Web aplikacija za potporu i praćenje fizikalnih terapija

Česnek, Filip

Undergraduate thesis / Završni rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:462451

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-14**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

WEB APLIKACIJA ZA POTPORU I PRAĆENJE FIZIKALNIH TERAPIJA

Završni rad

Filip Česnek

Osijek, 2018.

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

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

Osijek, 18.09.2018.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada

Ime i prezime studenta:	Filip Česnek
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3758, 20.09.2017.
OIB studenta:	22974242059
Mentor:	Prof.dr.sc. Goran Martinović
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Web aplikacija za potporu i praćenje fizikalnih terapija
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Predložena ocjena 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: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	18.09.2018.
Datum potvrde ocjene Odbora:	26.09.2018.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 26.09.2018.

Ime i prezime studenta:	Filip Česnek
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3758, 20.09.2017.
Ephorus podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za potporu i praćenje fizikalnih terapija**

izrađen pod vodstvom mentora Prof.dr.sc. Goran Martinović

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:

TABLICA SADRŽAJA

1. UVOD 1
   1.1. Zadatak završnog rada 1
2. IZAZOVI U POTPORI I PRAĆENJU FIZIKALNIH TERAPIJA 2
   2.1. Fizikalne terapije 2
      2.1.1. Maksimalni volumen kisika (VO_2) 2
   2.2. Pregled postojećih rješenja 3
      2.2.1. Web stranica ExRx 3
      2.2.2. Web stranica FitnessBlender 4
      2.2.3. Web stranica Spine – Health 5
      2.2.4. Web stranica PhysioTherapy eXercises 5
      2.2.5. Osvrt na postojeća rješenja 6
3. IDEJA VLASTITOG RJEŠENJA 7
   3.1. Klijentska strana 7
   3.2. Poslužiteljska strana 9
   3.3. Opis programskih okruženja i tehnologija 10
      3.3.1. JSON 10
      3.3.2. DOM 11
      3.3.3. Ovisnosti 12
      3.3.4. Vue.js 12
      3.3.5. MongoDB 17
      3.3.6. Node.js i Express.js 19
   3.4. Prikaz implementacije vlastitog rješenja 20
      3.4.1. Korisnički profil 22
      3.4.2. Dodavanje, izmjena, dohvaćanje vježbi i dohvaćanje prijedloga 30
      3.4.3. Povijest pregledanih vježbi i plan vježbi 35
4. TESTIRANJE RADA APLIKACIJE 39
   4.1. Korisnički profil 39
   4.2. Dodavanje, izmjena i dohvaćanje vježbi 41
   4.3. Povijest pregledanih vježbi i plan vježbi 45
   4.4. Performanse web aplikacije 47

5. ZAKLJUČAK 49
LITERATURA 50
SAŽETAK 52
ABSTRACT 53
ŽIVOTOPIS 54
PRILOZI 55

1. UVOD

Zdrava kralježnica jedan je od prioriteta za normalno funkcioniranje u svakodnevnom životu. Bol u leđima je vodeći uzrok invaliditeta kod osoba mlađih od 45 godina, a u većini slučajeva bol je uzrokovana nepravilnim i nezdravim navikama, kao što su, navedeno u [1]: manjak tjelesne aktivnosti, nepravilno, pogrbljeno držanje, nepravilno dizanje tereta, pretilost, pušenje, te nedovoljan unos kalcija i vitamina D. Vježbe za kralježnicu su iznimno bitne i pomažu osigurati jakost i snagu ligamenata i mišića oko kralježnice, održavaju ju fleksibilnom, te stimuliraju i kralježnicu i njene strukture na regeneraciju.

Cilj ovog završnog rada je napraviti web aplikaciju koja će omogućiti korisniku stvaranje osobnog plana terapijskih vježbi, praćenja osobnog napretka, te davanja prijedloga relevantnih vježbi na temelju određenih parametara koje korisnik unese.

U drugom će se poglavlju dati kratki pregled terapija, te će se analizirati već postojeća rješenja za ovaj problem. U trećem poglavlju dat će se prijedlog vlastitog rješenja, opis programskih okruženja i tehnologija korištenih za realizaciju rješenja, te će biti prikazani i opisani dijelovi koda za bitnije funkcije aplikacije. U četvrtom poglavlju, testirat će se ispravan rad aplikacije, kao i performanse.

1.1. Zadatak završnog rada

U radu je potrebno opisati fizikalne terapije za kralježnicu, te razraditi model ponude vježbi, definiranja načina izvođenja, praćenja napretka i analize rezultata fizikalnih terapija. Nadalje, potrebno je opisati programske okoline i tehnologije koje će se koristiti u izradi aplikacije, kao što su Node.js, Express.js, MongoDB i Vue.js. Programski treba ostvariti korisničko sučelje s mogućnošću prijave korisnika, unos postavki terapije, prikaz vježbi, te analizu i prikaz rezultata terapije. Također, treba omogućiti pristup bazi podataka i pohranu podataka. Ostvarenu aplikaciju treba ispitati na odgovarajućem skupu primjera.

2. IZAZOVI U POTPORI I PRAĆENJU FIZIKALNIH TERAPIJA

2.1. Fizikalne terapije

Prema [2], ako bol u leđima traje između dva i šest tjedana, ili ako se bol često ponavlja, fizička terapija je često preporučena. Neki liječnici preporučaju fizičku terapiju i ranije, pogotovo ako je bol jaka. Općenito, cilj fizikalne terapije je smanjenje boli, povećanje funkcionalnosti, i pružanje edukacije kako bi se smanjila mogućnost povratka boli u budućnosti. Plan fizikalne terapije uglavnom ima dvije komponente: pasivna fizikalna terapija (hladni/topli oblozi, jontoferoza, ultrazvuk), i aktivna fizikalna terapija (vježbe).

Generalno, plan vježbi za bol u leđima trebao bi sadržavati kombinaciju vježbi za istezanje leđa, vježbi za jačanje leđa, te tjelesne aktivnosti poput hodanja, bicikliranja ili plivanja. Prema [3], kada se rade na pravilan, kontroliran, progresivan način, vježbe za olakšavanje boli u leđima imaju brojne pozitivne učinke, kao npr. jačanje mišića koji podupiru kralježnicu, što dovodi do smanjenja pritiska na diskove kralježnice, ublažavanje ukočenosti i poboljšanje pokretljivosti, poboljšavanje cirkulacije, te smanjenje frekvencije epizoda boli, i smanjenje jakosti boli kada se dogodi.

2.1.1. Maksimalni volumen kisika (VO_2)

Jedna od važnijih stvari pri izvođenju fizikalnih terapija je sposobnost pacijenta da bez poteškoća izvede neku vježbu. Za mjerenje te sposobnosti pacijenta da izvede tu fizičku aktivnost često se koristi VO_2 test. VO_{2max} je numerička mjera maksimalne razine opterećenja kod koje, bez obzira koliko disali, neće doći do porasta razine kisika u tijelu, drugim riječima, to je maksimalna razina potrošnje kisika tijekom vježbanja. VO_{2max} se često koristi kao pokazatelj zdravlja neke osobe. Postoji nekoliko načina izračuna i procjene maksimalnog volumena kisika. Kod testiranja atletičara, prema [4], mjeri se udio kisika i ugljika u udisanom i izdisanom zraku tijekom vježbanja, te se VO_2 računa Fickovom jednadžbom. Postoji još nekoliko metoda, no najrelevantnija metoda za ovaj rad je, prema [5], procjena VO_{2max} parametra koristeći dob, spol, broj otkucaja srca u jednoj minuti pri mirovanju, te opseg struka. Kod ove metode, postoje dvije formule prema kojima se procjenjuje VO_{2max} parametar, gdje se prema spolu osobe odabire formula.

Tako za muškarce vrijedi sljedeće:

$$100.27 - (0.296 * \mathrm{A}) - (0.369 * \mathrm{WC}) - (0.155 * \mathrm{RHR}) + (0.226 * \mathrm{PA}_{index}) \quad (2\text{-}1)$$

A za žene sljedeći izraz:

$$74.74 - (0.247 * \mathrm{A}) - (0.259 * \mathrm{WC}) - (0.114 * \mathrm{RHR}) + (0.198 * \mathrm{PA}_{index}) \quad (2\text{-}2)$$

gdje za obje formule vrijedi:

- A – dob osobe,
- WC – opseg struka,
- RHR – broj otkucaja srca u jednoj minuti pri mirovanju,
- PA_{index} – indeks tjelesne aktivnosti osobe.

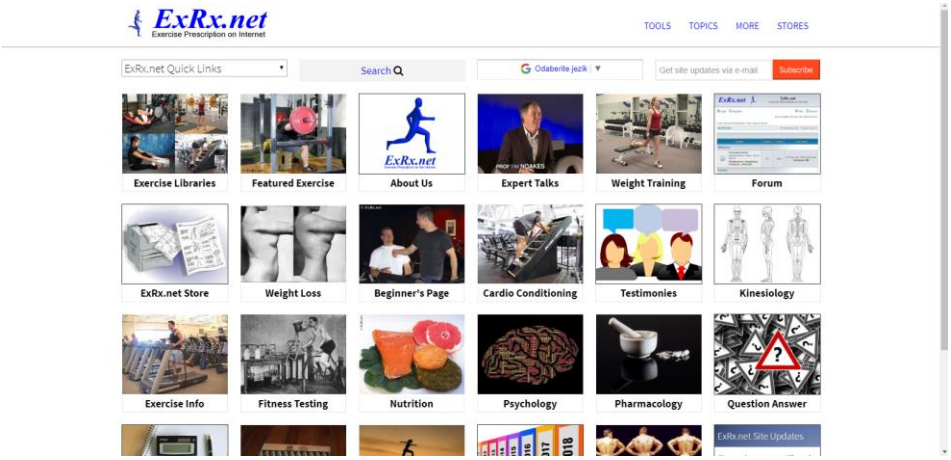
Prosječan muškarac ima VO_{2max} u rasponu od 35–40 mL/(kg·min), a prosječna žena 27–31 mL/(kg·min).

2.2. Pregled postojećih rješenja

Iako postoje različite internetske stranice koje pružaju mogućnost pronalaska, opisa i prikaza različitih vježbi, rijetko koja daje mogućnosti kao što su spremanje vježbi i stvaranje vlastitog plana vježbanja ili davanje prijedloga vježbi na temelju unešenih parametara o korisniku, i sve stranice su na engleskom jeziku. U nastavku je navedeno nekoliko primjera pronađenih stranica koje se bave sličnom tematikom.

2.2.1. Web stranica ExRx

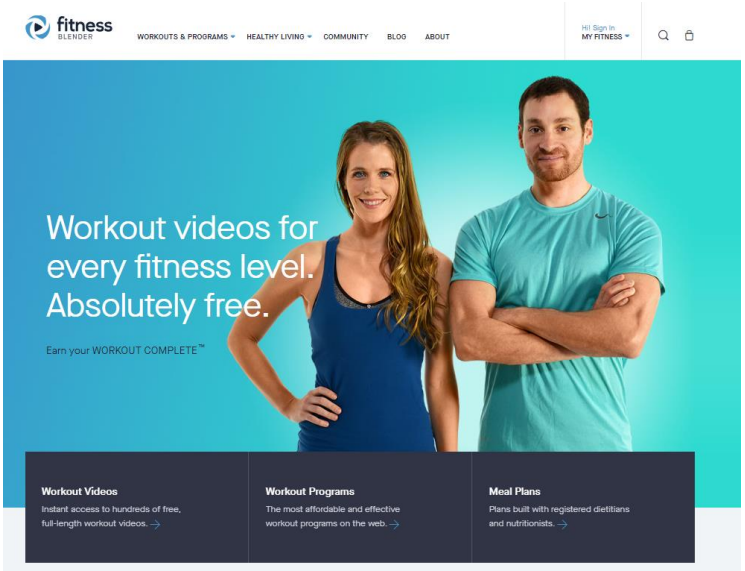
ExRx [6] je statična internetska stranica koja je ogromna zbirka od oko 1800 različitih vježbi koje nisu nužno medicinskog karaktera. Stranica, osim vježbi, nudi i različite druge usluge prikazane na slici 2.1, kao što su razni fitness kalkulatori koji mogu obavljati jednostavne funkcije poput pretvaranja nekih relevantnih mjernih jedinica (masa, temperatura, udaljenost) ili obavljati složenije funkcije kao što je procjena fizičke sposobnosti osobe na temelju unesenih parametara. Osim kalkulatora, web stranica ExRx nudi aktivan forum za diskusiju raznih tema vezanih za vježbe i zdrav način života, te medicinske članke iz područja poput kineziologije, psihologije i farmakologije.



Slika 2.1. *Naslovnica internetske stranice ExRx.net [6]*

2.2.2. Web stranica FitnessBlender

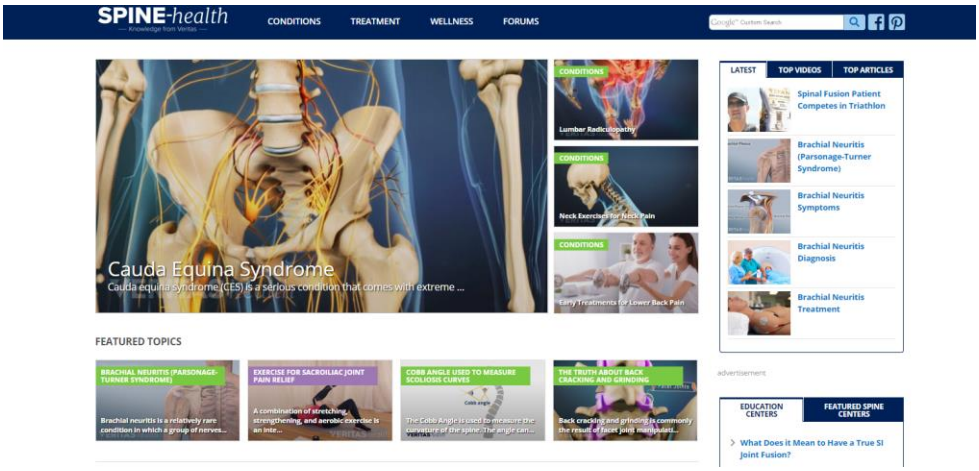
FitnessBlender [7], čija je naslovna stranica prikazana na slici 2.2, je naprednija web stranica u usporedbi s web stranicom ExRx, koja za neregistrirane korisnike nudi pregled različitih videa tjelovježbi, te omogućuje njihovu pretragu po različitim kriterijima. Osim videa, FitnessBlender nudi kupnju planova prehrane i tjelovježbe, te čitanje članaka i bloga. Registriranim korisnicima omogućeno je spremanje vježbi u listu favorita, zakazivanje vježbi za određeni dan, što je vizualno prikazano kalendarom, te komentiranje na stranici neke vježbe.



Slika 2.2. *Naslovnica internetske stranice FitnessBlender [7]*

2.2.3. Web stranica Spine – Health

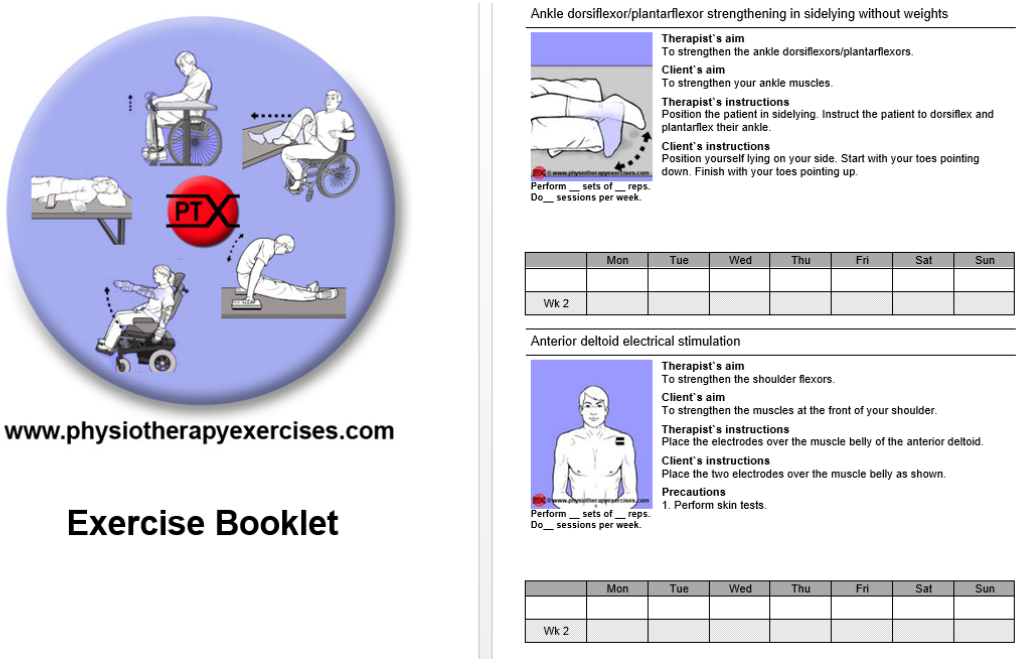
Spine–Health [8], čija je naslovnica prikazana na slici 2.3, je internetska stranica na kojoj se nalaze različiti medicinski članci čija je svrha educirati čitatelje i pomoći im razumjeti kralježnicu i medicinska stanja vezana za kralježnicu. Osim medicinskih članaka, stranica daje mogućnost pronalaska liječnika i medicinskih centara koji se bave kralježnicom u blizini (vrijedi samo za Sjedinjene Američke Države), te aktivan forum za diskusiju.



Slika 2.3. *Naslovnica internetske stranice Spine-Health [8]*

2.2.4. Web stranica PhysioTherapy eXercises

PhysioTherapy eXercises [9] je internetska stranica na kojoj se oko 1400 fizioterapeutskih vježbi za ljude s ozljedama i invaliditetom. Moguće je pretraživati, tj. filtrirati vježbe pomoću različitih kriterija, kao što su dob, vrsta ozljede, težina vježbe, potrebna oprema, vrsta vježbe i ozlijeđeni dio tijela. Nadalje, moguće je označiti željene vježbe i spremiti ih u formatu spremnom za ispis (docx, html, pdf). U spremljenom dokumentu nalaze se upute za izvođenje vježbe, ilustracija vježbe, cilj koji se želi postići, te tablica za praćenje rezultata, što je prikazano na slici 2.4. Osim odabira individualnih vježbi, moguće je odabrati i neke od programa vježbi za neke ozljede, kao što su ozljede leđne moždine ili povrede kralježnice zbog naglog pokreta glave.



Slika 2.4. *Prikaz spremljenih vježbi sa web stranice PhysioTherapy eXercises [9]*

2.2.5. Osvrt na postojeća rješenja

Iako većina stranica imaju bogatu kolekciju vježbi, poučnih medicinskih članaka i korisnih alata, većina njih ima zastarjeli dizajn, te se dosta teško snaći na tim stranicama. Neke stranice nude mogućnost registracije korisnika, te dodavanja vježbi na popis, no nijedna ne nudi personalizirani sustav prijedloga vježbi, kao ni digitalni sustav praćenja napretka.

