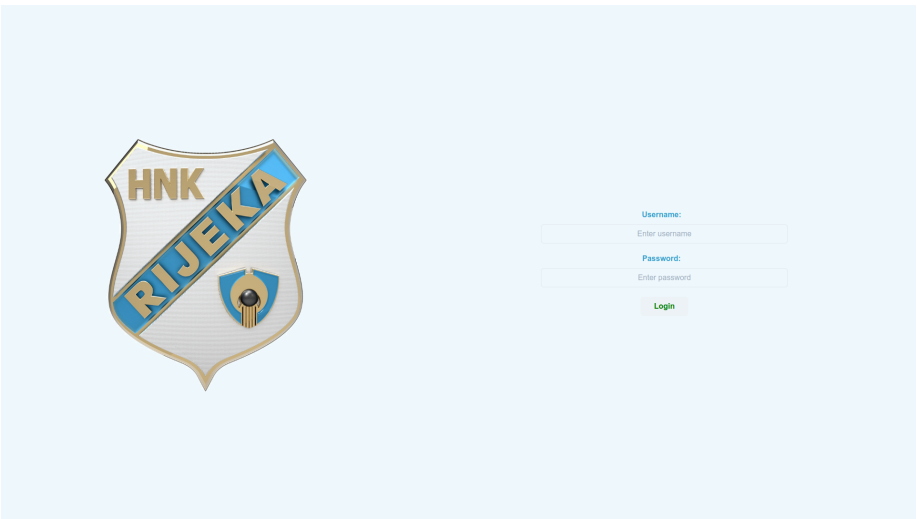
4.1 Korisničko sučelje

Korisničko sučelje implementirano je izradom *React* komponenata u kojima se odvija logika za dinamički prikaz sadržaja. Sučelje je dizajnirano jednostavno. S lijeve

strane prikazana je vertikalna navigacijska traka ili „ladica“ (engl. *drawer*) koja sadrži postavljeni logotip, ime korisnika ili korisničko ime (engl. *username*) ako ime nije postavljeno te linkovi na stranice i odjavu korisnika. Ukoliko korisnik nije prijavljen, prikazat će se stranica za prijavu. Za izradu grafičkog sučelja korištene su *Chakra UI* komponente koje su unaprijed stilizirane, ali se mogu i dodatno unaprijediti. *Chakra UI* također pruža izradu vlastite teme u kojoj mogu biti definirani posebni stilovi i ponašanja komponenata kako bi u cijeloj aplikaciji postojao konzistentni izgled.

4.2 Autentikacija korisnika

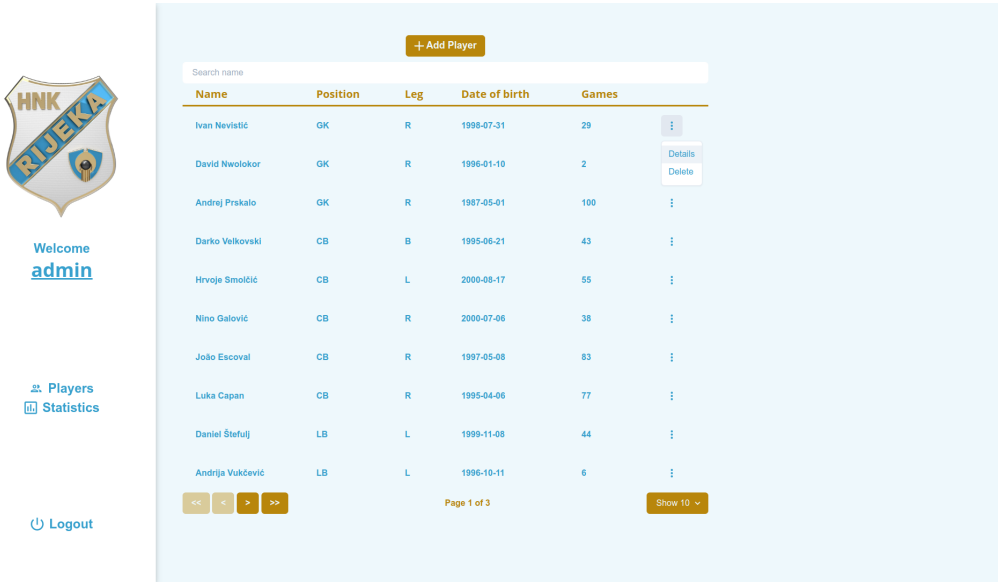
Za autentikaciju korisnika, korištene su *Django REST Framework TokenAuthentication* i *SessionAuthentication* klase. Uz pomoć *obtain_auth_token* pogleda korisnik prilikom prijave dobiva token koji se na klijentskoj strani sprema u pohranu preglednika iz kojeg pristupa aplikaciji. Taj token služi za autenticiranje zahtjeva prema API-ju. Prilikom prijave korisnik pruža svoje korisničko ime i lozinku. Registracija korisnika nije implementirana zbog namjene aplikacije. Nove korisnike u mogućnosti je dodati samo administrator kroz administratorsko sučelje. Na slici 4.1 je stranica za prijavu u sustav.