3. IDEJA VLASTITOG RJEŠENJA

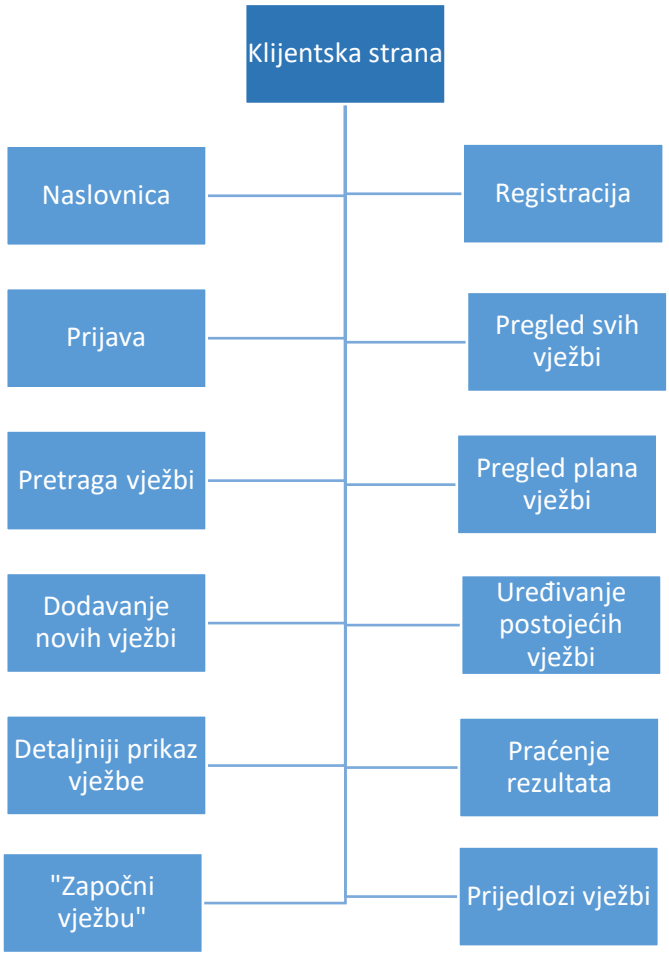
U ovom poglavlju bit će opisana struktura web aplikacije i mogućnosti koje ona pruža. Osim toga, opisat će se programska okruženja, tehnologije i druga pomagala pomoću kojih je web aplikacija realizirana. Naposljetku, detaljno će se opisati i objasniti različiti dijelovi koda aplikacije kako na korisničkom sučelju, tako i na poslužiteljskoj strani. Aplikacija će imati mogućnost registracije, prijave, uređivanja korisničkog profila, traženja vježbi po kriterijima, detaljniji prikaz vježbe, dodavanja vježbi u plan vježbi, praćenje rezultata vježbi i prikaz prijedloga vježbi.

3.1. Klijentska strana

Na klijentskom dijelu koji je vizualno prikazan na slici 3.1 nalazit će se različiti vizualni, interaktivni dijelovi aplikacije izrađeni pomoću programskog okvira Vue.js, a

1. **Registracija** – Stranica na kojoj korisnik unosi podatke kako bi imao pristup naprednijim mogućnostima web aplikacije, kao što je dodavanje vježbi u plan, praćenje rezultata, povijest pregledanih vježbi i prijedlozi vježbi. Podaci koje korisnik unosi su e-mail i lozinka, te neki osobni podaci (dob, spol, opseg struka, broj otkucaja srca u mirovanju, bolno područje tijela) koji služe za prikaz prijedloga vježbi.
2. **Prijava** – Stranica na kojoj registrirani korisnik unosi e-mail i zaporku radi prijave na stranicu i pristupu naprednijim mogućnostima web aplikacije.
3. **Pregled svih vježbi** – Stranica na kojoj su izlistane sve vježbe spremljene u bazi podataka, gdje su prikazne osnovne informacije o njima, te na kojoj je tražilica kojom je moguće pretražiti vježbe.
4. **Pregled plana vježbi** – Tablica u kojoj su prikazane vježbe koje je korisnik dodao u svoj plan. Prema tablici i sučelju moguće je pratiti trenutni napredak plana terapije.
5. **Dodavanje / uređivanje vježbi** – Stranica dostupna samo administratoru preko koje se unose podaci o vježbi i ona se sprema u bazu podataka.
6. **Detaljniji prikaz vježbe** – Stranica na kojoj je moguće vidjeti podatke o vježbi kao što je opis, upute za izvođenje, video i ostale podatke, te je moguće kliknuti na gumb kojim se započinje vježba (ukoliko je korisnik registriran).

7. **„*Start Exercise*“** – Stranica na kojoj je prikazan video vježbe i upute za izvođenje, ispod kojih se nalazi tajmer podešen za tu određenu vježbu, nakon čijeg isteka je moguće vježbu označiti kao završenu.
8. **Praćenje rezultata** – Traka napretka koja prikazuje postotak odrađenih vježbi, a ažurira se kada korisnik završi s vježbanjem neke vježbe koja se nalazi u njegovom planu.
9. **Prijedlozi vježbi** – Stranica na kojoj su prikazana vježbe koje su prema korisnikovim parametrima njemu prilagođene, te ih korisnik može dodati u plan vježbi.

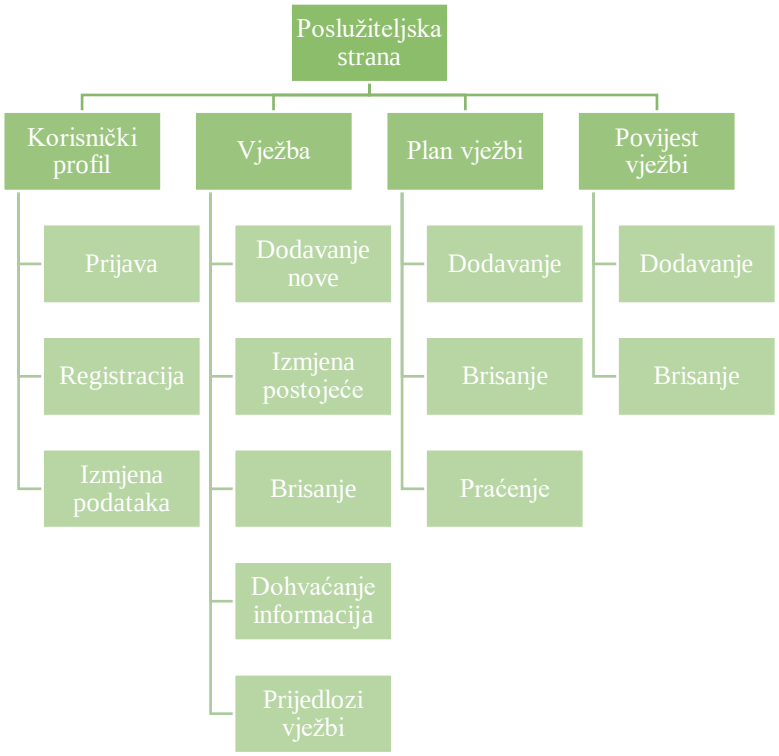


Slika 3.1. *Vizualni prikaz klijentskog dijela aplikacije*

3.2. Poslužiteljska strana

Na poslužiteljskoj strani nalaze se dijelovi aplikacije odgovorni za administraciju baze podataka, pokretanje poslužitelja, validaciju podataka, te izračun i vraćanje korisniku traženih podataka. Poslužiteljska strana napravljena pomoću izvršnog okruženja Node.js, programskog okvira Express.js, te baze podataka MongoDB, prikazana je vizualno na slici 3.2, a sastoji se od nekoliko dijelova:

1. **Upravljanje korisničkim profilom** – Dio koda koji validira unesene podatke, te na temelju tih podataka sprema korisnika u bazu podataka, prijavljuje ga na stranicu, ili izmjenjuje osobne podatke korisnika prema kojima se formiraju prijedlozi vježbi.
2. **Vježba** – Sav kod koji se brine o spremanju nove vježbe u bazu podataka, uređivanju već postojeće, brisanje vježbe, te dohvaćanju informacija o pojedinoj vježbi ili više njih. Također je potrebno onemogućiti običnim korisnicima mogućnost kreiranja, brisanja ili uređivanja vježbe.
3. **Plan vježbe** – Kod koji omogućuje brisanje, dohvaćanje i dodavanje neke određene vježbe u bazu podataka, te analizira napredak korisnika.
4. **Povijest vježbe** – Dio koda koji u bazu podataka registrira vježbe koje je korisnik prethodno posjetio, te omogućuje čišćenje povijesti.



Slika 3.2. *Vizualni prikaz klijentskog dijela aplikacije*

3.3. Opis programskih okruženja i tehnologija

Za potrebe izrade aplikacije korišteni su: programski okvir Vue.js za razvoj korisničkog sučelja, MongoDB kao baza podataka, a sa strane poslužitelja Node.js kao izvršno okruženje, te Express.js kao programski okvir za Node.js. Prije opisa ove četiri temeljne tehnologije koje su se koristile za izradu aplikacije, radi boljeg razumijevanja njihovih funkcija valja definirati nekoliko pojmova:

3.3.1. JSON

Prema [10], JSON **(**engl. *JavaScript Object Notation***)** je jedan od tekstualnih otvorenih standarda dizajniran za ljudima razumljivu razmjenu podataka. Ekstenzija datoteke s podacima u JSON formatu je **.json**, dok je meta oznaka (MIME format) **application/json**. Unutar strukture JSON datoteke mogu se koristiti sljedeći tipovi podataka: broj, niz znakova (engl. *string*), *Boolean* (istina ili laž), polje, objekt i *null*. Struktura JSON datoteke organizirana je u obliku JavaScript objekta sa parovima ključ-vrijednost. Vrlo je lako raditi sa JSON datotekama u JavaScriptu, jer je JSON sintaksa identična JavaScript sintaksi objekata. JSON datoteke se u radu koriste za prikaz informacija o projektu i definiranje različitih ovisnosti (engl. *dependency*), tj. instaliranih paketa i pomoćnih biblioteka koje

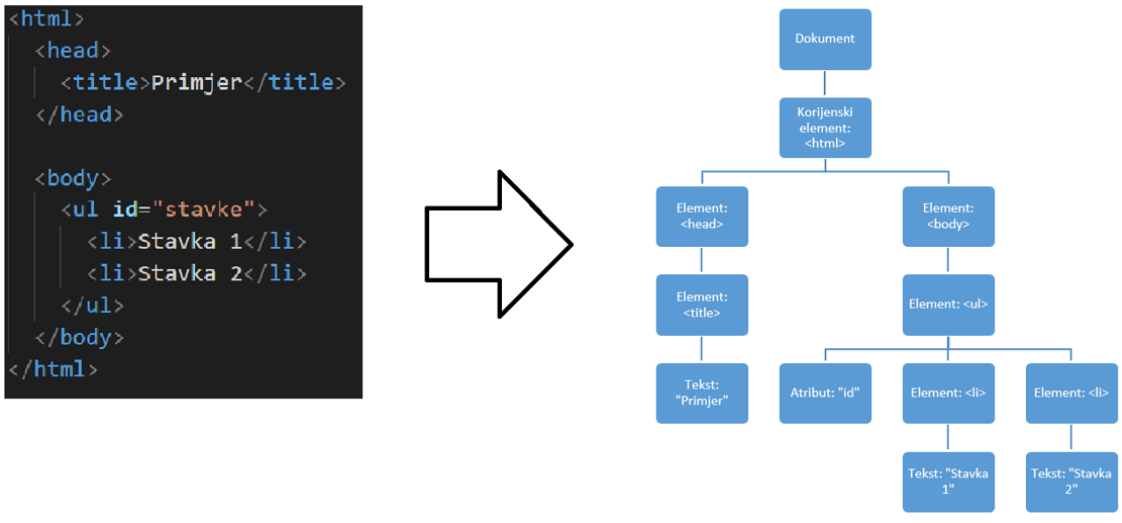
olakšavaju rad, unutar *package.json* datoteke. MongoDB također sprema podatke u bazu podataka u JSON obliku, kao što je prikazano na slici 3.3.

```
{
    // "ključ": "vrijednost"
    "email": "primjerJSON@email.com",
    "password": "zaporka"
}
```

Slika 3.3. *Primjer JSON objekta*

3.3.2. DOM

Prema [11], DOM (engl. *Document Object Model*), ili na hrvatskom, objektni model dokumenta, je način predstavljanja strukturiranog dokumenta preko objekata. To je višeplatformska i o jeziku neovisna konvencija za predstavljanje i interakciju s HTML elementima u dokumentu, gdje su sve HTML oznake predstavljene u obliku podatkovne strukture stabla, kao što je prikazano na slici 3.4. DOM definira objekte i svojstva svih elemenata dokumenta i metode za pristup tim objektima, tj. pruža sučelje preko kojeg se mogu mijenjati čvorovi stabla, te se uglavnom koristi JavaScript jezik za rad s DOM-om. Jedan od glavnih problema DOM-a je to što nije optimiziran sa stvaranje dinamičkog korisničkog sučelja, a taj se problem u programskim okvirima kao što su React.js i Vue.js riješio virtualnim DOM-om.



Slika 3.4. *HTML dokument (lijevo) i njegova DOM predodžba (desno)*

3.3.3. Ovisnosti

Ovisnosti (engl. *dependency*) su pomoćne biblioteke, tj. paketi, instalirani pomoću Node Packet Managera (npm), a proširuju aplikaciju s dodatnim mogućnostima ili olakšavaju pisanje koda.

Primjeri nekih paketa:

axios – služi za slanje HTTP zahtjeva (POST, GET, DELETE, PUT...) poslužitelju iz internetskog preglednika

mongoose – paket koji olakšava rad sa Mongo bazom podataka

eslint – paket koji služi za provjeru sintakse koda

lodash – paket koji dodaje nove funkcije u aplikaciju kojima se olakšava rad sa strukturama podataka u JavaScriptu

body-parser – paket koji služi za pretvaranje tijela zahtjeva u JSON oblik

morgan – paket koji služi za detaljniji ispis zahtjeva prema poslužitelju unutar terminala

nodemon – paket koji služi za automatsko pokretanje poslužitelja na svaku promjenu neke od datoteka

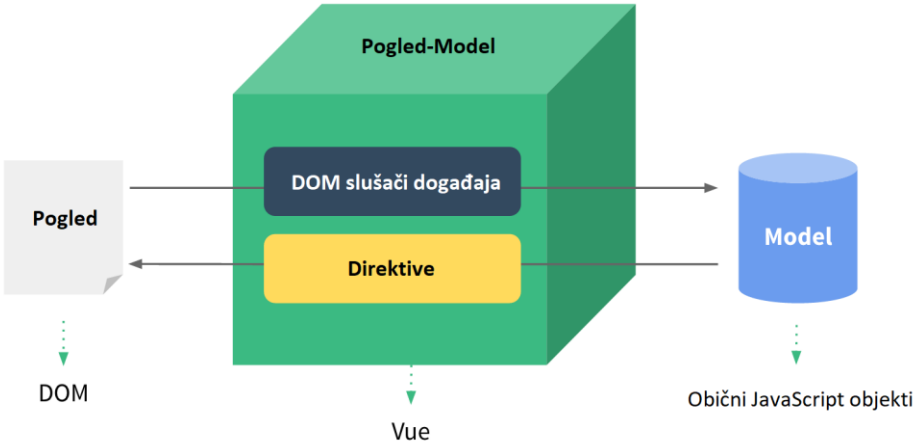
3.3.4. Vue.js

Prema [12], Vue, čiji je logo prikazan na slici 3.5, je progresivni programski okvir za izradu korisničkih sučelja i smatra se jednim od najjednostavnijih JavaScript programskih okvira. Kao takav, vrlo je brz, ima malu veličinu i veoma detaljnu dokumentaciju. Vue.js je fleksibilan i prilagodljiv različitim vrstama projekata.

Slika 3.5. *Logo Vue.js programskog okvira*

Kao što je spomenuto u prethodnom poglavlju, Vue koristi virtualni DOM za manipulaciju elementima dokumenta. Razlog tomu je što su u današnje vrijeme DOM stabla ogromna, te je potrebno modificirati DOM stablo konstantno i više puta, što utječe na performanse web aplikacije. Prema [13, 14], virtualni DOM je apstrakcija HTML DOM-a, tj. njegova kopija. Virtualni DOM izmjenjuje se na željen način, i onda se te promjene spremaju u stvarno DOM stablo. Time se povećava brzina i performanse web aplikacije.

Vue.js djelomično je inspiriran MVVM (engl. *Model-View-ViewModel*) arhitekturom, prikazanoj na slici 3.6, koja omogućuje razdvajanje aplikacije na različite cjeline radi bolje razumljivosti i preglednosti.



Slika 3.6. *Arhitektura Vue-js programskog okvira, prema [15]*

Model je dijelom modificiran običan JS objekt vezan za podatke, a u kontekstu Vue okvira predstavljen je *data* svojstvom. Kada se vrijednosti unutar *data* svojstva promijene, stranica će se ažurirati na način da se pogled podudara s novim vrijednostima unutar *data* svojstva.

Pogled (engl. *View*) je stvarni DOM koji je upravljan Vue instancama. Svaka Vue instanca je povezana s odgovarajućim DOM elementom. Kada je Vue instanca stvorena, ona rekurzivno prolazi kroz DOM stablo do korijenskog elementa, povezivajući pri tome sve potrebne DOM elemente s vrijednostima unutar *data* svojstva, te se na taj način omogućuje dinamičko prikazivanje modela.

Pogled-Model (engl. *ViewModel*) je objekt koji sinkronizira model i pogled. U Vue.js-u, svaka Vue instanca je Pogled-Model, a one su instancirane Vue konstruktorom. Ovo je primarni objekt pomoću kojeg se manipulira podacima iz modela prije nego su prikazani u pogledu.

Jedna od najmoćnijih funkcija Vue.js-a su komponente. Komponente služe za enkapsulaciju koda i podijelu aplikacije na manje dijelove. Jednom definirane i registrirane komponente moguće je koristiti više puta u aplikaciji jednostavnim pozivom preko HTML oznake.

Na slici 3.7 prikazana je tipična struktura Vue datoteke – pod oznakom <*template*> nalazi se HTML kod koji će biti prikazan na stranici. Osim uobičajenih HTML oznaka kao što je <*div*>, postoje i posebne Vue oznake koje imaju prefiks „v-„, a dobivene su instalacijom *Vuetify* paketa preko *npm*-a. Te oznake zapravo predstavljaju Vue komponente kojima se dodaju određena svojstva i CSS klase. Komponente dobivene u *Vuetify* paketu su gumbovi, ladice, tablice, navigacijski paneli, zaglavlje i drugi elementi kojima je moguće urediti korisničko sučelje. Nadalje, unutar <*script*> oznaka nalazi se JavaScript kod u kojemu je moguće pozvati druge pakete, upravljati podacima koji su vezani za Vue komponentu, definirati metode, podatke i druge potrebne stvari koje će se odraziti u pogledu. Unutar <*style*> oznaka definiraju se vlastite CSS klase kojima se mijenja izgled, pozicija ili animacije neke komponente. U ovom konkretnom primjeru, definirana je vlastita komponenta koja se zove „Panel“. Ona je sastavljena od drugih komponenti kao što je npr. <*v-toolbar*>. Nakon definiranja komponente, potrebno ju je registrirati kako bi ju bilo moguće koristiti. Komponentu je moguće registrirati lokalno unutar neke druge komponente ili globalno.

```
<template>
  <div class="white elevation-2">
    <v-toolbar flat dense class="green darken-1" dark>
      <v-toolbar-title>{{title}}</v-toolbar-title>
      <slot name="action" />
    </v-toolbar>
    <div class="pl-4 pr-4 pt-2 pb-2">
      <slot>
        No slot content defined.
      </slot>
    </div>
  </div>
</template>

<script>
export default {
  props: [
    'title'
  ]
}
</script>

<style scoped>
</style>
```

Slika 3.7. *Definiranje Vue komponente pod nazivom "Panel"*

Komponenta se lokalno registrira na način prikazan na slici 3.8: prvo se pomoću ključne riječi *import* unese u željenu komponentu, te se doda u *components* objekt pod istim nazivom. Zatim je moguće tako registriranu komponentu koristiti kao običnu HTML oznaku. Pošto je u Panel komponenti definirano svojstvo *title*, to svojstvo je moguće proslijediti toj komponenti. Osim tog svojstva, unutar *<panel>* oznaka moguće je proslijediti neki sadržaj, u ovom slučaju to je tekstualno polje definirano oznakom *<v-text-field>*. Sve što je unutar *<panel>* oznaka proslijedit će se Panel komponenti, te će doći na mjesto tamošnje *<slot>* oznake.

```
<template>
  <panel title="Traži">
    <v-text-field
      label="Traži po ključnoj riječi"
      v-model="search"
    ></v-text-field>
  </panel>
</template>

<script>
import Panel from '@/components/Panel'
export default {
  data () {…},
  watch: {…},
  components: {
    Panel
  }
}
</script>

<style scoped>
</style>
```

Slika 3.8. *Lokalna registracija komponente pod nazivom Panel unutar komponente Pretraga*

Globalna registracija (Slika 3.9) je gotovo identična: unutar glavne, *main.js* datoteke, jednostavno se preko ključne riječi *import* unese komponenta, te se koristi metoda *component* na Vue klasi. Time je tu komponentu moguće koristiti bilo gdje u aplikaciji.

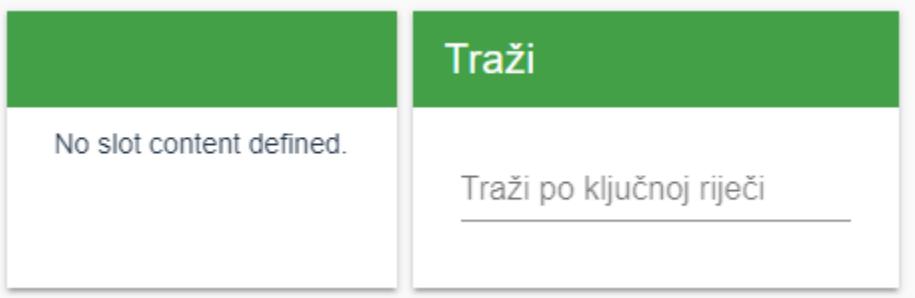
```
import Panel from '@/components/globals/Panel'
import VueCountdown from '@xkeshi/vue-countdown'

Vue.component('panel', Panel)
Vue.component('countdown', VueCountdown)

new Vue({
  el: '#app',
  router,
  store,
  components: { App },
  template: '<App/>'
})
```

Slika 3.9. *Globalna registracija komponenti Panel i VueCountdown*

Na slici 3.10 su prikazane prazna Panel komponenta i komponenta Pretraga koja koristi Panel komponentu.



Slika 3.10. *Komponenta Panel (lijevo) i komponenta Pretraga (desno)*

3.3.5. MongoDB

Prema [16], MongoDB, čiji je logo prikazan na slici 3.11, je baza podataka otvorenog koda koja se temelji na konceptu zbirki i dokumenata. Mongo je jedna od vodećih NoSQL baza podataka, te osigurava visoke performanse, visoku dostupnost, skalabilnost i fleksibilnost. Dokumenti pohranjeni u bazi podataka najčešće su JSON oblika. Svaki dokument se sastoji od barem jednog atributa, čije vrijednosti mogu biti broj, znakovni niz, *Boolean*, objekt, polje, datum, regularni izraz, *null* ili neka korisnički definirana struktura ili tip podataka, a dokumenti u sustavu pohranjuju se u nekoj od zbirki. Zbirka je grupa dokumenata sa zajedničkim svojstvima.

Slika 3.11. *Logo MongoDB baze podataka*

MongoDB, za razliku od SQL baza podataka, ne mora imati definiranu shemu, što znači da svaki dokument može imati različite podatke, te ne moraju sva polja biti popunjena u nekom dokumentu. Opći pregled osnovnih razlika između Mongo baze podataka i NoSQL baze podataka dan je u tablici 3.1.

Tablica 3.1. *Usporedba relacijskih i NoSQL baza podataka*

Relacijske baze podataka	**NoSQL baze podataka**
Baza podataka	Baza podataka
Tablica	Zbirka
Redak	Dokument
Stupac	Polje
Spajanje tablica	Ugrađeni dokument
Primarni ključ	Primarni ključ (predodređen, automatski dodano polje _id)

Radi proširenja mogućnosti Mongo baze podataka, preko *npm*-a instaliran je paket *Mongoose*. Prema [17], *Mongoose* pruža jednostavno razumljivo rješenje u obliku *scheme* za modeliranje podataka za aplikacije koje koriste Node.js. Uključuje ugrađeno mijenjanje tipova podataka (engl. *casting*), validaciju, pisanje upita, spremanje i ažuriranje podataka i brojne druge mogućnosti.

Na slici 3.12 prikazan je primjer modeliranja zbirke *Product* preko *scheme* koristeću *Mongoose*. U *schemi* je definirano nekoliko atributa različitih tipova, s time da je atributima moguće dodati neka ograničenja ili posebne vrijednosti, kao što je slučaj kod *outOfStock* atributa, koji će biti postavljen na *false* ako drukčije ne bude specificirano prilikom izrade dokumenta u toj zbirci.

```
const mongoose = require('mongoose')

const productSchema = new mongoose.Schema({
  name: String,
  price: Number,
  description: String,
  tags: [String],
  expiresAt: Date,
  outOfStock: {
    type: Boolean,
    default: false
  }
})

const Product = mongoose.model('Product', productSchema)
module.exports.Product = Product
```

Slika 3.12. *Primjer modeliranja modela proizvoda koristeći Mongoose*

Osim pohrane podataka, moguće je i obavljati pretragu nad spremljenim podatcima, tj. moguće je vršiti upite. Upiti služe za dohvaćanje nekog dokumenta iz određene zbirke prema proizvoljnim kriterijima. Upite je moguće obavljati pomoću raznih funkcija koje nudi MongoDB (*find, findById*), kojima je potrebno proslijediti željene parametre pretrage, te ih po potrebi proširiti sa dodatnim opcijama. Primjer jednog upita nad zbirkom *Product* prikazan je na slici 3.13, koji će pronaći sve proizvode čije je ime „TV“, cijena između 300 i 800, ograničiti rezultat upita na 5 rezultata koji će biti poredani od najveće do najmanje cijene. Osim toga, koristi se projekcija *select* naredbom kako bi se izabrala samo željena polja.

```
Product.
  find().
  where('name').equals('TV').
  where('price').gt(300).lt(800).
  limit(5).
  sort({ price: -1 }).
  select('name price')
```

Slika 3.13. *Primjer upita nad zbirkom Product*

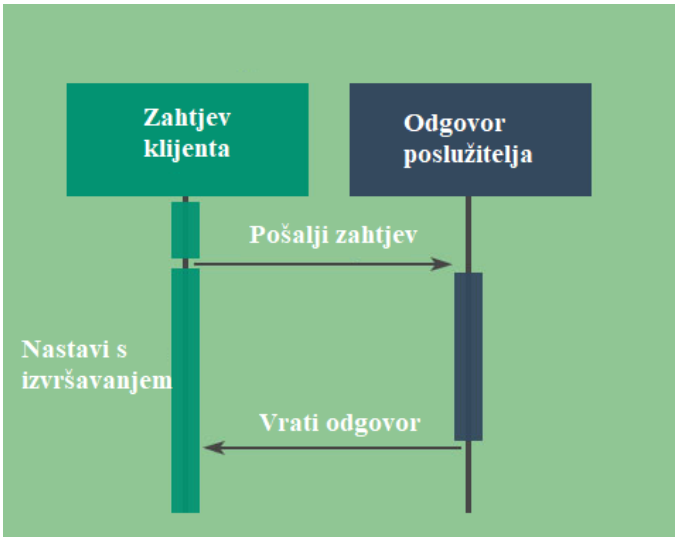
3.3.6. Node.js i Express.js

Prema [18, 19], Node.js, čiji je logo prikazan na slici 3.14, je JavaScript izvršno okruženje na strani poslužitelja, i koristi Chrome V8 JavaScript *engine*, čime se osiguravaju visoke brzine i dobre performanse. Node.js-a je prilično jednostavan: glavne su funkcionalnosti svedene na minimum i kompleksnost koja dolazi pri pisanju koda zadržana je na što nižoj razini.

Slika 3.14. *Logo Node.js izvršnog okruženja*

Node.js je poseban po tome što koristi asinkron način programiranja. U nekim slučajevima, sinkroni način programiranja zahtjeva dugo čekanje, npr. u slučajevima dohvaćanja podataka iz baze podataka, te se time usporava aplikacija. Taj problem riješen je uvođenjem pojma niti izvršavanja. Dok se u jezicima poput Jave ili C# može pojaviti više niti koje se izvršavaju u isto vrijeme (te može doći do

zastoja u izvršavanju programa, npr. u slučaju da više niti konkurentno pristupaju istom resursu), kod Node.js-a uvijek se izvršava samo jedna nit. Ako se izvršavaju neke sporije operacije na koje treba pričekati određeno vrijeme na rezultat, program ne čeka, nego izvršava sljedeću liniju koda, kao što je vizualno prikazano slikom 3.15.



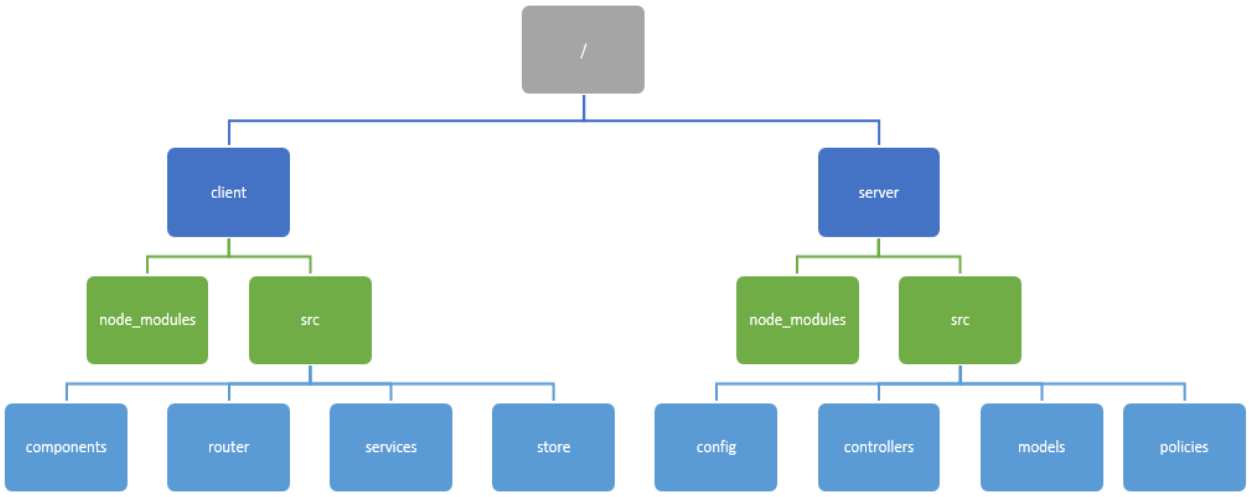
Slika 3.15. *Asinkroni način izvršavanja koda u Node.js-u, prema [20]*

Kada se rezultat takve spore operacije vrati, program će dobiti tu informaciju i imati pristup dobivenom rezultatu. Za upravljanje asinkronim funkcijama u JavaScriptu postoji nekoliko različitih pristupa: *callback* funkcije, *Promises* i ključne riječi *async/await*.

Još jedna od velikih prednosti Node.js-a je već spomenuti *Node Packet Manager* – alat pomoću kojeg je moguće pretražiti preko 600,000 paketa otvorenog koda koji se mogu iskoristiti u Node.js aplikaciji. Jedan od najpopularnijih *npm* paketa jest programski okvir Express.js, koji je standard za većinu Node aplikacija. Express pruža veliku zbirku pomoćnih funkcija i ostalih značajki koje uvelike olakšavaju rad.

3.4. Prikaz implementacije vlastitog rješenja

Radi lakšeg snalaženja tijekom izrade, kod aplikacije je organiziran na način prikazan na slici 3.16. Posebno je odvojen dio koda za korisničko sučelje od dijela koda na strani poslužitelja. Nadalje, u svakom dijelu kod je podijeljen u podmape koje su logički povezane u jednu cjelinu.



Slika 3.16. *Vizualni prikaz organizacije koda aplikacije*

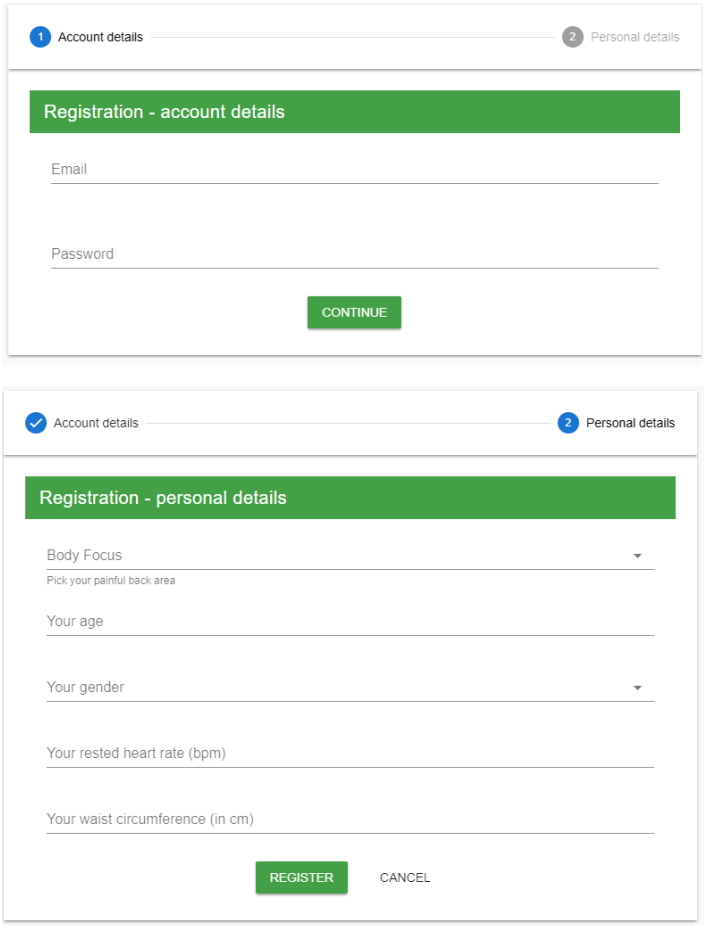
U tablici 3.2 dan je kratak opis svih važnijih mapa i podmapa koje čine cjelokupnu aplikaciju.

Tablica 3.2. *Objašnjenje strukture aplikacije*

DIREKTORIJ	OBJAŠNJENJE
/client/src	Mapa u kojoj se nalazi kod korisničkog sučelja aplikacije
/client/src/components	Mapa u kojoj se nalaze sve Vue.js komponente
/client/src/router	Mapa u kojoj su definirani svi URL-ovi aplikacije
/client/src/services	Mapa u kojoj se nalaze datoteke koje šalju potrebne HTTP metode poslužitelju
/client/src/store	Mapa koja sadrži stanje aplikacije (podaci o korisniku, informaciju o stanju prijave korisnika)
/server/src	Mapa u kojoj se nalaze kod vezan za poslužiteljsku stranu
/server/config	Mapa u kojoj se nalaze konfiguracijske datoteke vezane za aplikaciju
/server/controllers	Mapa u kojoj se nalazi kod odgovoran za komunikaciju između korisničkog sučelja i poslužitelja
/server/models	Mapa u kojoj se nalaze modeli potrebnih zbirki za rad s bazom podataka
/server/policies	Mapa u kojoj su definirana pravila za ovjeru korisnika

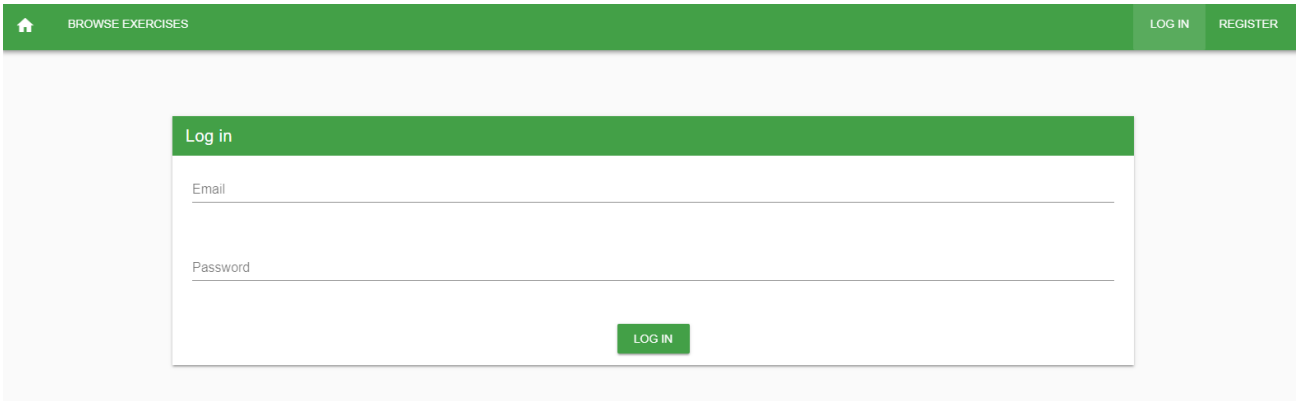
3.4.1. Korisnički profil

U nastavku je prikazan način registracije i prijave korisnika, te izmjene podataka korisnika. Na slici 3.17 prikazano je korisničko sučelje za registraciju. Ono je podijeljeno u dva koraka koristeći komponentu *v-stepper*. Prvi korak je unošenje e-maila i lozinke, a drugi korak je unošenje osobnih podataka kao što je popis dijelova tijela koja zahtjevaju terapiju, dob, spol, te broj otkucaja srca u mirovanju u minuti i opseg struka. Na temelju tih parametara izračunat će se maksimalni volumen kisika (VO_{2max}), te će se prema tom izračunatom volumenu i unesenim dijelovima tijela prikazati prijedlozi vježbi.