Slika 4.1 Obrazac za prijavu u sustav

4.3 Popis igrača

Popis igrača (slika 4.2) početna je stranica aplikacije. Popis igrača izlistan je u paginiranoj tablici koju je moguće sortirati prema osnovnim informacijama o igračima te filtrirati unosom u polje za unos iznad tablice. Tablica je izrađena uz pomoć *React Table* knjižnice koja pruža funkcije koje se koriste za definiranje ponašanja tablice prilikom određenih akcija. Posljednji stupac tablice sadrži gumb koji, kada se klikne, prikazuje popis mogućih radnji vezanih uz pripadajućeg igrača. Dostupne su akcije navigacija na stranicu za pregled detalja o igraču i brisanje igrača.

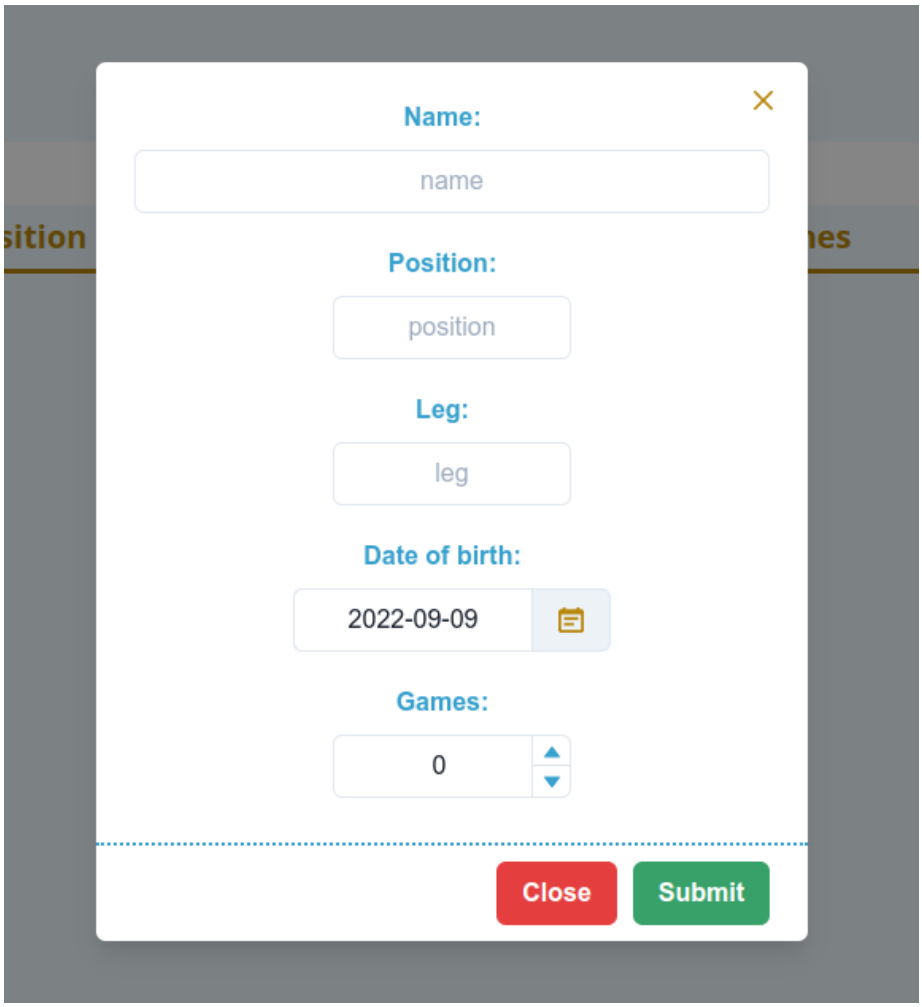


Slika 4.2 Popis igrača

4.3.1 Kreiranje igrača

Klikom na gumb *Add Player* otvara se skočni prozor (engl. *modal*) s obrascem (slika 4.3) kojeg je potrebno ispuniti odgovarajućim informacijama. Skočni prozor komponenta je *Chakra UI-a*, vrlo jednostavna za korištenje jer dolazi s već ugrađenim stilskim rješenjima i stanjem kojim je potrebno upravljati. Obrazac je izrađen pomoću knjižnice *Formik*. Formik pruža rješenja za upravljanje stanjem obrasca,

upravljanje validacijom i prikazom poruka grešaka te za upravljanje slanja obrasca. Izrada obrasca iste razine bez *Formika* bilo bi puno sporije i podložnije greškama. Prije slanja obrasca, na obrascu se vrši validacija prema shemi definiranoj korištenjem knjižnice *Yup*. *Yup* sheme su izuzetno ekspresivne i omogućuju modeliranje složenih, međuovisnih validacija ili transformacija vrijednosti. Poslani zahtjev s podacima iz obrasca prima poslužitelj u *PlayerViewSetu* na lokaciji definiranoj u *urls.py* datoteci. U *ViewSetu*, *PlayerSerializer* prevodi primljene podatke te se uz pomoć *CreateModelMixina* kreira novi igrač.

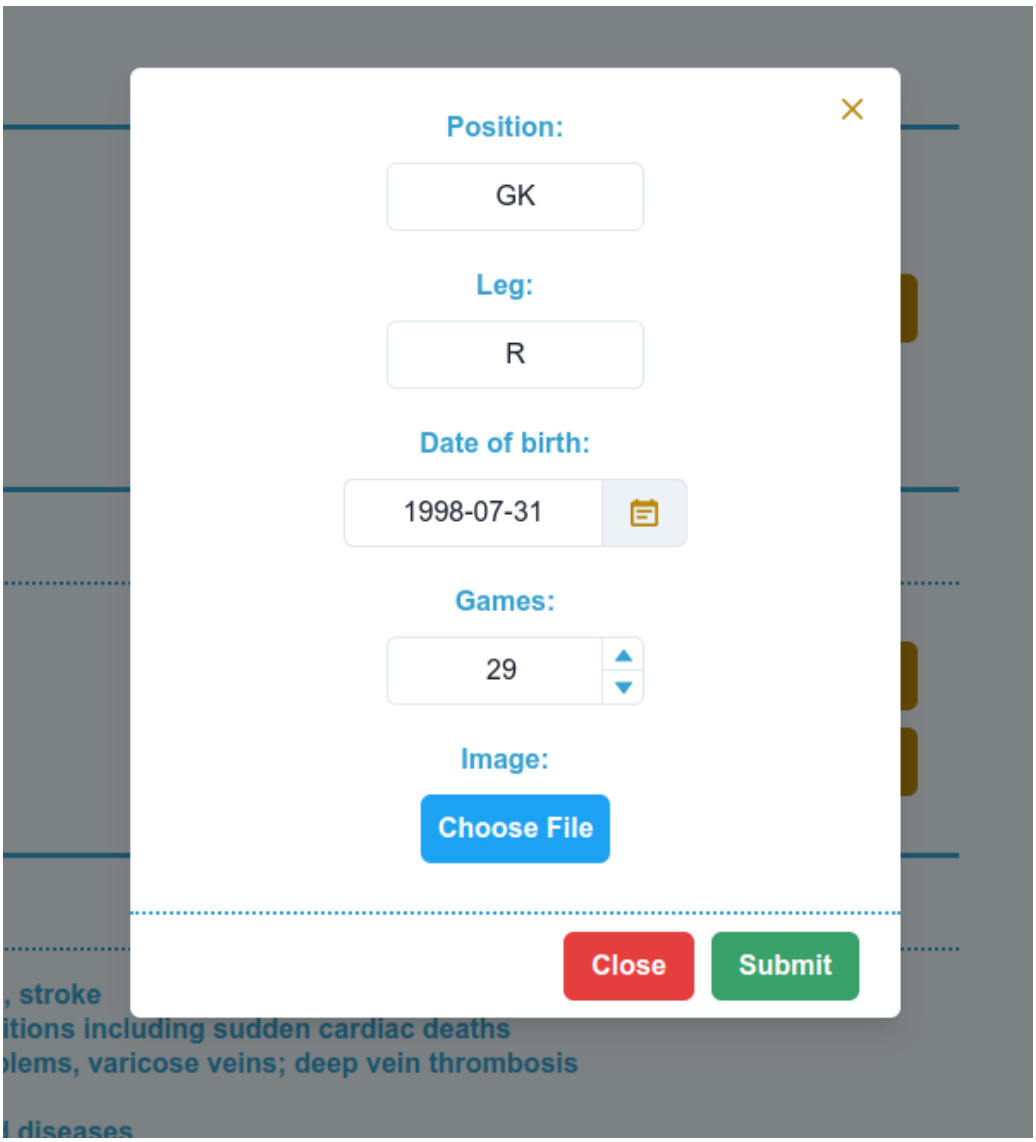


Slika 4.3 Obrazac za dodavanje novog igrača

4.4 Detalji o igraču

Slika 4.4 Detalji o igraču

Na ovoj stranici nalaze se sve informacije o izabranom igraču (slika 4.4) te sve dostupne akcije za unos i promjenu informacija. Na vrhu stranice nalazi se puno ime igrača nakon čega dolaze osnovne informacije o igraču koje se mogu promijeniti: pozicija, jača noga, datum rođenja i broj odigranih utakmica (slika 4.5). U sekciji za generalni pregled igrača mogu se vidjeti informacije s posljednjeg pregleda: visina, masa i tlak krvi. Generalne preglede moguće je uz dodavanje novog, izlistati i brisati. Anamneza igrača prikazuje informacije s posljednjeg pregleda o povijesti bolesti u obitelji, povijesti osobnih bolesti i trenutnim simptomima. Anamneze je moguće dodavati, izlistati i brisati. U sekcijama za prikaz lijekova i cjepiva moguće je osim naziva vidjeti i izuzeće za terapijsku uporabu (engl. *therapeutic use exemption*) u slučaju lijekova te datum cijepljenja u slučaju cjepiva. S obje liste moguće je brisati stavke i dodavati nove.