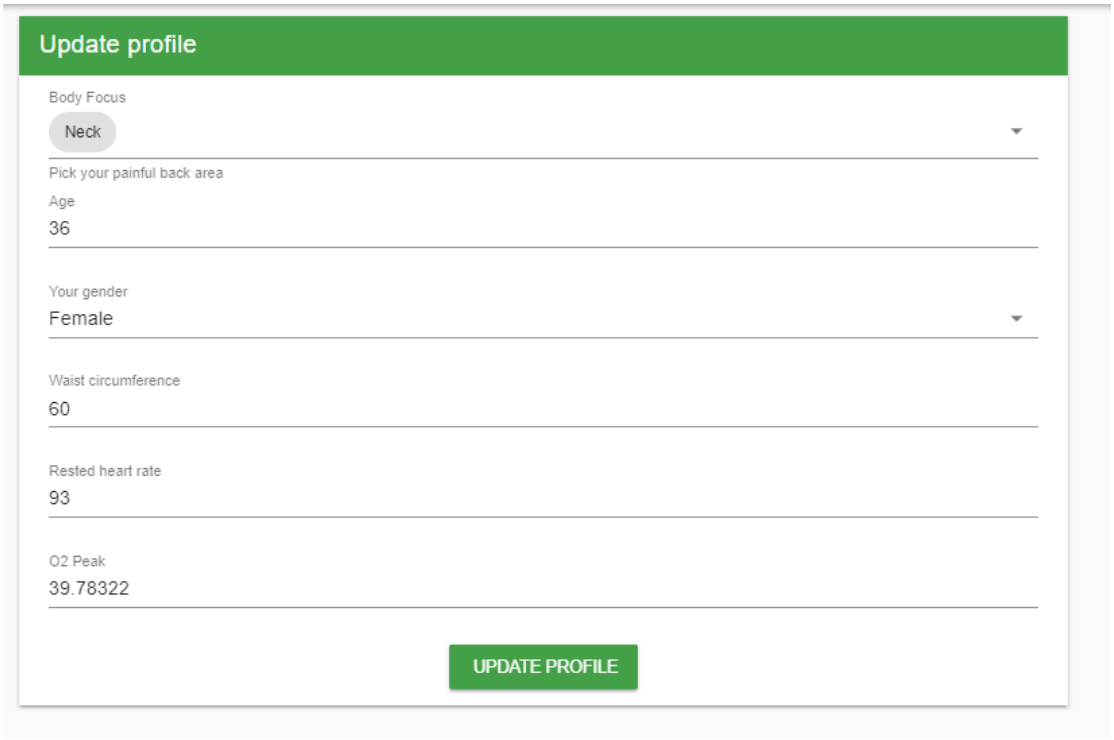
Slika 3.17. *Sučelje za registraciju novog korisnika*

Sučelje za prijavu (Slika 3.18) je gotovo identično prvom koraku pri registraciji, osim minimalnih promjena u izgledu, no kroz sučelje za prijavu šalju se drukčiji HTTP zahtjevi prema poslužitelju.



Slika 3.18. *Sučelje za prijavu korisnika*

Sučelje za izmjenu podataka, tj. ažuriranje profila vidljivo je na slici 3.19. Pri otvaranju tog sučelja, dohvaćaju se podaci iz baze podataka o trenutnom korisniku, te se popunjavaju polja koja korisnik može izmijeniti. Korisnik ne može mijenjati polje „*O2 Peak*“, jer je tom polju postavljeno *read-only* svojstvo. Klikom na gumb „*Update profile*“ šalje se PUT zahtjev prema poslužitelju, ažuriraju se korisnikovi podaci, te se korisnik preusmjeri na stranicu s vježbama.



Slika 3.19. *Sučelje za ažuriranje korisnikovih podataka*

Na slici 3.20 nalazi se HTML kod (lijevo) koji je povezan sa JavaScript kodom (desno), te zajedno čine cijelo sučelje za registraciju korisnika. Koristi se *Vuetify* komponenta *v-stepper* koja se sastoji od zaglavlja (*v-stepper-header*) u kojima definiramo naslove svakog od koraka, a sadržaj svakog koraka definiramo unutar *v-stepper-content* oznake.

```
<template>
  <v-flex xs6 offset-xs-3>
    <v-stepper v-model="e1">
      <v-stepper-header>···
      </v-stepper-header>
      <v-stepper-items>
        <v-stepper-content step="1">
          <panel title="Registration - account details">
            <form···
            </form>
            <v-btn···
            >
              Continue
            </v-btn>
          </panel>
        </v-stepper-content>
        <v-stepper-content step="2">
          <panel title="Registration - personal details">
            <v-select···
            ></v-select>
            <v-text-field
              label="Your age"
              v-model="age"
            ></v-text-field>
            <v-select···
            ></v-select>
            <v-text-field···
            ></v-text-field>
            <v-text-field···
            ></v-text-field>
            <v-btn
              mt-5
              dark
              color="green darken-1"
              @click="register">
              Register
            </v-btn>
            <v-btn···
            </v-btn>
          </panel>
          <v-alert :value="!!this.error" type="error">
            <div v-html="error" />
          </v-alert>
        </v-stepper-content>
      </v-stepper-items>
    </v-stepper>
  </v-flex>
</template>
```

```
import UserProfileService from '@/services/UserProfileService'
import ExercisesService from '@/services/ExercisesService'

export default {
  data () {
    return {
      email: '',
      password: '',
      age: null,
      gender: '',
      waistCircumference: null,
      restedHeartRate: null,
      error: null,
      e1: 0,
      genderSelect: ['Male', 'Female'],
      bodyFocus: null,
      bodyFocusSelect: []
    }
  },
  methods: {
    async register () {
      try {···
      } catch (error) {···
      }
    }
  },
  async mounted () {
    let exercises = (await ExercisesService.index()).data
    for (let exercise of exercises) {
      this.bodyFocusSelect.push(exercise.bodyFocus)
    }
  }
}
</script>
```

Slika 3.20. *Kod sučelja za registraciju korisnika*

Koristi se *v-model* Vue direktivu kako bi se povezao tekst koji korisnik unese u tekstualni okvir na sučelju sa podacima koji se nalaze u *data* funkciji unutar <*script*> oznake, kao što je prikazano kod tekstualnog unosa dobi. Koristimo asinkronu *register* metodu koja se aktivira pritiskom na gumb „*Register*“, a zajedno su povezani *@click* direktivom. *Register* metoda će pokušati pozvati drugu *register* metodu koja se nalazi unutar *UserProfileService* datoteke, te joj predati parametre koje je unio korisnik. Zatim će korisniku postaviti JWT žeton kojim će se šifrirati zaporka, i stanje korisnika,

što će omogućiti pristup nekim drugim svojstvima korisnika, te tako olakšati komunikaciju između sučelja i poslužitelja. Nakon postavljanja te dvije stavke, korisnik će biti preusmjeren na drugu stranicu definirana kao *exercises*. U slučaju da je registracija korisnika bila neuspješna, korisniku će se prikazati odgovarajuća greška definirana na strani poslužitelja.

Komunikacija sa stranom poslužitelja ostvarena je pomoću npm paketa *axios*. Na slici 3.21 na desnoj strani prikazan je način na koji je to ostvareno.

```
import Api from '@/services/Api'

export default {
  register (credentials) {
    return Api().post('register', credentials)
  },
  login (credentials) {
    return Api().post('login', credentials)
  },
  update (profileInfo) {
    return Api().put('update-profile', profileInfo)
  },
  show () {
    return Api().get('profile')
  }
}
```

```
import axios from 'axios'
import store from '@/store/store'

export default () => {
  return axios.create({
    baseURL: `http://localhost:8081/`,
    headers: {
      Authorization: `Bearer ${store.state.token}`
    }
  })
}
```

Slika 3.21. *Kod datoteka UserProfileService (lijevo) i Api (desno)*

Pomoću *create* metode *axios* instanci predaje se *baseURL* parametar u kojem se nalazi URL i pristup na kojem je pokrenut poslužitelj, te, u *headers* objektu, podatci vezani za JWT žeton koji se koristi za ovjeru lozinke prilikom prijave korisnika. Lijevo je taj API iskorišten na način da se *register* metodom, koja kao parametar prima korisničke podatke, pošalje POST zahtjev poslužitelju, zajedno sa predanim parametrima.

Zahtjevi primljeni na strani poslužitelja evaluiraju se u *routes.js* datoteci prikazanoj na slici 3.22, te se pozivaju odgovarajuće metode definirane u drugim datotekama.

```
const UserProfileController = require('./controllers/UserProfileController')
const UserProfileControllerPolicy = require('./policies/UserProfileControllerPolicy')
const ExercisesController = require('./controllers/ExercisesController')
const ExercisesPlanController = require('./controllers/ExercisesPlanController')
const HistoriesController = require('./controllers/HistoriesController')
const ExercisesDoneController = require('./controllers/ExercisesDoneController')
const isAuthenticated = require('./policies/isAuthenticated')

module.exports = (app) => {
  app.post('/register',
    UserProfileControllerPolicy.register,
    UserProfileController.register)

  app.post('/login',
    UserProfileController.login)

  app.get('/profile',
    isAuthenticated,
    UserProfileController.show)

  app.put('/update-profile',
    isAuthenticated,
    UserProfileControllerPolicy.update,
    UserProfileController.update)

  app.get('/exercises',
    ExercisesController.index)

  app.get('/exercises/:exerciseId',
    ExercisesController.show)

  app.put('/exercises/:exerciseId',
    isAuthenticated,
    ExercisesController.put)

  app.post('/exercises',
    isAuthenticated,
    ExercisesController.post)

  app.get('/plan',
    isAuthenticated,
    ExercisesPlanController.index)

  app.put('/exercises/:exerciseId/add',
    isAuthenticated,
    ExercisesPlanController.add)
```

Slika 3.22. *Prikaz nekih od ruta dostupne korisniku u routes.js datoteci*

Konkretno u slučaju prijave ili registracije, predani POST zahtjev evaluirat će se pomoću datoteka *UserProfileControllerPolicy* i *UserProfileController* u kojima su definirane *register*, *login* i *update* metode koje će provjeriti parametre poslane u zahtjevu i registrirati korisnika, prijaviti ga u sustav, ili mu ažurirati podatke, ovisno o željenoj radnji, ili će izbaciti grešku.

U datoteci *UserProfileControllerPolicy* definirana su pravila koja vrijede prilikom registracije i ažuriranja profila, te se vrši validacija podataka nad predanim parametrima pomoću *npm* paketa *Joi*. Validacija pomoću tog paketa vrši se na način da se definira *schema* prema kojoj će se uspoređivati dobiveni parametri iz sučelja. Na slici 3.23 prikazan je kod te datoteke za slučaj registracije, te je vidljivo da je u *schemi* definirano nekoliko parametra koja imaju pravila: *email*, *password, age, gender, restedHeartRate, waistCircumference* i *bodyFocus.*

```
const Joi = require('joi')

module.exports = {
  register (req, res, next) {
    const schema = {
      email: Joi.string().email(),
      password: Joi.string().regex(
        new RegExp('^[a-zA-z0-9]{8,32}$')
      ),
      age: Joi.number().integer().min(10).max(75),
      gender: Joi.string(),
      restedHeartRate: Joi.number().integer().min(35).max(140),
      waistCircumference: Joi.number().integer(),
      bodyFocus: Joi.array()
    }
    const {error, value} = Joi.validate(req.body, schema)
    if (error) {
      console.log('error', error)
      switch (error.details[0].context.key) {
        case 'email':
          res.status(400).send({
            error: 'You must enter a valid email address.'
          })
          break
        case 'password':
          res.status(400).send({
            error: `Provided password must obide following rules:
            <br>
            1. It can contain only capital and non-capital letters, and numbers
            <br>
            2. It must be at least 8 characters in length and less than 32 characters in length
            `
          })
          break
        case 'age':
          res.status(400).send({
            error: `Entered age must be an integer between 10 and 75.`
          })
          break
```

Slika 3.23. *Datoteka UserProfileControllerPolicy*

E-mail mora biti niz znakova, te mora biti e-mail oblika (npr. primjer@email.com), dok lozinka također mora biti string, te mora odgovarati regularnom izrazu u kojem je definirano da lozinka mora biti niz od barem 8, a maksimalno 32 alfanumerička znaka. Svaki od ostalih parametara ima svoja određena pravila. Ukoliko dođe do greške prilikom validacije, poslužitelj šalje odgovarajuće HTTP kodove i objekt sa ključem *error* u kojem je definirana greška koja će se prikazati na korisničkom sučelju. *Register* metoda unutar ove datoteke je posredna (engl. *middleware*) metoda, što znači da je ona jedna u nizu metoda koja se izvršava. Ukoliko se validacija podataka ispostavi točnom, pozvat će se sljedeća u nizu metoda, što vidimo u *else* bloku pozivom *next* metode.

A sljedeća u nizu metoda tijekom registracije je *register* metoda koja se nalazi unutar *UserProfileController* datoteke (Slika 3.24), koja će provjeriti unesene parametre na temelju kojih će se izračunati maksimalni volumen kisika, i nakon toga korisniku se dodjeljuje JWT žeton i sprema ga se u bazu podataka ili se vraća greška kao odgovor.

```
const {User} = require('../models/User')
const jwt = require('jsonwebtoken')
const config = require('../config/config')

function jwtSignUser (user) {
  const ONE_WEEK = 60 * 60 * 24 * 7
  return jwt.sign(user, config.authentication.jwtSecret, {
    expiresIn: ONE_WEEK
  })
}
module.exports = {
  async register (req, res) {
    const adminCode = req.body.adminCode
    let isAnAdmin = false
    let PA_INDEX = req.body.gender === 'Male' ? 0.44 : 0.39
    let O2Peak = null

    if(req.body.gender === 'Male') {
      O2Peak = 100.27 - (0.296 * req.body.age) - (0.369 * req.body.waistCircumference) - (0.155 * req.body.restedHeartRate) + (0.226 * PA_INDEX)
    } else {
      O2Peak = 74.74 - (0.247 * req.body.age) - (0.259 * req.body.waistCircumference) - (0.114 * req.body.restedHeartRate) + (0.198 * PA_INDEX)
    }

    if(adminCode === config.authentication.adminSecret) {
      isAnAdmin = true
    }

    let user = new User({···
    })
    try {
      user.save(function (error, user) {
        if (error) {
          res.status(400).send({
            error: `This email address is already in use.`
          })
        } else {
          const userJson = user.toJSON()
          res.send({
            user: userJson,
            token: jwtSignUser(userJson)
          })
        }
      })
    } catch (error) {
      res.status(400).send({
        error: `This email address is already in use.`
      })}
  },
```

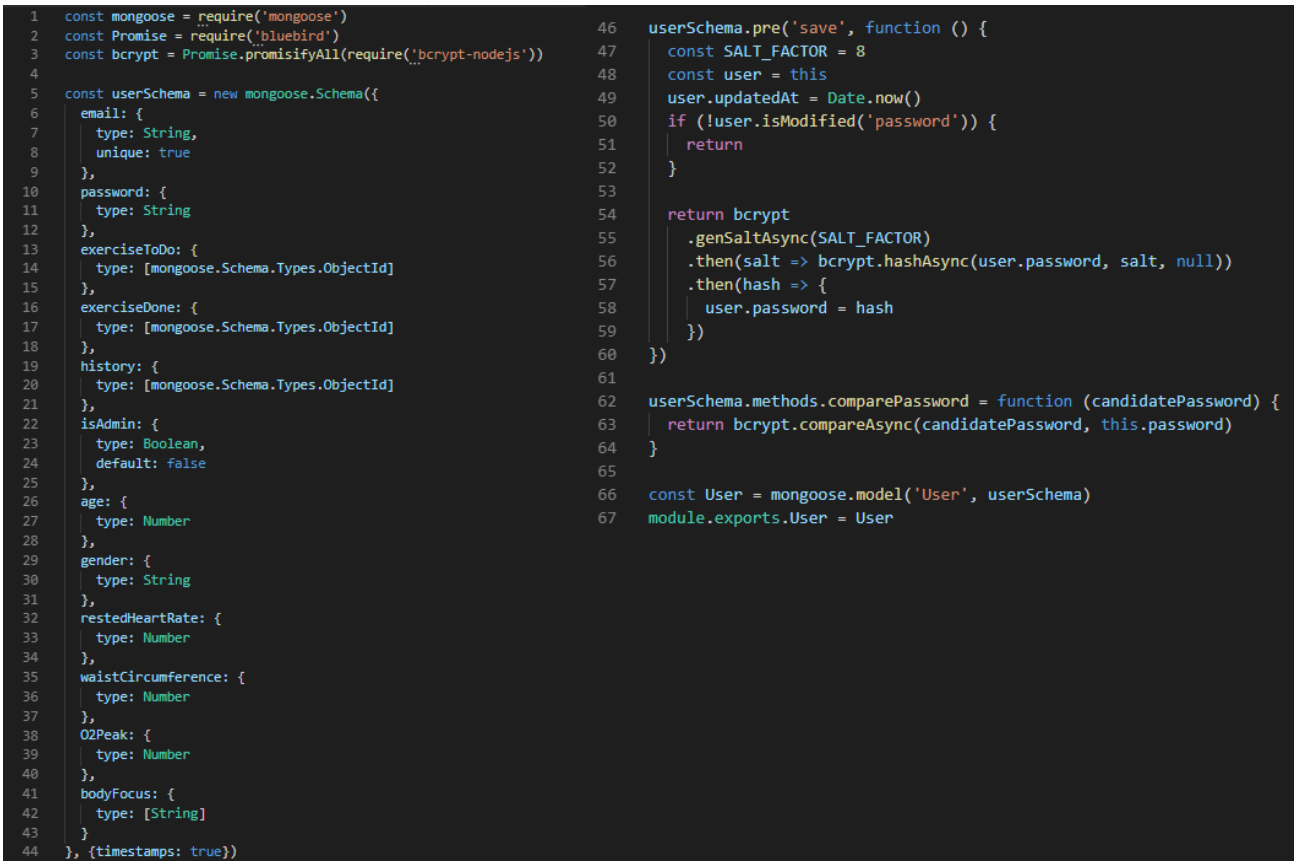
Slika 3.24. *Datoteka UserProfileController s metodom za registraciju*

Ovdje se koristi model korisnika izrađen pomoću *Mongoose* paketa kojeg se sprema u Mongo bazu podataka. Metoda za prijavu prikazana na slici 3.25 naći će jednog korisnika čiji e-mail odgovara e-mail-u spremljenim u bazi podataka, usporedit će unesenu lozinku sa šifriranom lozinkom u bazi podataka pomoću *comparePassword* metode koja se nalazi u modelu korisnika, te će kao odgovor vratiti ili grešku ili traženog korisnika.

```
57  async login (req, res) {
58    try {
59      const {email, password} = req.body
60      let user = await User.findOne({
61        email: email
62      })
63      if (!user) {
64        return res.status(403).send({
65          error: 'Invalid login data.'
66        })
67      }
68      const isPasswordValid = await user.comparePassword(password)
69      if (!isPasswordValid) {
70        return res.status(403).send({
71          error: 'Invalid login data.'
72        })
73      }
74      const userJson = user.toJSON()
75      res.send({
76        user: userJson,
77        token: jwtSignUser(userJson)
78      })
79    } catch (err) {
80      res.status(500).send({
81        error: `An error has occurred while authenticating the user.`
82      })
83    }
84  },
```