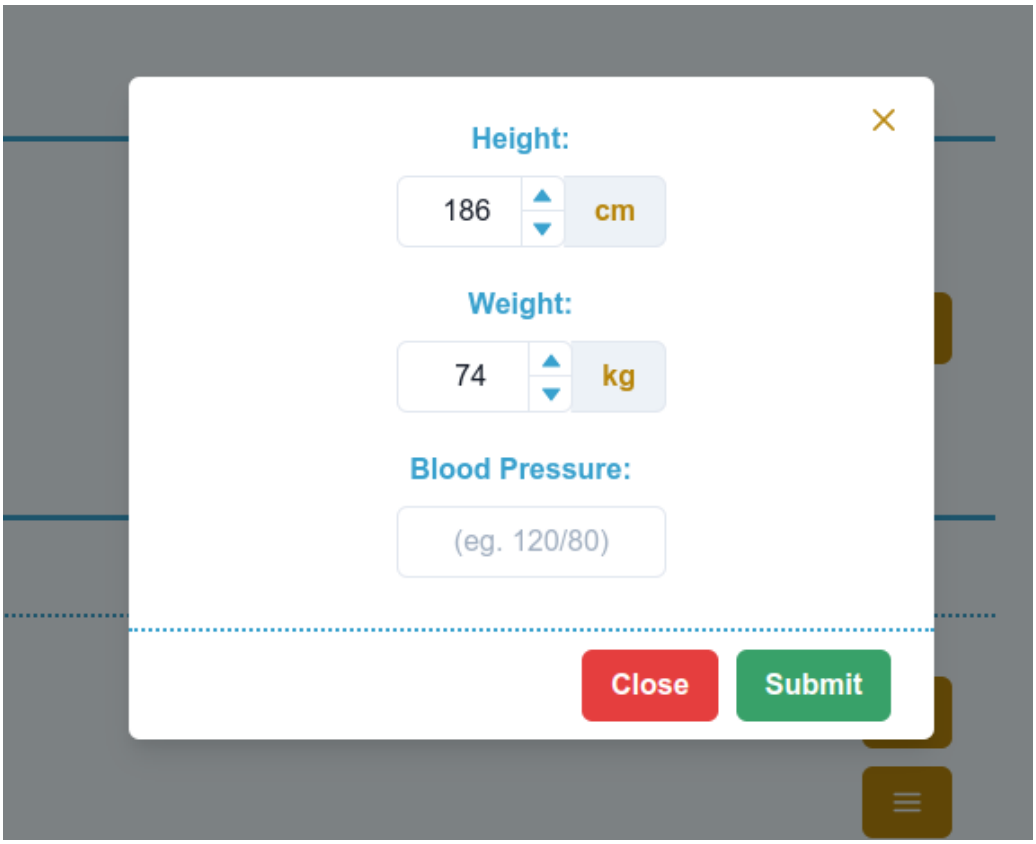


Slika 4.5 Obrazac za promjenu podataka o igraču

4.4.1 Generalni pregled igrača

Klikom na gumb s ikonom plusa otvara se skočni prozor (slika 4.6) u kojem je moguće dodati novu instancu generalnog pregleda igrača ispunjavanjem obrasca i slanjem u API. Na poslužiteljskoj strani zahtjev se prima u *GeneralMedicalOverviewViewSetu* na lokaciji definiranoj u *urls.py* datoteci. U *ViewSetu*, *GeneralMedicalOverviewSe-*

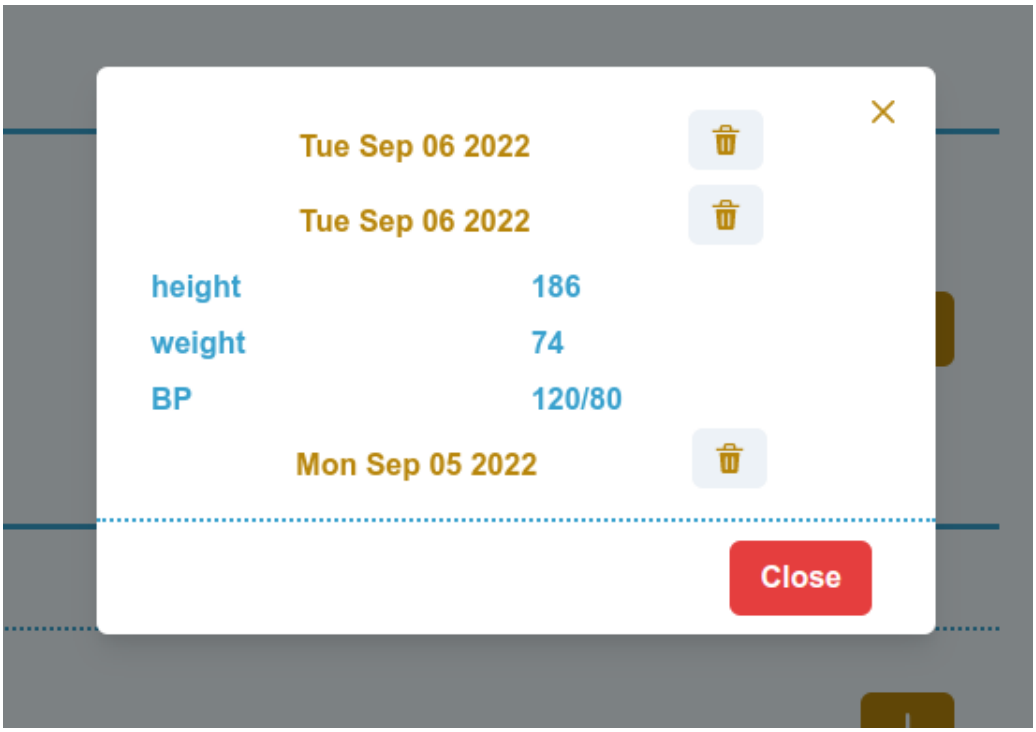
rializer prevodi primljene podatke te se poziva definirana *create* funkcija u kojoj se iz primljenih podataka izvlaci *player_id* kojim se pronalazi određeni igrač u bazi te se nakon toga kreira novi generalni pregled za tog igrača pozivanjem *create* metode *Django* Model klase.



Slika 4.6 Obrazac za dodavanje novog generalnog pregleda

Klikom na gumb s ikonom crtica otvara se skočni prozor (slika 4.7) u kojem je izlistana povijest svih pregleda počevši od posljednjeg. Klikom na gumb s određenim datumom otkrivaju se detalji tog određenog pregleda. Pored svake stavke nalazi se gumb s ikonom kante na čiji klik se šalje zahtjev za brisanje te stavke prema API-ju. Na poslužiteljskoj strani zahtjev se prima u *GeneralMedicalOverviewViewSetu* na lokaciji definiranoj u *urls.py* datoteci. U *ViewSetu*, *GeneralMedicalOverviewSerializer* prevodi primljene podatke te se poziva definirana *destroy* funkcija u kojoj

se prvo pronalazi objekt koji je potrebno izbrisati, prevede uz pomoć serijalizatora i nakon brisanja vrati na klijentsku stranu u odgovoru (engl. *response*), gdje su ove informacije bitne radi ažuriranja *redux storea*.



Slika 4.7 Povijest generalnih pregleda

4.4.2 Anamneza igrača

Kada se klikne gumb sa znakom plusa, pojavljuje se skočni prozor (slika 4.8). Ovaj prozor ima obrazac koji je potrebno ispuniti kako bi se dodala nova instanca anamneze igrača. Obrazac se tada može poslati API-ju. Zahtjev se prima na poslužiteljskoj strani u *MedicalAnamnesisViewSetu* na lokaciji navedenoj u datoteci *urls.py*. Unutar *ViewSeta*, *MedicalAnamnesisSerializer* prevodi podatke koji su primljeni, a zatim se poziva definirana funkcija *create*. Unutar funkcije, *player_id* se izdvaja iz podataka koji su primljeni te se koristi za pronalaženje određenog igrača unutar baze podataka. Nakon toga se stvara nova anamneza za tog igrača pozivanjem metode

create Django klase *Model*. Nakon što je generiranje anamneze dovršeno bez pogreške, vraćeni podaci se prevode kroz serijalizator, a zatim se šalje odgovor klijentskoj strani.

Slika 4.8 Obrazac za dodavanje nove anamneze

Kada se klikne gumb s ikonom crtica, pojavljuje se skočni prozor (slika 4.9). Ovaj prozor sadrži popis svih anamneza, počevši od najnovije. Kada se odabere datum klikom na gumb, prikažu se dodatne informacije o toj anamnezi. Postoji gumb s

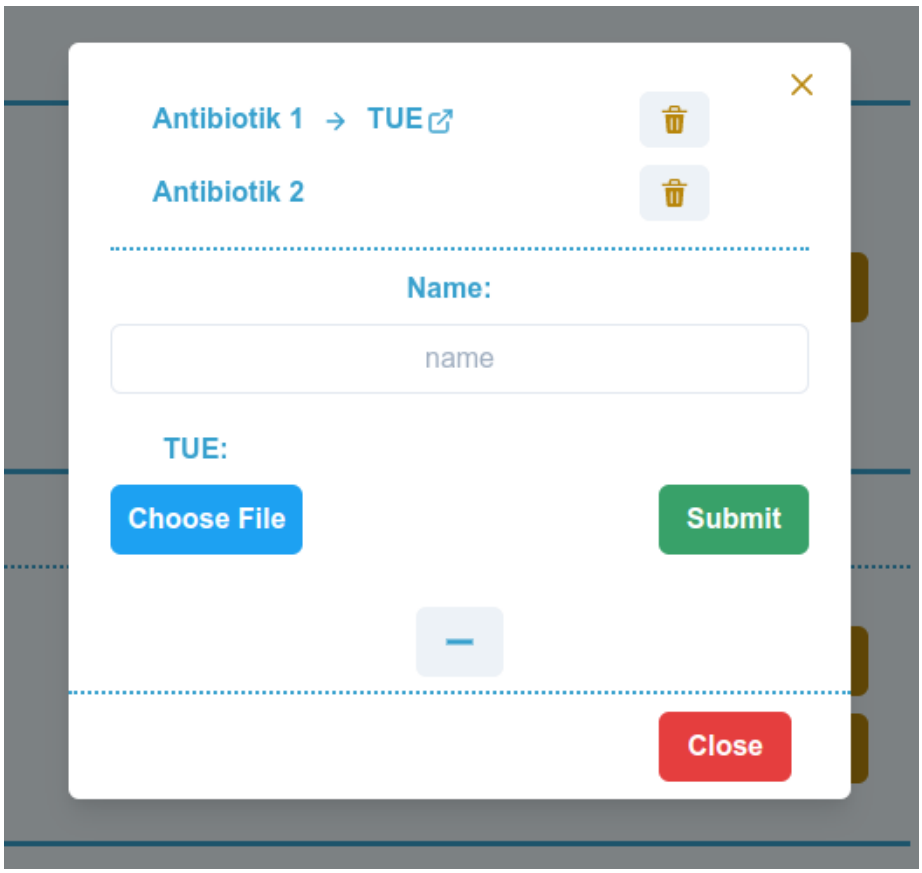
Slika 4.9 Povijest anamneza

ikonom kante pokraj svake stavke, a kada se na taj gumb klikne, API-ju se izdaje zahtjev za uklanjanje te određene stavke. Zahtjev se isporučuje u *MedicalAnamnesisViewSet* na poslužitelju koji se nalazi na lokaciji navedenoj u datoteci *urls.py*. U *ViewSetu*, *MedicalAnamnesisSerializer* prevodi podatke koji su primljeni, a zatim

se poziva definirana funkcija *destroy*. Unutar funkcije, prvo se traži objekt u bazi koji treba izbrisati, zatim se prevodi uz pomoć serijalizatora i na kraju, nakon brisanja, taj se objekt vraća klijentskoj strani u odgovoru. Ove su informacije bitne za precizno ažuriranje *redux storea*.