Slika 3.25. *Datoteka UserProfileController sa metodom za prijavu*

Na slici 3.26 prikazan je model korisnika u dijelu koda na lijevoj polovici. Vidljivo je da korisnik, osim e-mail-a i lozinke, ima još nekoliko svojstava vezanih uz vježbe, te *Boolean* varijablu *isAdmin* koja definira je li korisnik administrator, tj. ima li ovlasti da postavlja i mijenja vježbe.



```
const mongoose = require('mongoose')
const Promise = require('bluebird')
const bcrypt = Promise.promisifyAll(require('bcrypt-nodejs'))

const userSchema = new mongoose.Schema({
  email: {
    type: String,
    unique: true
  },
  password: {
    type: String
  },
  exerciseToDo: {
    type: [mongoose.Schema.Types.ObjectId]
  },
  exerciseDone: {
    type: [mongoose.Schema.Types.ObjectId]
  },
  history: {
    type: [mongoose.Schema.Types.ObjectId]
  },
  isAdmin: {
    type: Boolean,
    default: false
  },
  age: {
    type: Number
  },
  gender: {
    type: String
  },
  restedHeartRate: {
    type: Number
  },
  waistCircumference: {
    type: Number
  },
  O2Peak: {
    type: Number
  },
  bodyFocus: {
    type: [String]
  }
}, {timestamps: true})
```

```
userSchema.pre('save', function () {
  const SALT_FACTOR = 8
  const user = this
  user.updatedAt = Date.now()
  if (!user.isModified('password')) {
    return
  }

  return bcrypt
    .genSaltAsync(SALT_FACTOR)
    .then(salt => bcrypt.hashAsync(user.password, salt, null))
    .then(hash => {
      user.password = hash
    })
})

userSchema.methods.comparePassword = function (candidatePassword) {
  return bcrypt.compareAsync(candidatePassword, this.password)
}

const User = mongoose.model('User', userSchema)
module.exports.User = User
```

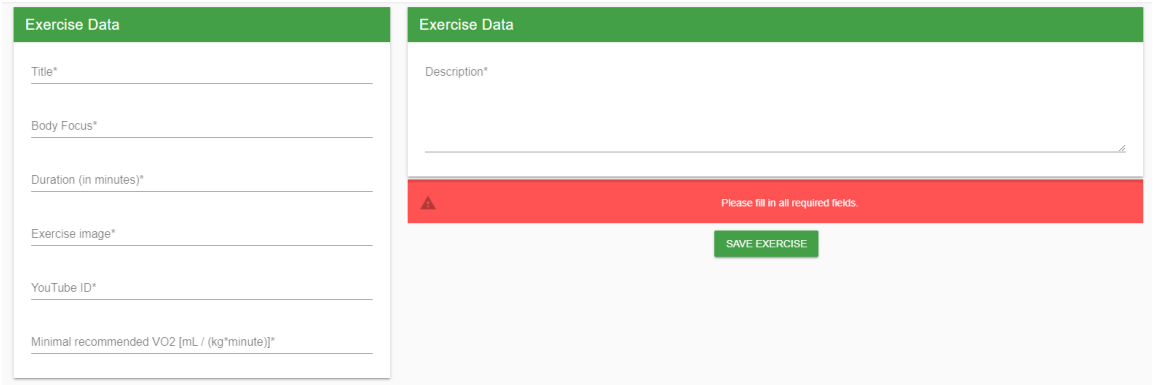
Slika 3.26. *Datoteka User.js sa definiranim modelom (lijevo) i metodama korisnika (desno)*

Osim modela korisnika, definirana je funkcija koja se aktivira prije svake promjene podataka. Ta metoda koristi *bcrypt* paket za šifriranje lozinke, te će se u slučaju promjene lozinke postojećeg, ili pri stvaranju novog korisnika njegova lozinka šifrirati i takva biti spremljena u bazu podataka. Osim šifriranja lozinka, definirana je i metoda *comparePassword* koja opet koristi *bcrypt* za usporedbu unesene lozinke sa onom šifriranom u bazi podataka.

3.4.2. Dodavanje, izmjena, dohvaćanje vježbi i dohvaćanje prijedloga

Na slici 3.27 prikazano je sučelje za dodavanje nove vježbe, a na slici 3.28 pripadni JavaScript kod. Za stvaranje nove vježbe potrebno je ispuniti sva označena polja. Ukoliko korisnik klikne na gumb „*Save exercise*“, aktivirat će se metoda *create,* koja će provjeriti jesu li tražena tekstualna polja ispunjena, te će, ako nisu, izbaciti grešku kao što je prikazano na 3.27, a ako jesu, pozvat će se *post*

metoda definirana u *ExercisesService* datoteci zajedno sa parametrima vježbe. Po uspješnom spremanju vježbe, korisnik će biti preusmjeren na *exercises* stranicu.



Slika 3.27. *Sučelje za dodavanje nove vježbe*

```
<script>
import ExercisesService from '@/services/ExercisesService'
export default {
  data () {
    return {
      exercise: {
        title: null,
        bodyFocus: null,
        duration: null,
        exerciseImgUrl: null,
        youtubeId: null,
        description: null,
        minO2: null
      },
      error: null,
      required: (value) => !!value || 'Field is required.'
    }
  },
  methods: {
    async create () {
      this.error = null
      const areAllFieldsFilledIn = Object
        .keys(this.exercise)
        .every(key => !!this.exercise[key])
      if (!areAllFieldsFilledIn) {
        this.error = 'Please fill in all required fields.'
        return
      }
      try {
        await ExercisesService.post(this.exercise)
        this.$router.push({
          name: 'exercises'
        })
      } catch (err) {
        console.log(err)
      }
    }
  }
}
</script>
```

Slika 3.28. *Kod sučelja za dodavanje vježbe*

Na slici 3.29 prikazan je JavaScript kod sučelja za uređivanje vježbe. Pritiskom na gumb „*Save exercise*“ pozove se *save* metoda koja će provjeriti jesu li popunjena zahtjevana polja, te će se, ako je sve u redu, pozvati *put* metoda iz *ExercisesService* datoteke i korisnika će se preusmjeriti na *exercises* stranicu.

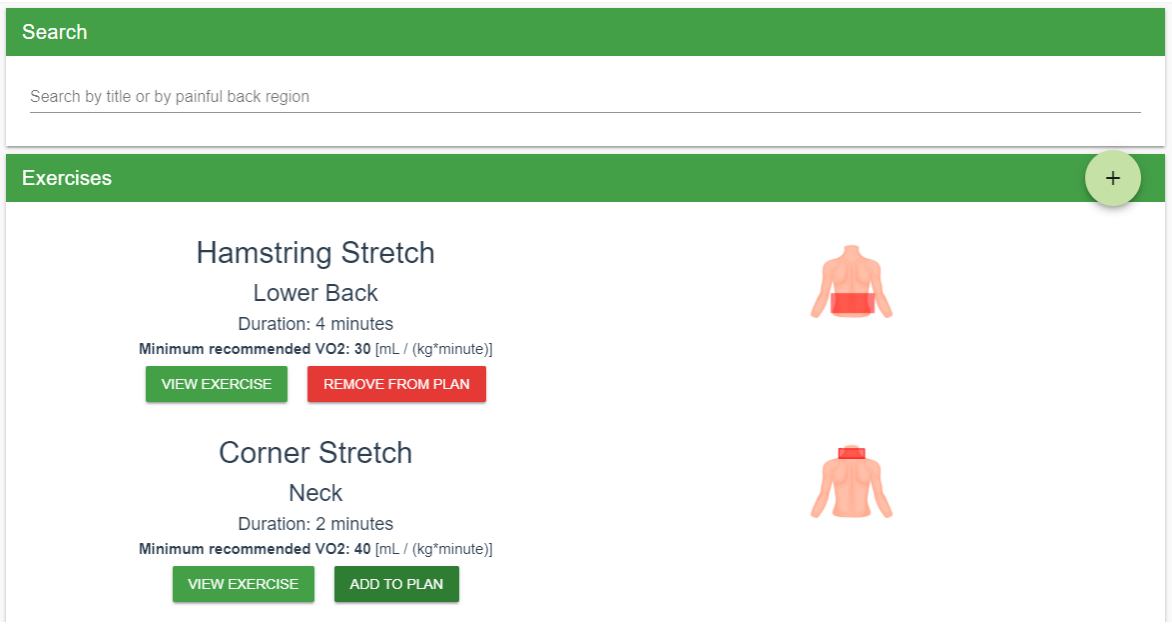
```
92    methods: {
93      async save () {
94        this.error = null
95        let subsetExercise = _.pick(this.exercise, ['title', 'bodyFocus',
96          'duration', 'exerciseImgUrl', 'youtubeId', 'description', 'minO2'])
97        const areAllFieldsFilledIn = Object
98          .keys(subsetExercise)
99          .every(key => !!subsetExercise[key])
100       if (!areAllFieldsFilledIn) {
101         this.error = 'Please fill in all required fields.'
102         return
103       }
104       const exerciseId = this.$store.state.route.params.exerciseId
105       try {
106         await ExercisesService.put(this.exercise)
107         this.$router.push({
108           name: 'exercise',
109           params: {
110             exerciseId: exerciseId
111           }
112         })
113       } catch (err) {
114         console.log(err)
115       }
116     }
117   },
```

Slika 3.29. *Kod sučelja za uređivanje vježbe*

Predani zahtjev se, kao i kod izmjena parametra korisnika ili registracije novog, preda *axios* instanci koja će pozvati potrebne metode definirane na strani poslužitelja. U *ExercisesController* datoteci prikazanoj na slici 3.30 definirano je nekoliko metoda koje služe za dohvaćanje, izmjenu, prikaz i pretragu vježbi. Kod *post* metode se jednostavno stvori nova vježba prema parametrima dobivenim iz korisničkog sučelja, te se spremi u bazu podataka. *Index* metoda služi za prikaz više vježbi, kao što je prikazano na slici 3.31. U slučaju da ne postoje nikakvi parametri prema kojima se pretražuju vježbe, naći će se njih 10 i poslati kao odgovor, a u slučaju da postoje parametri pretrage, poslat će se one vježbe koje odgovaraju tim parametrima.

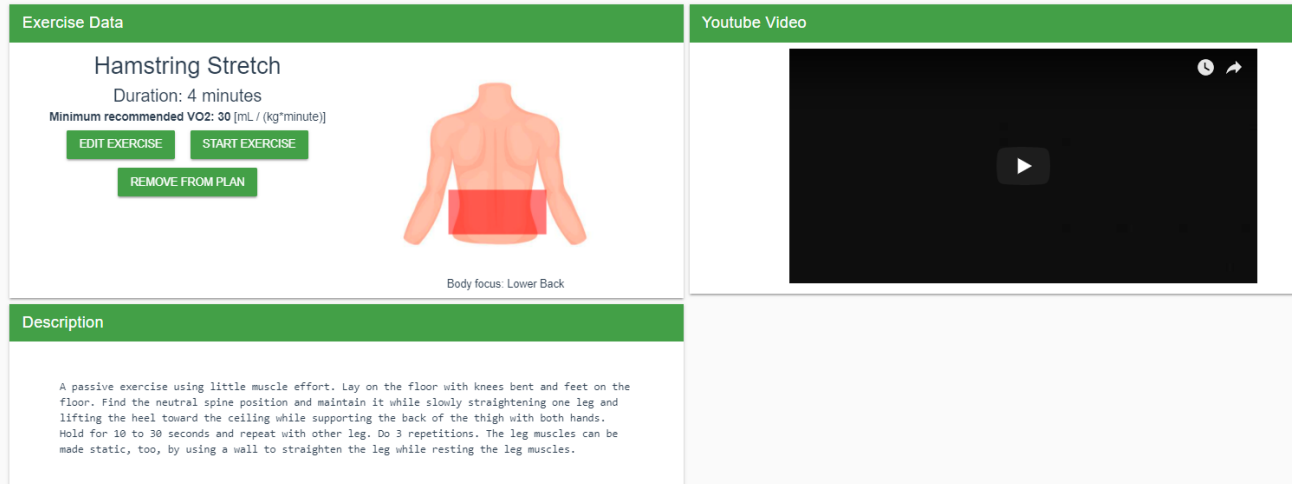
```
async post (req, res) {
  try {
    const exercise = new Exercise(req.body)
    exercise.save()
    res.send(exercise)
  } catch (err) {
    res.status(500).send({
      error: `Error while creating a new exercise.`
    })
  }
},
async index (req, res) {
  try {
    let exercises = null
    if (req.query.search) {
      exercises = await Exercise.find({
        $or: [
          {'title': new RegExp(req.query.search, 'i')},
          {'bodyFocus': new RegExp(req.query.search, 'i')}
        ]
      })
    } else {
      exercises = await Exercise.find().limit(10)
    }
    res.send(exercises)
  } catch (err) {
    res.status(500).send({
      error: `Error while retrieving exercises.`
    })
  }
},
async show (req, res) {
  try {
    const exercise = await Exercise.findById(req.params.exerciseId, '-__v')
    res.send(exercise)
  } catch (err) {
    res.status(500).send({
      error: `Error while retrieving exercises.`
    })
  }
},
async put (req, res) {
  try {
    const exerciseId = req.params.exerciseId
    const exerciseUpdate = req.body
    let exercise = await Exercise.findByIdAndUpdate(exerciseId, exerciseUpdate)
    res.send(exercise)
  } catch (err) {
    res.status(500).send({
      error: `Error while editing the exercise.`
    })
  }
},
async showSuggestions(req, res) {
  try {
    const userId = req.user._id
    const user = await User.findById(userId, '-__v')
    let exercises = await Exercise.find({
      minO2: {
        $lte: user.O2Peak
      },
      bodyFocus: {
        $in: user.bodyFocus
      }
    })
    res.send(exercises)
  } catch (error) {
    res.status(500).send({
      error: 'Error while retrieving exercises suggestions.'
    })
```

Slika 3.30. *Datoteka ExercisesController*



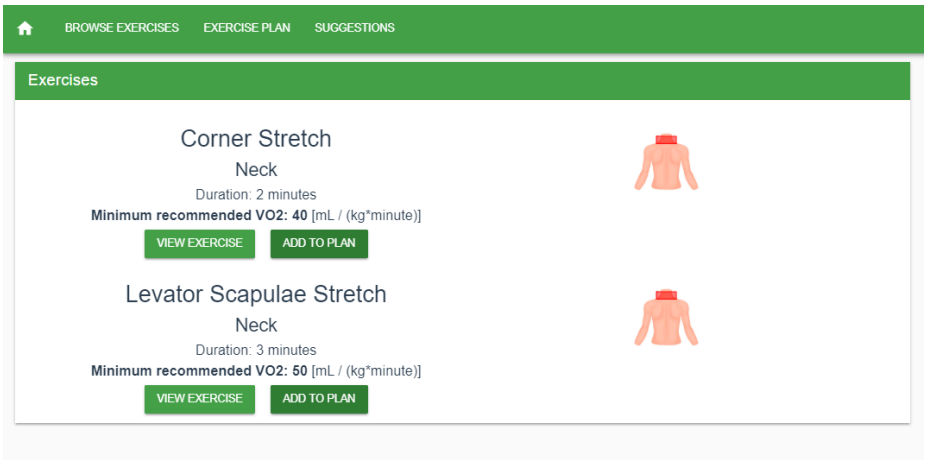
Slika 3.31. *Korisničko sučelje za pretragu i prikaz više vježbi*

Metodom *show* kao odgovor se šalje jedna vježba koja odgovara predanom identifikacijskom parametru, što je *_id* parametar vježbe, a metodom *put* se pronalazi jedna vježba koja odgovara *_id* parametru, te se ažurira prema drugim predanim parametrima. Vježba se na korisničkom sučelju prikazuje kao na slici 3.32.



Slika 3.32. *Korisničko sučelje za prikaz jedne vježbe*

Metoda *showSuggestions* će kao odgovor poslati sve vježbe čiji je minimalni preporučen VO_{2max} manji ili jednak korisnikovom maksimalnom kapacitetu i koje odgovaraju dijelu tijela za kojeg korisnik traži terapiju. Na slici 3.33 je prikazano sučelje za preporuke, konkretno za korisnika čiji VO_{2max} iznosi 53, i koji je označio vrat kao bolno područje.

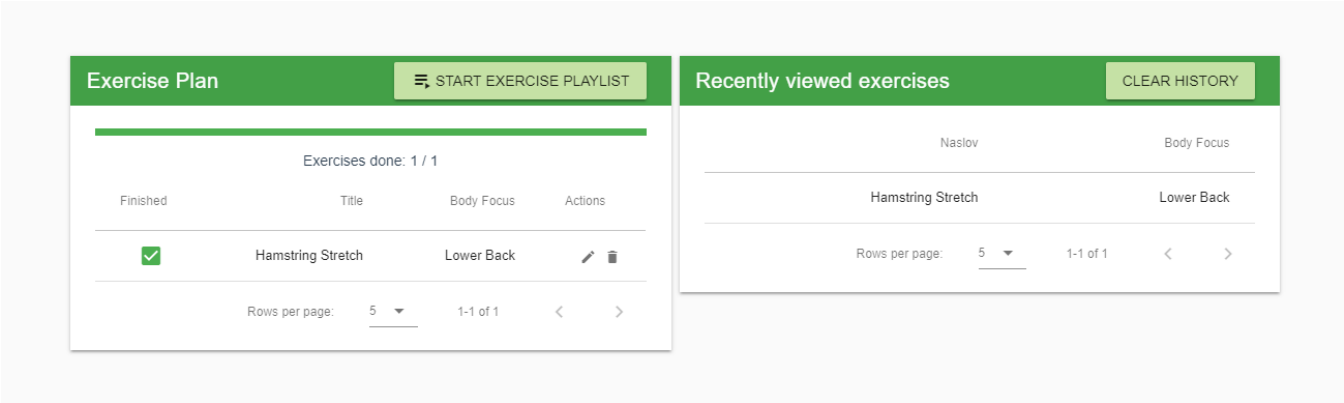


Slika 3.33. *Korisničko sučelje za prikaz preporuka*

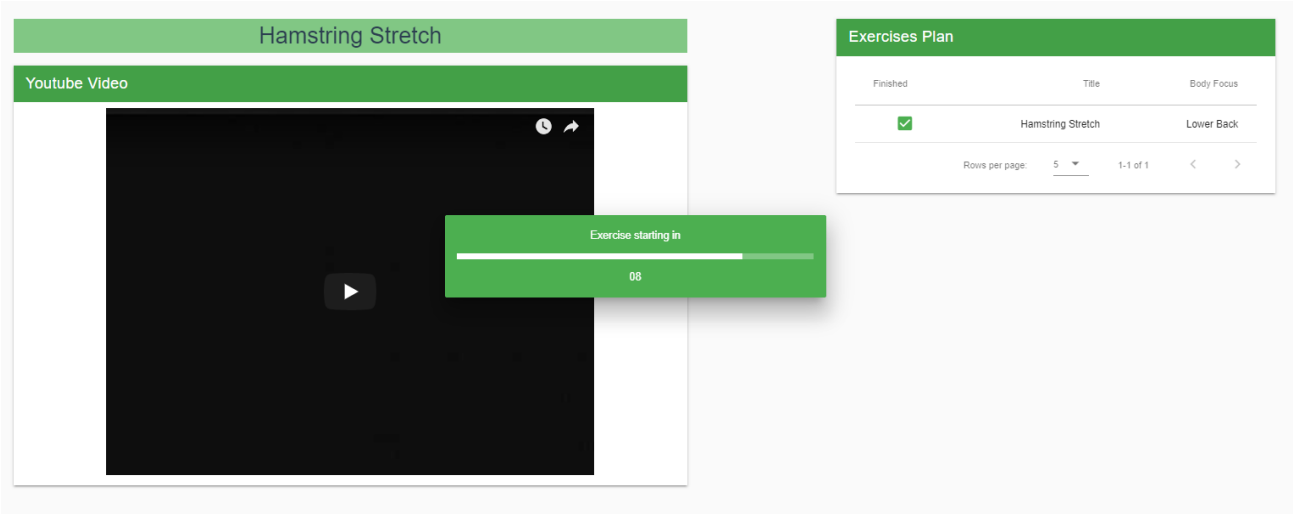
3.4.3. Povijest pregledanih vježbi i plan vježbi

U korisničkom sučelju se u planu vježbi jednostavno prate vježbe dodane u bazu podataka pritiskom na gumb „*Add to plan*“ u sučelju za detaljniji prikaz jedne vježbe. U korisničkom sučelju na slici 3.32 pritiskom na gumb „*Start exercise*“ otvara se nova stranica prikazana na slici 3.35.