4.4.3 Lijekovi igrača

Nakon što se klikne na gumb prikazan kao znak olovke, pojavit će se skočni prozor (slika 4.10). Ovaj prozor nudi popis svih lijekova koji se mogu ukloniti iz sustava odabirom gumba prikazanog kao kanta za smeće. U sljedećem dijelu prozora nalazi se dio koji je skriven iza gumba u obliku znaka plus. Ovaj odjeljak sadrži obrazac koji se



Slika 4.10 Popis lijekova i obrazac za kreiranje novog lijeka

može koristiti za dodavanje novog lijeka. Slanjem popunjenog obrasca prema API-ju generira se novi lijek. Zahtjev se prima na strani poslužitelja u *MedicineViewSetu*. Primljene podatke prvo prevodi *MedicineSerializer*, unutar *ViewSeta*, a zatim se poziva definirana funkcija *create*. Unutar funkcije, *player_id* se dohvaća iz podataka koji su primljeni i zatim se koristi kako bi se pronašao taj određeni igrač unutar baze. Nakon toga, funkcija *create Django* klase *Model* koristi se za kreaciju novog lijeka za tog određenog igrača. Nakon uspješnog kreiranja novog lijeka, serijalizator se koristi za prevođenje podataka, a zatim se odgovor šalje na klijentsku stranu. Obrazac se može sakriti klikom na gumb prikazanog znakom minus.

Klikom na gumb za brisanje izdaje se API zahtjev. U *MedicineViewSet*, *MedicineSerializer* prevodi primljene podatke prije poziva definirane *destroy* funkcije. Funkcija počinje pretraživanjem baze podataka za stavku koju treba uništiti, zatim je prevodi uz pomoć serijalizatora i vraća objekt klijentu nakon brisanja tako da se *redux store* može ažurirati.

4.4.4 Cjepiva igrača

Nakon klika na gumb u obliku olovke otvorit će se skočni prozor (slika 4.11). Ovaj prozor nudi popis svih cjepiva koja se mogu izbrisati iz sustava klikom na ikonu kante za smeće. U sljedećem dijelu prozora nalazi se dio koji je skriven iza gumba u obliku znaka plus. Ovaj odjeljak nudi obrazac za dodavanje novog cjepiva. Slanjem ispunjenog obrasca API-ju izrađuje se novo cjepivo. *VaccinationViewSet* (kodni isječak 4.1) na strani poslužitelja prima zahtjev. Unutar *ViewSeta*, *VaccinationSerializer* prevodi dostavljene podatke prije poziva definirane funkcije *create*.

```
1 class VaccinationViewSet(RetrieveModelMixin, UpdateModelMixin,
    ListModelMixin, GenericViewSet):
2     serializer_class = VaccinationSerializer
3     queryset = Vaccination.objects.all()
4 
5     def create(self, request, *args, **kwargs):
6         player = Player.objects.get(id=request.data.pop('
    player_id'))
7         vaccination = Vaccination.objects.create(
```

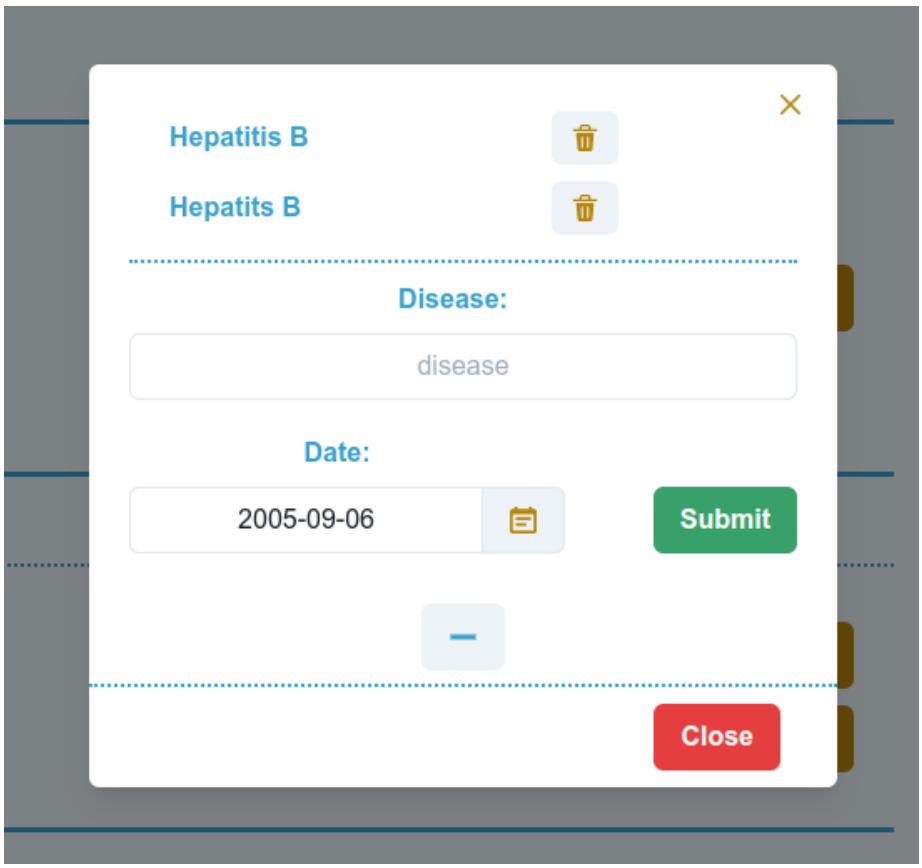
```
            player=player,
            **request.data
        )
        return Response(VaccinationSerializer(vaccination).data
    )

    def destroy(self, request, *args, **kwargs):
        id = kwargs.get('pk')
        vaccination = Vaccination.objects.get(id=id)
        responseData = VaccinationSerializer(vaccination).data
        vaccination.delete()
        return Response(responseData)
```