Na ovoj stranici korisnik pritiskom na gumb „*Start*“ započinje odbrojavanje od 10 sekundi, nakon čijeg isteka korisnik može pokrenuti YouTube video i počinje odbrojavanje vremena koje je specifično za svaku vježbu. Tijekom odbrojavanja korisnik može pauzirati odbrojavanje. Istekom vremena pojavljuje se opcija „*Reset*“ koja će resetirati vrijeme i gumb „*Mark as finished*“ koji će tu vježbu označiti kao završenu, kao što je prikazano na slici 3.34. Korisniku je sa strane prikazan i trenutni plan vježbi kojim se može služiti radi lakšeg navigiranja. Sve završene vježbe će u tablici s planom vježbi imati zelenu kvačicu, te će se na temelju odnosa broja završenih vježbi i broja ukupnih vježbi u planu ažurirati traka napretka, kao i tekst ispod nje. Korisnik može kliknuti na naslov vježbe, te će biti preusmjeren na detaljniji prikaz te vježbe. Osim toga, korisnik može ukloniti vježbu s plana klikom na ikonu koša za smeće, ili, ukoliko je korisnik administrator, može kliknuti na ikonu olovke koja će ga preusmjeriti na stranicu za uređivanje vježbe. Korisnik također može očistiti povijest pregledanih vježbi klikom na „*Clear history*“ gumb na panelu „*Recently viewed exercises*“. Korisnika se klikom na gumb „*Start exercise playlist*“ preusmjerava na vizualno identično sučelje kao i klikom na gumb „*Start*“ (slika 3.35), no razlika je u tome što se video automatski pokreće, a nakon isteka vremena pojedine vježbe korisnika se automatski preusmjerava na sljedeću vježbu u planu, a završena vježba se označava kao završena.



Slika 3.34. *Korisničko sučelje za plan i povijest pregledanih vježbi*



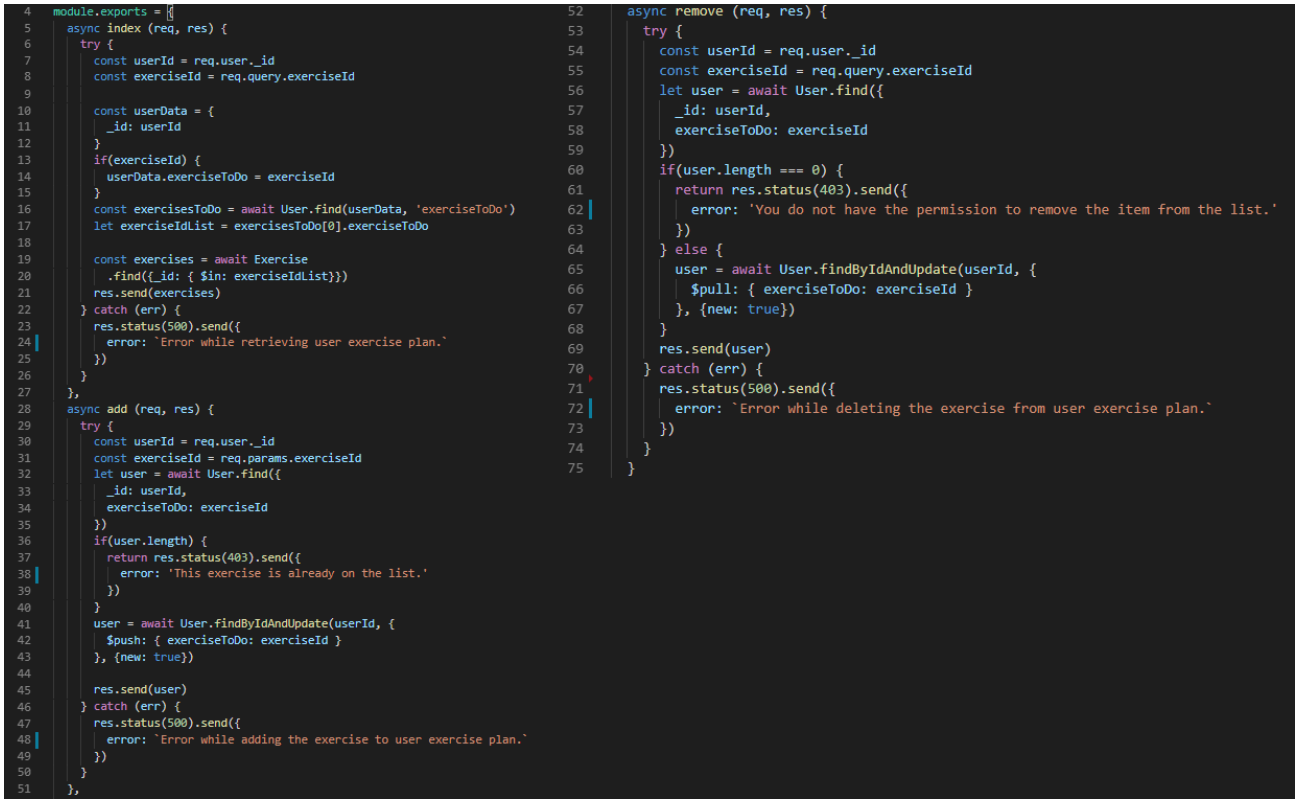
Slika 3.35. *Korisničko sučelje sa videom vježbe i odbrojavanjem*

Na strani korisničkog sučelja, tablica se ažurira kao što je prikazano na slici 3.36 na sljedeći način: kada se stranica učita, ako je korisnik prijavljen, poziva se *index* metoda iz *ExercisesPlanService* datoteke koja će se preko *axios*-a poslati na obradu poslužitelju.

```
76   export default {
77     data () {···
109    },
110    async mounted () {
111      if (this.isUserLoggedIn) {
112        this.planFinished = (await ExercisesDoneService.index()).data
113        this.plan = (await ExercisesPlanService.index()).data
114        this.progress = (this.planFinished.length / this.plan.length) * 100
115      }
116    },
117    methods: {
118      goToRoute (route, id) {···
120      },
121      containsObject (obj, list) {···
124      },
125      async removeFromPlan (exercise, index) {···
140      }
141    },
```

Slika 3.36. *Kod na strani korisničkog sučelja za dohvaćanje planiranih vježbi*

Na strani poslužitelja, u *ExercisesPlanController* datoteci prikazanoj na slici 3.37 nalaze se metode za manipuliranje podacima vezanim za plan vježbi. *Index* metodom nalazi se prijavljeni korisnik, i podatak u njegovom modelu *exerciseToDo*, koje je tipa podataka polje koje sadrži identifikacijske brojeve dodanih vježbi. Zatim se, prema svim identifikacijskim brojevima, pronalaze i kao odgovor šalju sve vježbe u tom polju. *Add* metoda se pozove iz korisničkog sučelja prikazanim na slici 3.33 pritiskom na gumb „*Add to plan*“. Ona pronalazi prijavljenog korisnika koji šalje zahtjev za dodavanjem te vježbe, te mu u *exerciseToDo* polje stavlja identifikacijski broj željene vježbe. Na sličan način je implementirana i *remove* metoda, koja iz tog istog polja briše identifikacijski broj željene vježbe.



Slika 3.37. *Kod na strani poslužitelja za plan vježbi*

Povijest pregledanih vježbi nalazi se u *HistoriesController* datoteci koja je prikazana na slici 3.38, i izvedena je na gotovo identičan način kao i plan vježbi, uz male promjene kod pozivanja metoda i načina spremanja podataka. *Index* metoda kod povijesti pregledanih vježbi pozove se u istim slučajevima kao i *index* metoda kod vježbi dodane u plan, te na identičan način dohvaća podatke. *Add*

metoda kod povijesti pregledanih vježbi je malo drukčija od *add* metode kod plana vježbi. Ova *add* metoda je pozvana svaki puta kada učitamo detaljniji prikaz vježbe prikazan na slici 3.32, a radi na način da pronađe ulogiranog korisnika, te u njegovo *history* polje sprema identifikacijski broj pregledane vježbe, pritom pazeći da je to polje jedinstvenih vrijednosti, zato se, umjesto *$push* metode kao kod dodavanja plana u vježbu, ovdje koristi *$addToSet* metoda. Također, pregledanoj vježbi ažurira se polje *historyUpdateAt* novim datumom, kako bi bilo moguće izvršiti sortiranje po vremenu posjete na korisničkom sučelju. *Clear* metoda jednostavno postavlja korisniku njegovo *history* polje na prazno polje, te se na taj način čisti povijest pregledanih vježbi.

```
async index (req, res) {
  try {
    const userId = req.user._id
    const exerciseId = req.query.exerciseId
    const userData = {
      _id: userId
    }
    if(exerciseId) {
      userData.history = exerciseId
    }
    const history = await User.find(userData, 'history')
    let exerciseIdList = history[0].history
    const exercises = await Exercise
      .find({_id: { $in: exerciseIdList}})
    res.send(exercises)
  } catch (err) {
    res.status(500).send({
      error: `Error while retrieving user history.`
    })
  }
},
async add (req, res) {
  try {
    const userId = req.user._id
    const exerciseId = req.body.exerciseId

    const user = await User.findByIdAndUpdate(userId, {
      $addToSet: { history: exerciseId }
    }, {new: true})

    await Exercise.findByIdAndUpdate(exerciseId, {historyUpdateAt: new Date()})
    res.send(user)
  } catch (err) {
    res.status(500).send({
      error: `Error while adding the item to history.`
    })
  }
},
async clear (req, res) {
  try {
    const userId = req.user._id
    const user = await User.findByIdAndUpdate(userId, {
      $set: { history: [] }
    }, {new: true})
    res.send(user)
  } catch (err) {
    res.status(500).send({
      error: `Error while clearing user history.`
```

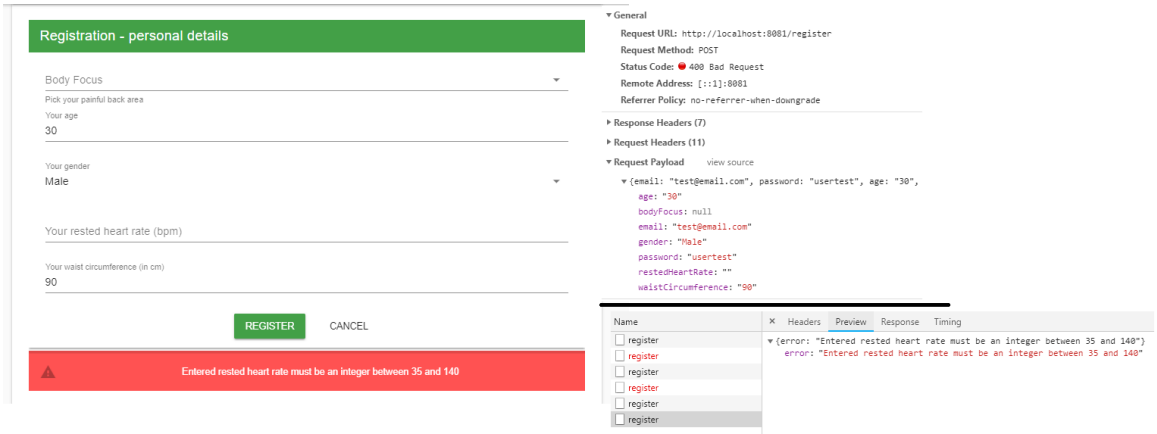
Slika 3.38. *Kod na strani poslužitelja za povijest pregledanih vježbi*

4. TESTIRANJE RADA APLIKACIJE

U ovom poglavlju provjerit ćemo pravilan rad web aplikacije prateći poslane zahtjeve prema poslužitelju pomoću *Network* sekcije u *Chrome Developer Tools* alatu.

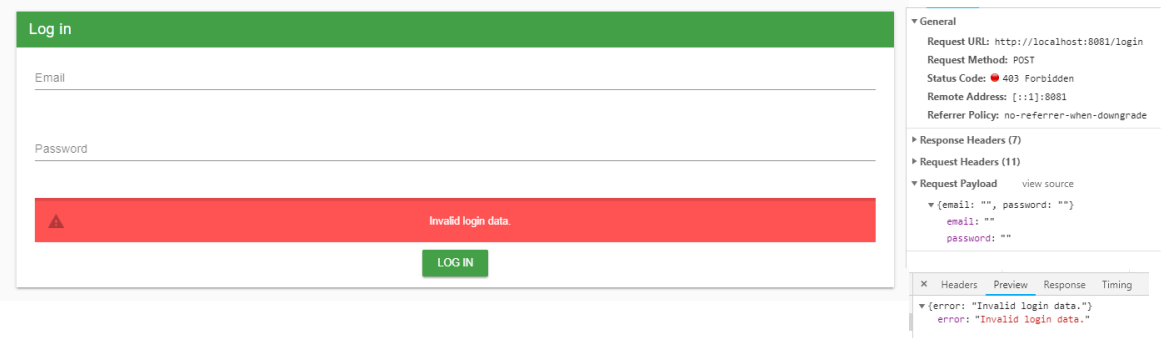
4.1. Korisnički profil

U slučaju da su neka potrebna polja pri registraciji prazna i korisnik klikne na gumb „*Register*“, šalje se HTTP POST zahtjev poslužitelju, a poslužitelj odgovara HTTP kodom 400 i porukom kako je došlo do greške kako je i definirano u kodu aplikacije. Greške će se pojavljivati za svaki slučaj neuspješne validacije podataka, a primjer jedne prikazan je na slici 4.1.



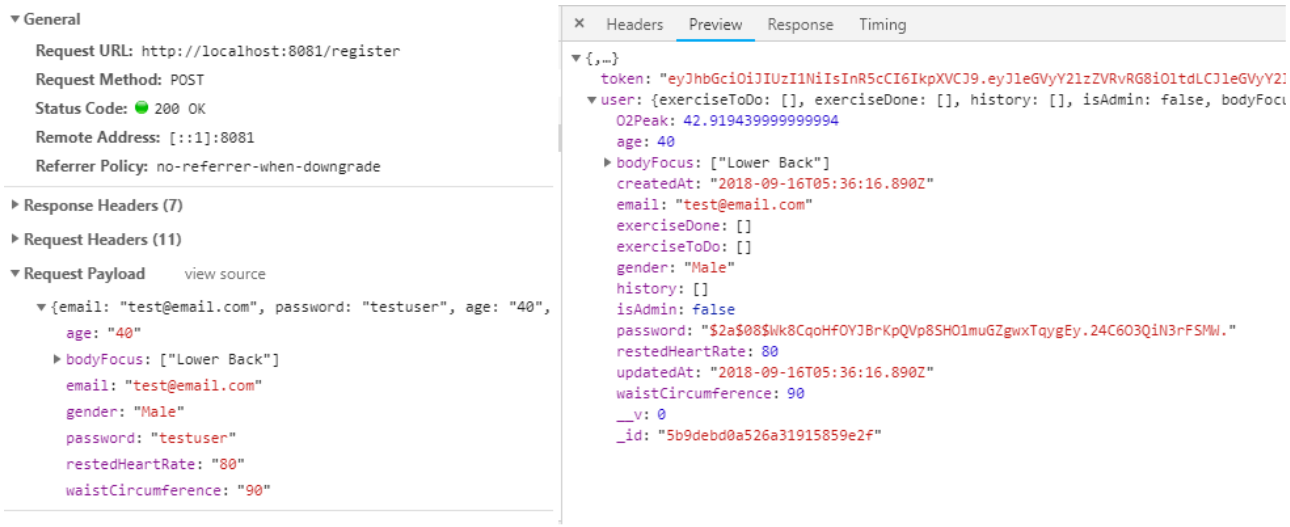
Slika 4.1. *Prikaz neuspješne registracije (lijevo) i zahtjeva i odgovora poslužitelja (desno)*

Isti je slučaj i kod prijave korisnika – u slučaju netočno unesenih podataka, poslužitelj će odgovoriti HTTP kodom 403 i odgovarajućom porukom (Slika 4.2).



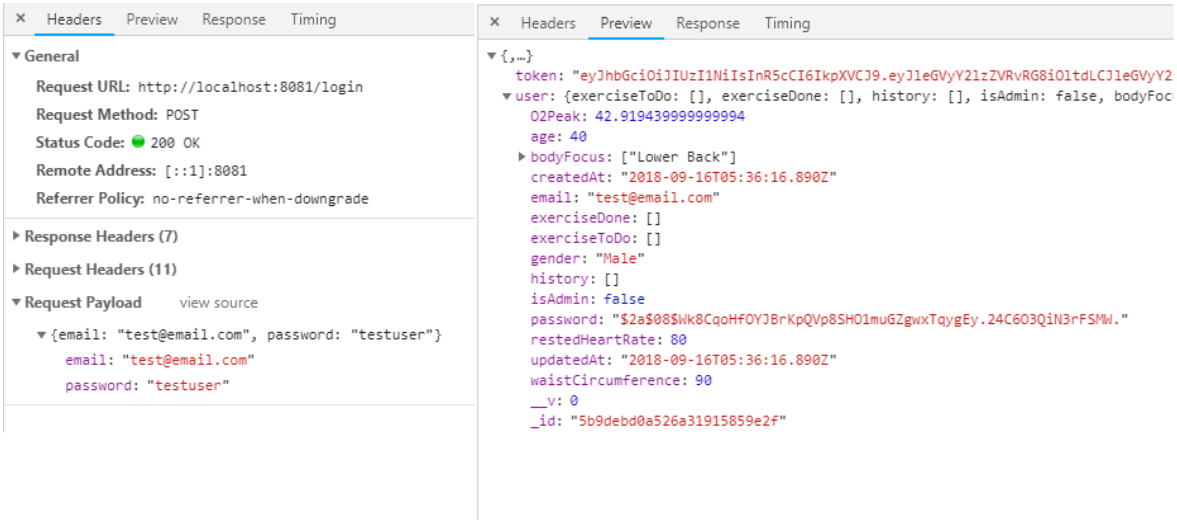
Slika 4.2. *Prikaz neuspješne prijave (lijevo) i zahtjeva i odgovora poslužitelja (desno)*

Ako su pri registraciji ispunjena sva potrebna polja i korisnik klikne na gumb „*Register*“, poslužitelj vraća odgovor s HTTP kodom 200 i spremljenim korisnikom. Na slici 4.3 prikazana je registracija korisnika sa e-mailom „test@email.com “ i lozinkom „testuser“. Pošto sva polja zadovoljavaju uvjete, poslužitelj odgovara na očekivan način, te kao odgovor, uz HTTP kod 200, vraća korisnika sa šifriranom lozinkom i njegov JWT žeton.

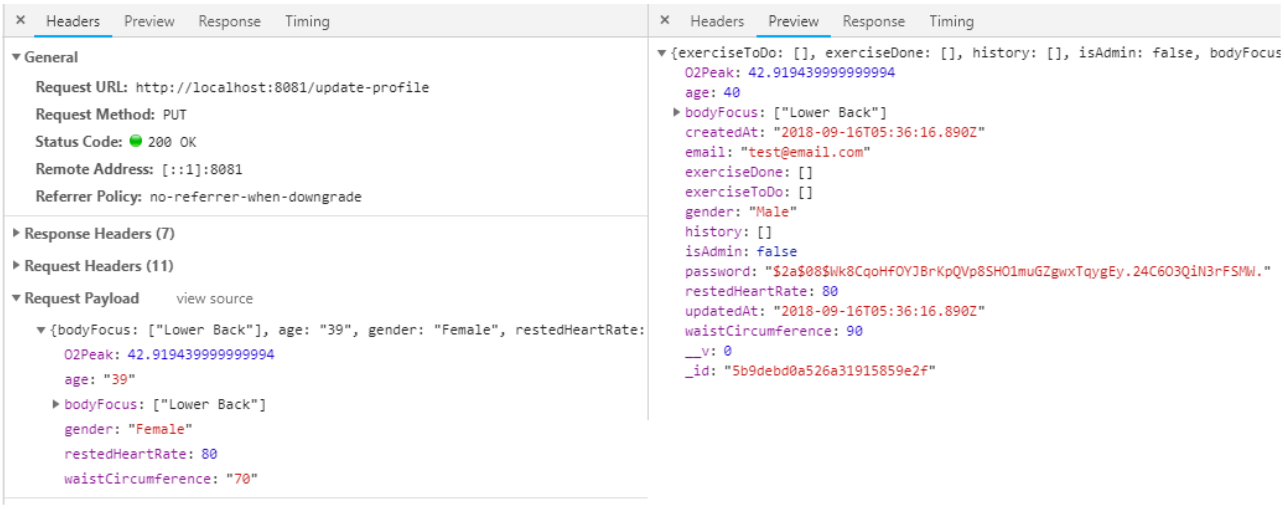


Slika 4.3. *Prikaz zahtjeva poslužitelju (lijevo) i prikladnog odgovora (desno) za registraciju korisnika*

Identična situacija je i kod prijave korisnika (Slika 4.4) i kod ažuriranja, tj. izmjene profila (Slika 4.5). U slučaju da je prijava uspješna, odgovor poslužitelja je HTTP kod 200, korisnik i njegov pripadajući JWT žeton, a u slučaju da su korisnikovi podaci uspješno izmjenjeni, poslužitelj šalje HTTP kod 200 zajedno s objektom korisnika.



Slika 4.4. *Prikaz zahtjeva poslužitelju (lijevo) i prikladnog odgovora (desno) za prijavu korisnika*

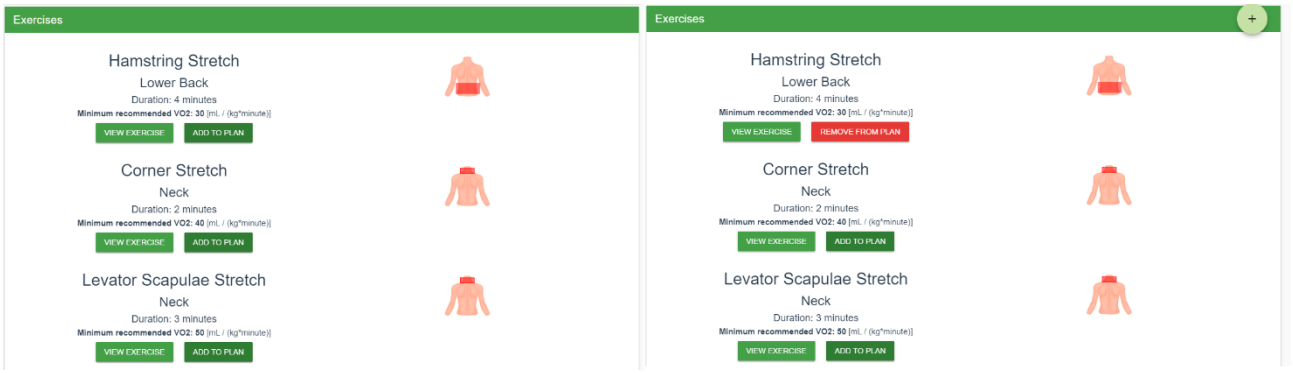


Slika 4.5. *Prikaz zahtjeva poslužitelju (lijevo) i prikladnog odgovora (desno) za izmjenu podataka korisnika*

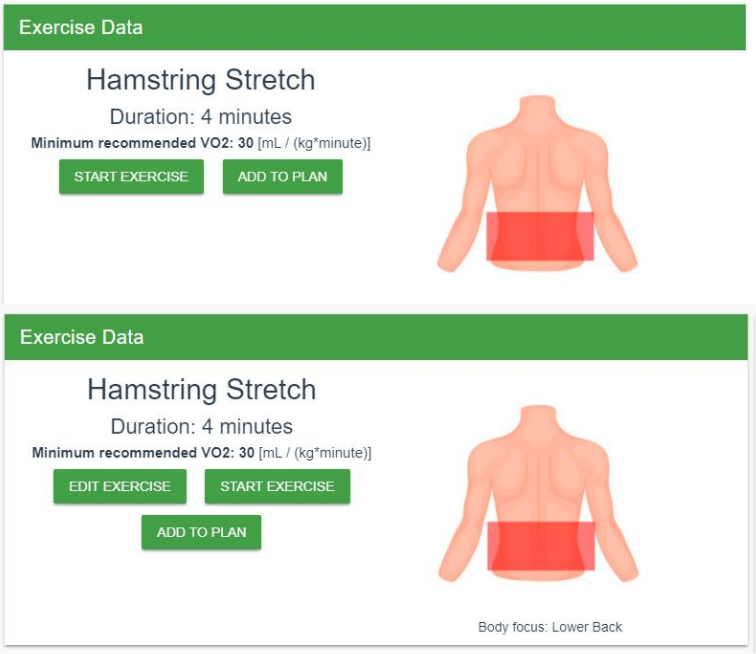
4.2. Dodavanje, izmjena i dohvaćanje vježbi

Na slici 4.6 prikazano je sučelje panela za vježbe, a na slici 4.7 sučelje detaljnijeg prikaza vježbe za obične korisnike i administratore, tj. korisnike višeg stupnja koji mogu dodavati nove i uređivati

postojeće vježbe. Običnim korisnicima se ne prikazuje gumb u gornjem desnom kutu panela „*Exercises*“ kojim se dodaju nove vježbe, kao ni gumb „*Edit exercise*“ na panelu „*Exercise Data*“.

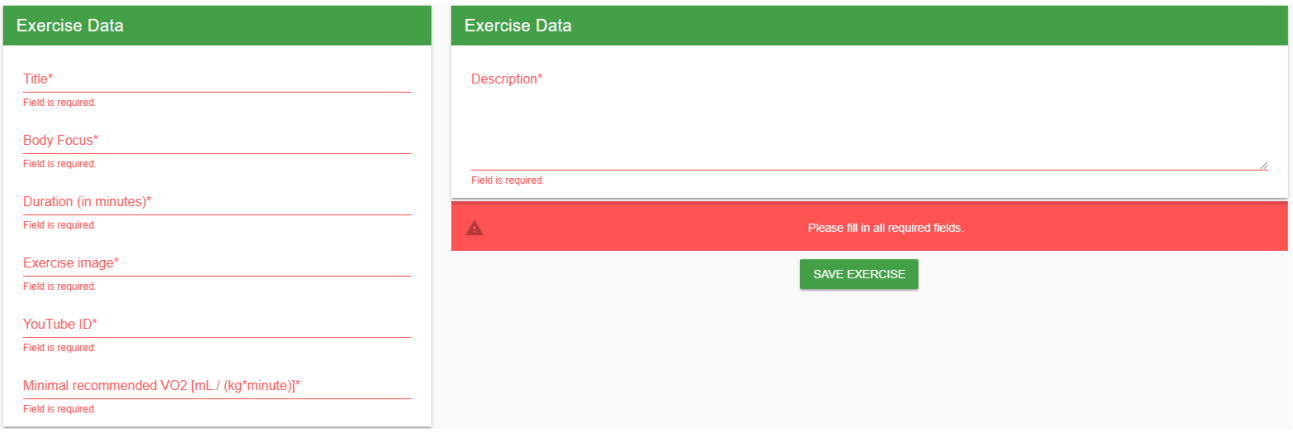


Slika 4.6. *Sučelje panela vježbi za obične korisnike (lijevo) i administratore (desno)*



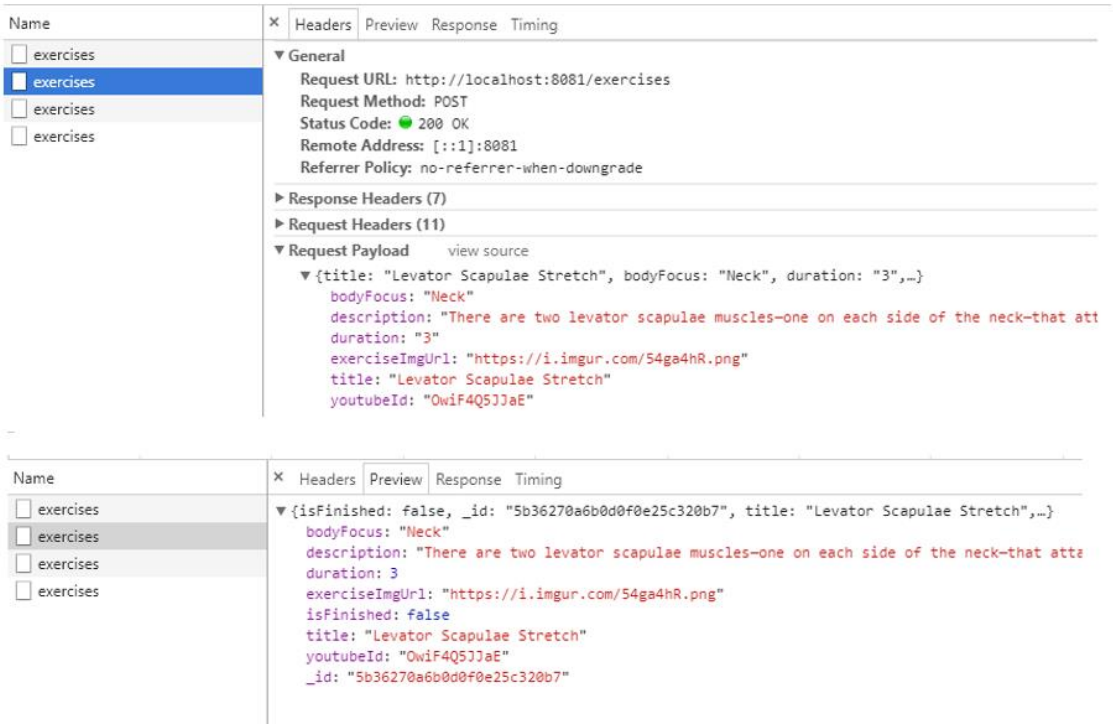
Slika 4.7. *Sučelje detaljnijeg prikaza vježbe za obične korisnike (gore) i administratore (dolje)*

Na sučelju za dodavanje vježbe prikazanom na slici 4.8 moraju se ispuniti sva označena polja. Zahtjev se poslužitelju neće poslati sve dok sva potrebna polja nisu ispunjena.



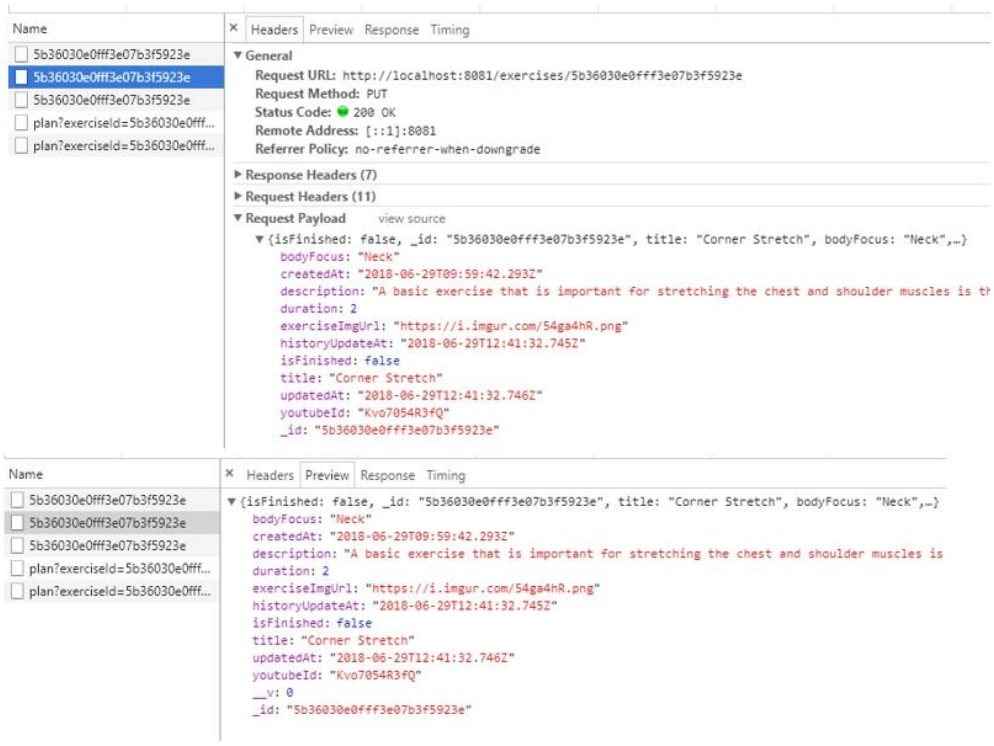
Slika 4.8. *Sučelje za dodavanje nove vježbe*

Kada se ispune sva polja, pošalje se HTTP POST metoda poslužitelju, koji unesene podatke sprema u bazu podataka i odgovara HTTP kodom 200 i spremljenom vježbom (Slika 4.9).



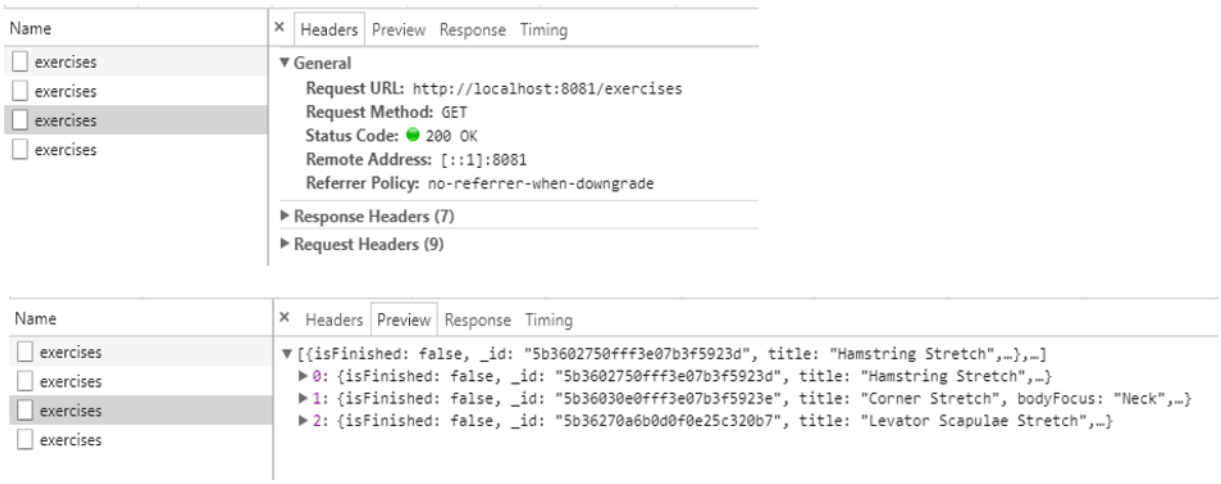
Slika 4.9. *Zahtjev (gore) i odgovor (dolje) na kreiranje nove vježbe*

Za izmjenu podataka o vježbi, u slučaju greške implementirana je ista logika kod validacije ispune svih polja, te se ne šalje zahtjev poslužitelju dok sva polja nisu ispunjena. U slučaju valjanog zahtjeva, poslužitelju se šalje HTTP PUT zahtjev zajedno sa unesenim parametrima. U slučaju da je zahtjev uspješno zaprimljen i obrađen, poslužitelj odgovara HTTP kodom 200 i ažuriranim podacima o vježbi (Slika 4.10).



Slika 4.10. *Zahtjev (gore) i odgovor (dolje) na uređivanje postojeće vježbe*

Osim kreiranja novih vježbi, važno je i dobiti informacije o spremljenim vježbama koje će se prikazati na panelu za vježbe. Na slici 4.11 vidi se da je na učitavanje panela za vježbe poslan HTTP GET zahtjev poslužitelju koji odgovara HTTP kodom 200 i poljem objekata u kojem je svaki objekt jedna vježba spremljena u bazi podataka.

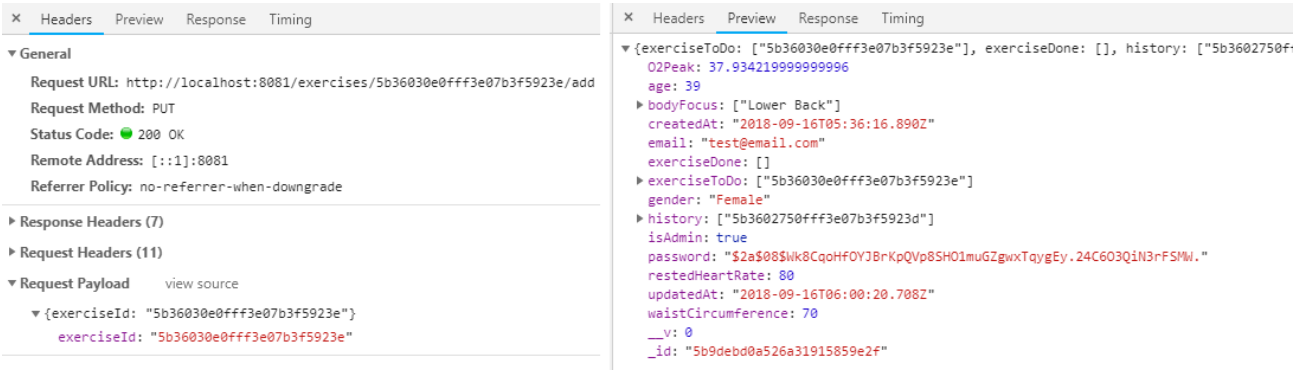


Slika 4.11. *Zahtjev (gore) i odgovor (dolje) na prikaz vježbi*

4.3. Povijest pregledanih vježbi i plan vježbi

Kao što je već opisano u prethodnim poglavljima, klikom na gumb „*Add to plan*“ na detaljnijem prikazu vježbe vježbu je moguće dodati na popis planiranih vježbi.