Kodni isječak 4.1 Primjer *ViewSeta*

Unutar funkcije, *player_id* izvlači se iz dostavljenih podataka i zatim se koristi za pronalaženje igrača unutar baze podataka. Zatim se *Djangova* metoda *create* klase *Model* koristi za generiranje novog cjepiva za igrača. Serijalizator se koristi za pretvaranje podataka nakon uspješne kreacije novog cjepiva, a odgovor se zatim isporučuje klijentskoj strani. Pritiskom na ikonu sa znakom minus obrazac se može sakriti.

Kada se klikne gumb za brisanje, upućuje se API poziv. Prije pokretanja definirane funkcije *destroy* unutar *VaccinationViewSeta*, *VaccinatonSerializer* prevodi dostavljene podatke. Funkcija dohvaća stavku koju treba izbrisati iz baze podataka, prevodi je pomoću serijalizatora i zatim vraća objekt u odgovoru klijentskoj strani, nakon što se objekt uspješno izbrisan.



Slika 4.11 Popis cjepiva i obrazac za kreiranje novog cjepiva

4.5 Statistika

Na stranici za statistiku iscrtana su dva grafikona, jedan koji prikazuje prvih deset igrača po broju odigranih utakmica, a drugi udio igrača po jačoj nozi. Za prikaz grafikona korišten je *Chart.js* i *react-chartjs-2* za *React* komponente potrebne za prikaz grafikona. Za prikaz je prvo potrebno pripremiti podatke i konfigurirati postavke prema kojima se određuje izgled i ponašanje grafikona (kodni isječak 4.2).

```
1 const BarChartComponent: FC<IBarChartComponent> = (chartData)
    => {
2     const data = {
3         labels: chartData.chartLabels,
```

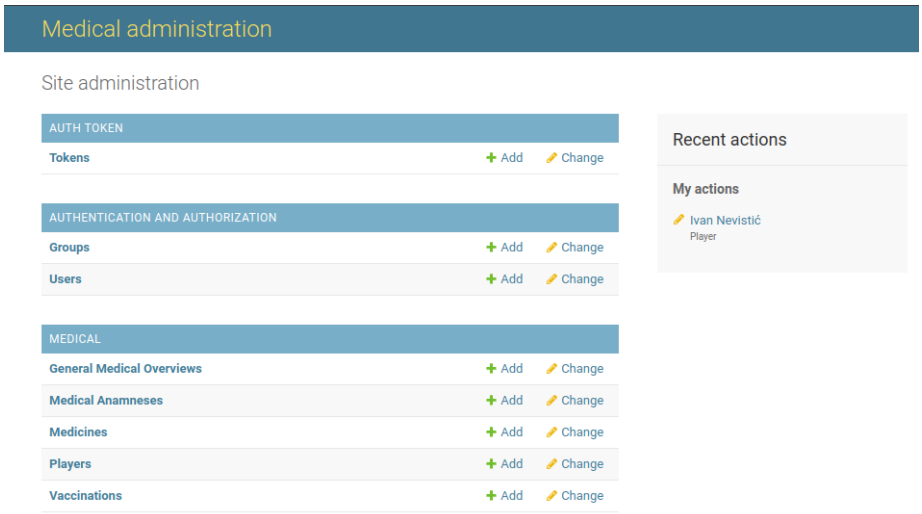
```
        datasets: [
            {
                display: false,
                label: '# of Games',
                data: chartData.chartData,
                backgroundColor: colors,
                borderColor: 'grey',
                borderWidth: 1,
            },
        ],
    };
    const options = {
        plugins: {
            legend: { display: false },
            title: {
                display: true,
                text: chartData.title,
                color: 'darkgoldenrod',
                font: { size: '20' },
            },
        },
    };
    return (
        <Flex marginY="20px">
            <Bar data={data} options={options} width={400}
   height={400} />
        </Flex>
    );
};
```

Kodni isječak 4.2 Stupčasti grafikon

Slika 4.12 Grafikoni s podacima o broju utakmica i jačoj nozi

4.6 Administratorsko sučelje

Jedan od najmoćnijih dijelova *Djanga* je automatsko administracijsko sučelje. Čita metapodatke iz definiranih modela kako bi pružilo sučelje usmjereno na model gdje pouzdani korisnici mogu upravljati sadržajem web stranice. Preporučena upotreba ograničena je da se koristi samo kao interni alat za upravljanje. Nije namijenjen za izgradnju cijelog *frontend* sučelja. Administracijsko sučelje koristi se samo kao alat za kreiranje pouzdanih korisnika.



Slika 4.13 Administracijsko sučelje

5 Zaključak

Glavni cilj izrade ovog web sustava bio je olakšati prikupljanje i pregled informacija o medicinskim pregledima sportaša. Implementirani sustav omogućava jednostavan unos podataka dobivenih medicinskim pregledom te pregled informacija s prijašnjih medicinskih pregleda vodeći računa o sigurnosti i privatnosti podataka. Unesene informacije se mogu izmjenjivati i brisati po potrebi.

Ovaj rad uključuje različite razvojne tehnologije i programske jezike i kao rezultat toga služi kao iznimno čvrst temelj za učenje. Korisničko sučelje, API poslužitelj i baza podataka podijeljeni su svaki u svoje zasebne entitete unutar ovog projekta, što je omogućilo dubinsko razumijevanje kako bi svaka od te tri komponente trebala funkcionirati.

Sustav ostavlja puno prostora za nadogradnju. Potrebno bi bilo implementirati napredniji sustav filtriranja i pretraživanja kako bi korisnici još lakše došli do potrebnih informacija i zanimljivih trendova koji se pojavljuju u sportskoj medicini.

Popis slika

Popis kodnih isječaka

Literatura

[1] Html. , s Interneta, https://en.wikipedia.org/wiki/HTML , rujan 2022.

[2] Css. , s Interneta, https://en.wikipedia.org/wiki/CSS , rujan 2022.

[3] Typescript. , s Interneta, https://www.typescriptlang.org/ , rujan 2022.

[4] Yarn. , s Interneta, https://yarnpkg.com/ , rujan 2022.

[5] React. , s Interneta, https://reactjs.org/ , rujan 2022.

[6] Redux. , s Interneta, https://redux.js.org/ , rujan 2022.

[7] Axios. , s Interneta, https://axios-http.com/ , rujan 2022.

[8] Chart.js. , s Interneta, https://www.chartjs.org/ , rujan 2022.

[9] react-chartjs-2. , s Interneta, https://react-chartjs-2.js.org/ , rujan 2022.

[10] React datepicker. , s Interneta, https://reactdatepicker.com/ , rujan 2022.

[11] React table. , s Interneta, https://react-table-v7.tanstack.com/ , rujan 2022.

[12] Chakra ui. , s Interneta, https://chakra-ui.com/ , rujan 2022.

[13] Python. , s Interneta, https://www.python.org/ , rujan 2022.

[14] Django. , s Interneta, https://www.djangoproject.com/ , rujan 2022.

[15] Django rest framework. , s Interneta, https://www.django-rest-framework.org/ , rujan 2022.

[16] Postgresql. , s Interneta, https://www.postgresql.org/ , rujan 2022.

[17] Visual studio code. , s Interneta, https://code.visualstudio.com/ , rujan 2022.

[18] Git. , s Interneta, https://git-scm.com/ , rujan 2022.

[19] Redux devtools. , s Interneta, https://chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklioeibfkpmmfibljd , rujan 2022.

[20] React developer tools. , s Interneta, https://microsoftedge.microsoft.com/addons/detail/react-developer-tools/gpphkfbcpidddadnkolkpfckpihlkkil , rujan 2022.

Pojmovnik

API engl. *Application Programming Interface*. 8

CSS engl. *Cascading Style Sheets*. 3

HTML engl. *Hyper Text Markup Language*. 2

HTTP engl. *Hyper Text Transfer Protocol*. 10

JSON engl. *JavaScript Object Notation*. 7

npm engl. *Node Package Manager*. 4

REST engl. *REpresentational State Transfer*. 9

Yarn engl. *Yet Another Resource Negotiator*. 4

Sažetak

U okviru ovog rada je proces dizajniranja i implementacije web sustava za praćenje medicinskih podataka profesionalnih sportaša. Sustav omogućuje obradu kliničkih podataka koji se dobivaju različitim dijagnostičkim postupcima. Sustav uključuje različite tipove medicinskih podataka, a sve ih mogu pregledavati i uređivati ovlašteni korisnici. Korisničko sučelje dizajnirano je s ciljem da bude što prirodnije i jednostavnije za korištenje. Kao rezultat korištenja izgrađenog sustava, administracija medicinskih podataka je pojednostavljena. To je zato što liječnici imaju centraliziranu lokaciju s koje mogu dobiti kontrolirani pregled relevantnih podataka.

Ključne riječi **— administracija medicinskih podataka, web sustav, Django REST Framework, React, autentikacija**

Abstract

Within the scope of this thesis is the process of designing and implementing a web system for tracking the medical data of professional athletes. The system makes it possible to handle the clinical data that is obtained from various diagnostic procedures. The system incorporates a variety of medical data types, all of which are viewable and editable by authorized users. The user interface was designed with the goal of being as natural and easy to use as possible. As a result of the use of the system that has been built, the administration of medical data is simplified. This is because physicians have a centralized location from which they can gain a controlled view on the relevant data.

Keywords **— medical data administration, web system, Django REST Framework, React, authentication**