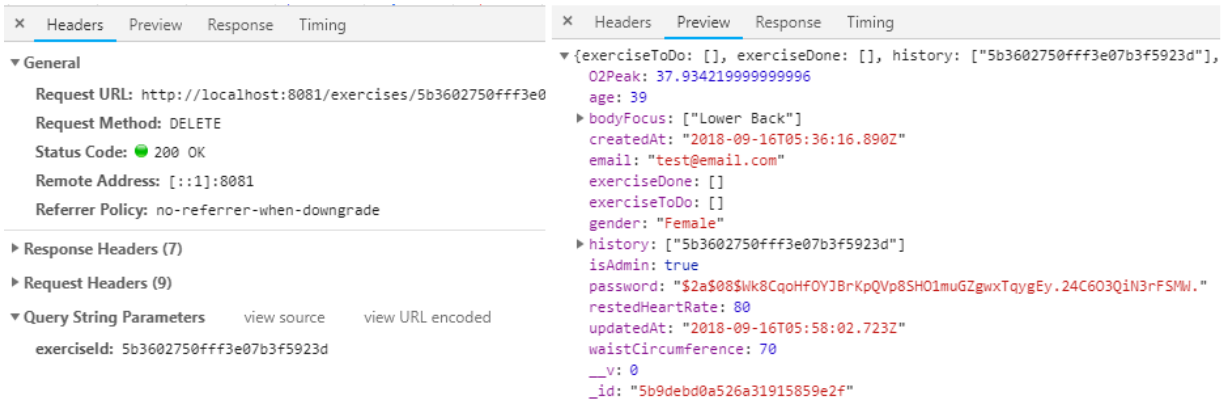
Klikom na taj gumb šalje se PUT zahtjev poslužitelju zajedno sa identifikacijskim brojem vježbe koju korisnik želi dodati u plan, a u slučaju uspješnog događaja poslužitelj odgovara HTTP kodom 200 i ažuriranim informacijama o korisniku (Slika 4.12).



Slika 4.12. *Zahtjev (lijevo) i odgovor (desno) na dodavanje vježbe u plan*

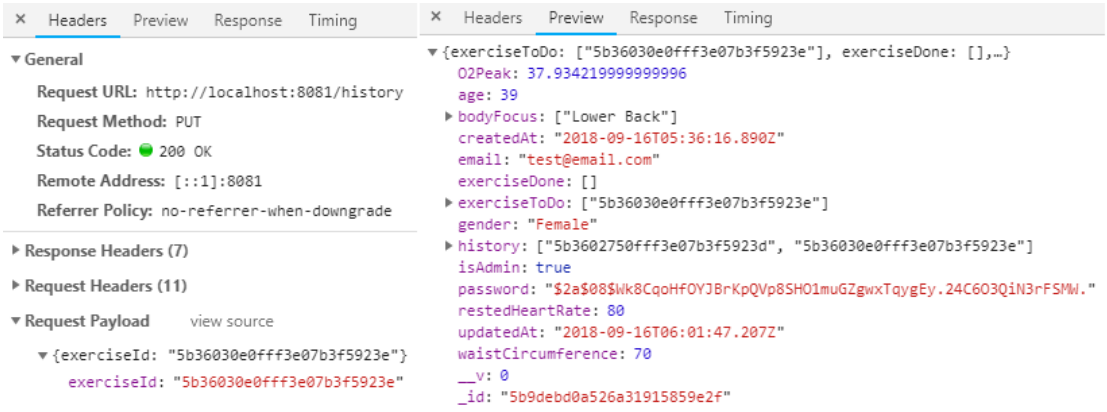
U slučaju da vježbu korisnik želi ukloniti s plana, klikom na gumb „*Remove from plan*“ na istom panelu poslat će se HTTP DELETE metoda poslužitelju koji će, identično kao i kod dodavanja vježbe

u plan, u slučaju uspješnog događaja odgovoriti HTTP kodom 200 i ažuriranim informacijama o korisniku (Slika 4.13).



Slika 4.13. *Zahtjev (lijevo) i odgovor (desno) na brisanje vježbe iz plana*

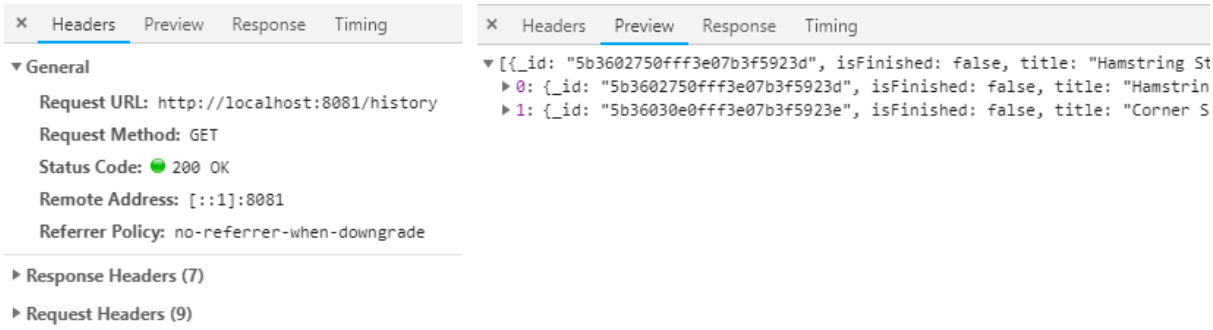
HTTP PUT zahtjev za dodavanjem vježbe u povijest pošalje se svaki put kada se korisnik otvori detaljniji prikaz vježbe, te se u slučaju uspješnog događaja korisniku šalje HTTP kod 200 i ažurirane informacije o korisniku (Slika 4.14).



Slika 4.14. *Zahtjev (lijevo) i odgovor (desno) na dodavanje vježbe u povijest vježbi*

Osim dodavanja i brisanja podataka o planu i povijesti vježbi, potrebno je i dohvatiti informacije iz baze podataka o vježbama i prikazati ih na sučelju, stoga se šalju HTTP GET zahtjevi poslužitelju.

Poslužitelj, u slučaju uspješnog događaja, odgovara HTTP kodom 200 i poljem objekata gdje je svaki objekt jedna vježba u bazi podataka. Na slici 4.15 je prikazan jedan takav zahtjev i odgovor za povijest vježbi, a situacija je identična za plan vježbi.

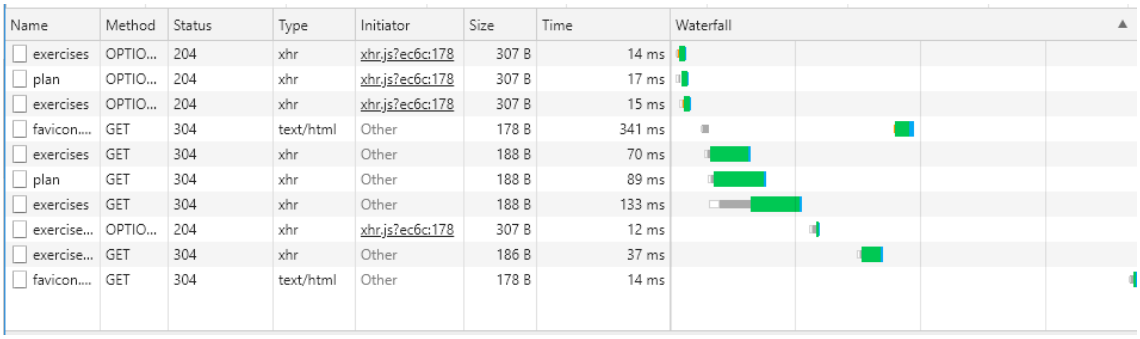


Slika 4.15. *Zahtjev (lijevo) i odgovor (desno) na dohvaćanje povijesti vježbi*

4.4. Performanse web aplikacije

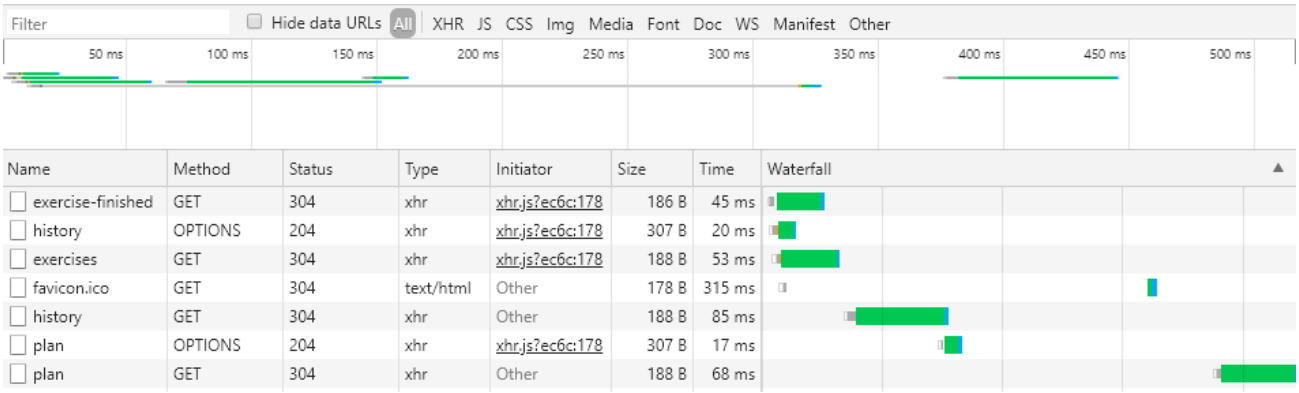
Jedna od najvažnijih karakteristika bilo kojeg programa ili aplikacije je, osim davanja točnih rezultata, i brzina izvođenja programa. Pomoću *Chrome Developer Tools-a* pratit ćemo brzinu učitavanja nekih Vue pogleda i slanja zahtjeva prema poslužitelju, te ukratko analizirati dobivene rezultate.

Na slici 4.16 vidljivo je vrijeme učitavanja korisničkog sučelja za pretragu i prikaz vježbi. Za učitavanje cijelog sučelja potrebno je oko 750 ms, od kojih je 341 ms potrebno za učitavanje favicon.ico ikone, koja će biti iskorištena u slučaju da korisnik doda stranicu na popis favorita. U kontekstu slanja zahtjeva prema poslužitelju, najviše vremena je potrebno za obradu GET zahtjeva koji dohvaća popis vježbi (133 ms).



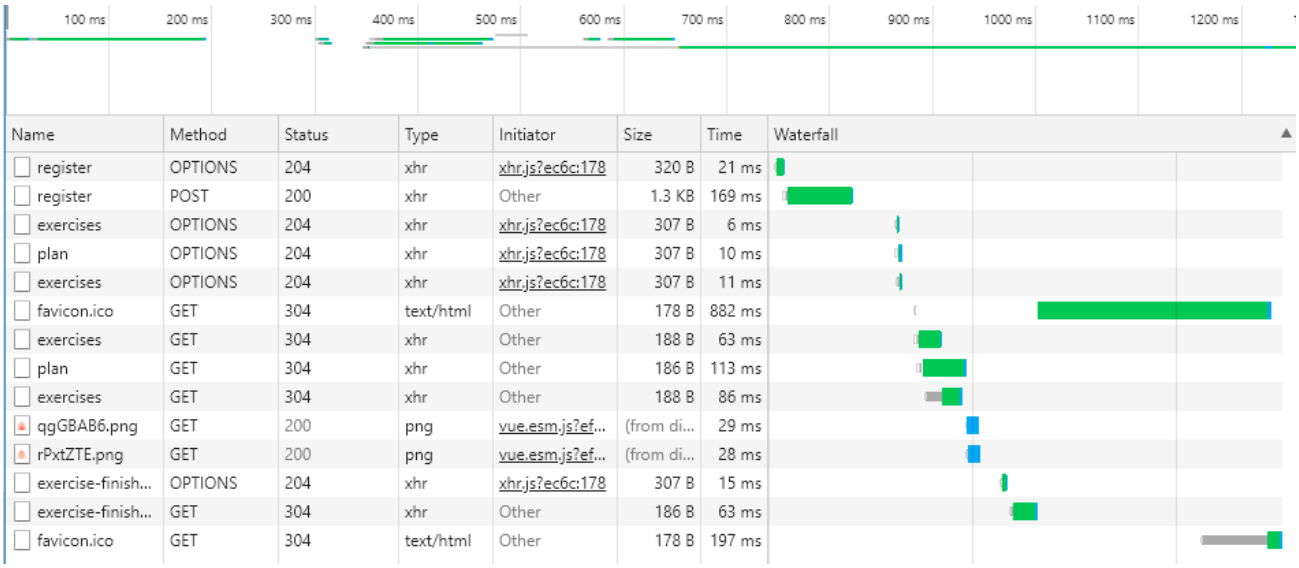
Slika 4.16. *Vrijeme potrebno za učitavanje korisničkog sučelja za pregled vježbi*

Za dohvaćanje plana vježbi i povijesti pregleda primjećuje se slična situacija, prikazana na slici 4.17. Vrijeme potrebno za učitavanje cijelog korisničkog sučelja iznosi oko 450 ms, od kojih je 315 ms potrebno za učitavanje favicon.ico ikone, 68 ms za učitavanje plana i 85 ms za učitavanje povijesti pregleda.



Slika 4.17. *Vrijeme potrebno za učitavanje korisničkog sučelja za plan i povijest pregleda vježbi*

Zadnji zahtjev koji će se analizirati je POST zahtjev za registraciju prikazan na slici 4.18. Ukupno vrijeme za registraciju korisnika je oko 1300 ms, od kojih je tek 170 ms potrebno za obradu POST zahtjeva s korisničkim podacima.



Slika 4.18. *Vrijeme potrebno za registraciju korisnika*

5. ZAKLJUČAK

Kroz završni rad izrađena je web aplikacija za potporu i praćenje fizikalnih terapija kralježnice. Implementirana je mogućnost registracije, prijave i odjave korisnika, validacije unesenih podataka, napravljena je podjela na obične korisnike, koji mogu dodati vježbe u svoj plan vježbanja i pratiti svoje rezultate, i administratore, koji imaju mogućnosti dodavanja novih i uređivanja već postojećih vježbi. U radu je dan kratki osvrt na terapije, te je dan prikaz sličnih postojećih rješenja ovog problema. Prikazan je plan aplikacije i njezine mogućnosti, te su objašnjene tehnologije pomoću kojih je aplikacija izrađena – Vue.js za korisničko sučelje, MongoDB kao baza podataka, te Node.js i Express.js kao programska podrška na strani poslužitelja. Prikazan je način kako organizirati složeniju aplikaciju razdvajajući je na manje module koji zajedno obavljaju jednu veliku funkciju. Objašnjeni su dijelovi koda od iznimne važnosti i za korisničko sučelje i programsku podršku na strani poslužitelja. Aplikacija je testirana i utvrđeno je da svi dijelovi aplikacije funkcioniraju kako je i planirano uz visoke performanse koristeći alate web preglednika Google Chrome.

LITERATURA

[1] Anonimno, 8 glavnih uzročnika bolova u leđima i rebrima, https://www.krenizdravo.rtl.hr/vitalnost/8-glavnih-uzrocnika-bolovi-u-ledima (posjećeno 20.6.2018.)

[2] P.F. Ullrich, Jr., Physical Therapy Benefits For Back Pain, https://www.spine-health.com/treatment/physical-therapy/physical-therapy-benefits-back-pain (posjećeno 20.6.2018.)

[3] J.J. Gopez, Exercise and Back Pain, https://www.spine-health.com/wellness/exercise/exercise-and-back-pain (posjećeno 20.6.2018.)

[4] A. Merritt, The VO2 Max formula, https://www.acefitness.org/fitness-certifications/resource-center/exam-preparation-blog/1545/the-vo2-max-formula (posjećeno 31.8.2018.)

[5] B.M. Nes, I. Janszky, L.J. Vatten, T.I.L. Nilsen, S.T. Aspenes, U. Wisløff, Estimating V˙O2peak from a Nonexercise Prediction Model: The HUNT Study, https://journals.lww.com/acsmmsse/fulltext/2011/11000/Estimating_V_O2peak_from_a_Nonexercise_Prediction.2.aspx (posjećeno 5.9.2018.)

[6] Internetska stranica ExRx - https://exrx.net/ (posjećeno 20.6.2018)

[7] Internetska stranica FitnessBlender - https://www.fitnessblender.com/ (posjećeno 20.6.2018)

[8] Internetska stranica Spine-Health - https://www.spine-health.com/ (posjećeno 20.6.2018)

[9] Internetska stranica PhysiotherapyExercises - https://www.physiotherapyexercises.com/ (posjećeno 3.9.2018)

[10] Anonimno, Introducing JSON - https://www.json.org/ (posjećeno 22.6.2018.)

[11] Anonimno, Introduction to the DOM, https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction (posjećeno 23.6.2018.)

[12] Anonimno, What is Vue.js and What are its Advantages - https://hackernoon.com/what-is-vue-js-and-what-are-its-advantages-4071b7c7993d (posjećeno 24.6.2018.)

[13] B. Krajka, The difference between Virtual DOM and DOM – http://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/ (posjećeno 24.6.2018.)

[14] T. Freed, What is Virtual Dom - https://tonyfreed.blog/what-is-virtual-dom-c0ec6d6a925c (posjećeno 24.6.2018.)

[15] Vue.js dokumentacija - https://012.vuejs.org/images/mvvm.png (posjećeno 24.6.2018.)

[16] MongoDB dokumentacija - https://www.mongodb.com/what-is-mongodb (posjećeno 25.6.2018.)

[17] Mongoose dokumentacija - http://mongoosejs.com/docs/guide.html (posjećeno 25.6.2018.)

[18] Node.js dokumentacija - https://nodejs.org/en/docs/ (posjećeno 25.6.2018.)

[19] P. Patel, What exactly is Node.js? - https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5 (posjećeno 25.6.2018.)

[20] A. Jones, Node.js Stream Dreams - https://www.zivtech.com/blog/nodejs-stream-dreams (posjećeno 25.6.2018.)

SAŽETAK

U ovom završnom radu modelirana je i izrađena web aplikacija za pomoć i potporu ljudima koji imaju problema s kralježnicom. Ostvareno je korisničko sučelje i programska podrška na strani poslužitelja koja omogućuje prijavu korisnika, dodavanje nove terapije, prikaz spremljenih terapija, te analizu i prikaz rezultata terapije. Izrađena aplikacija realizirana je pomoću Vue.js programskog okvira za izradu korisničkog sučelja, MongoDB baze podataka, te Node.js izvršnog okruženja i Express.js programskog okvira kojima se realizirala programska podrška na strani poslužitelja. Testiran je rad aplikacije koji se pokazao kao ispravan, uz visoke performanse.

Ključne riječi: Express, medicina, Mongo, Node, terapije, Vue, web aplikacija

ABSTRACT

WEB APPLICATION FOR SUPPORT AND MONITORING OF PHYSICAL THERAPIES

In this bachelor's thesis a web application for helping and supporting people who have health problems with their spine was modeled and built. A user interface was created, along with server-side backend which enables user registration, addition of new therapies, overview of the saved therapies and analysis and overview of the therapy results. Application was built using Vue.js framework for the frontend, MongoDB as the database, and Node.js as the backend runtime environment in combination with Express.js framework. Application was tested, and it showed that application works as intended, with high performance.

Keywords: Express, medicine, Mongo, Node, therapies, Vue, web application

ŽIVOTOPIS

Filip Česnek rođen je 7.listopada 1996. u Osijeku. Nakon završene osnovne škole u Magadenovcu, 2011. godine upisuje se u Elektrotehničku i prometnu školu u Osijeku, gdje stječe zvanje tehničar za mehatroniku. Nakon završetka srednje škole, 2015. godine upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

PRILOZI

Prilog 1. Dokument završnog rada u .docx formatu

Prilog 2. Dokument završnog rada u .pdf formatu

Prilog 3. Programski kod izrađene web aplikacije