Razvoj i testiranje web sustava za stvaranje preporuka prikladnih lijekova

Papež, Josip

Master's thesis / Diplomski rad

2021

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:200:985322

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-19**

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 diplomski studij računarstva

RAZVOJ I TESTIRANJE WEB SUSTAVA ZA STVARANJE PREPORUKA PRIKLADNIH LIJEKOVA

Diplomski rad

Josip Papež

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 07.09.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Josip Papež
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1080R, 06.10.2019.
OIB studenta:	74922527756
Mentor:	Prof.dr.sc. Goran Martinović
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 1:	Prof.dr.sc. Goran Martinović
Član Povjerenstva 2:	Izv. prof. dr. sc. Mirko Köhler
Naslov diplomskog rada:	Razvoj i testiranje web sustava za stvaranje preporuka prikladnih lijekova
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U teorijskom dijelu diplomskog rada treba opisati probleme izbora prikladnih lijekova s posebnim naglaskom na anamnezu pacijenta, kontraindikacije i nuspojave. Nadalje, treba analizirate sustave preporuka koji su prikladni za rješavanje opisanih problema, a posebno sustave preporuka zasnovane na višekriterijskom donošenju odluka, filtriranju na temelju sadržaja i na kolaborativnom filtriranju. Također, treba analizirati i predložiti model i programsku arhitekturu web sustava na strani korisnika i poslužitelja, prikladne programske
Prijedlog ocjene pismenog dijela ispita (diplomskog 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:	07.09.2021.

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, 16.09.2021.

Ime i prezime studenta:	Josip Papež
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1080R, 06.10.2019.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Razvoj i testiranje web sustava za stvaranje preporuka prikladnih lijekova**

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:

SADRŽAJ

1. UVOD 1

2. SUSTAVI ZA ODABIR PRIKLADNIH LIJEKOVA NA TEMELJU PREPORUKA I PREGLED STANJA U PODRUČJU 2

2.1. Problemi odabira lijekova 2

2.2. Stvaranje preporuka za izbor prikladnih lijekova 4

2.3. Značaj testiranja pri razvoju web sustava za preporuku prikladnih lijekova 4

2.4. Postojeća rješenja za stvaranje preporuka prikladnih lijekova 5

3. IDEJNO RJEŠENJE, MODEL I GRAĐA WEB SUSTAVA ZA STVARANJE PREPORUKA PRIKLADNIH LIJEKOVA 9

3.1. Funkcionalni i nefunkcionalni zahtjevi na web sustav 9

3.1.1. Funkcionalni zahtjevi 9

3.1.2. Nefunkcionalni zahtjevi 10

3.2. Postupci stvaranja preporuka prikladnih lijekova 11

3.2.1. Višekriterijsko filtriranje 11

3.2.2. Filtriranje na temelju sadržaja 13

3.2.3. Kolaborativno filtriranje 14

3.3. Postupak odabira lijekova s obzirom na mogućnosti i ograničenja pacijenta i lijeka 15

3.4. Programska arhitektura web sustava 17

3.5. Postupci ručnog i automatiziranog testiranja web sustava 18

3.5.1. Testiranje korisničkog sučelja 19

3.5.2. Stres test 20

3.5.3. Testiranje API-ja 20

4. PROGRAMSKO RJEŠENJE WEB SUSTAVA ZA STVARANJE PREPORUKA PRIKLADNIH LIJEKOVA 22

4.1. Korišteni programski jezici i alati 22

4.1.1. Korišteni programski jezici 22

4.1.2. Alati korišteni u izradi aplikacije 23

4.2. Programsko rješenje na strani korisnika 26

4.2.1. Naslovna stranica 26

4.2.2. Korisnički prikaz aplikacije 30

4.2.3. Administratorski prikaz aplikacije 37

4.2.4. Prikaz obavijesti 44

4.3. Programsko rješenje na strani poslužitelja 46

4.3.1. Ovjera korisnika 47
4.3.2. Relacije među tablicama korisnici, alergije, simptomi, pacijenti, lijekovi 50

5. TESTIRANJE WEB SUSTAVA ZA STVARANJE PREPORUKA PRIKLADNIH LIJEKOVA 59

5.1. Korišteni alati za testiranje 59

5.2. Testiranje korisničkog sučelja i testiranje korisničkog iskustva 59

5.3. Stres test 72

5.4. Testiranje API-ja 75

6. KORIŠTENJE I ISPITIVANJE WEB SUSTAVA ZA STVARANJE PREPORUKA PRIKLADNIH LIJEKOVA 81

6.1. Opis načina korištenja 81

6.1.1. Običan korisnik 81
6.1.2. Administrator kao korisnik 85

6.2. Uvjeti ispitivanja 88

6.2.1. Primjer korištenja 1 88
6.2.1. Primjer korištenja 2 89
6.2.1. Primjer korištenja 3 89

6.3. Rezultati ispitivanja s analizom 90

7. ZAKLJUČAK 93

LITERATURA 94

SAŽETAK 97

ABSTRACT 98

ŽIVOTOPIS 99

PRILOZI 100

1. UVOD

Zbog sve veće digitalizacije u svijetu moguće je pronaći korištenje raznih sustava, aplikacija i tehnologija općenito u svakodnevnom životu. Gotovo sve primjene takvih tehnologija su usmjerene na rješavanje određenih problema, olakšavanje života ili posla te pružanje raznih mogućnosti. Jedna od grana koja poprilično ovisi u prvom redu o računalnim tehnologijama, njihovoj točnosti i sigurnosti je medicina pa tako i predlaganje lijekova i liječenje pacijenata. Zbog toga je već sada u primjeni u raznim državama sve veći broj web sustava i aplikacija te raznih programa koji omogućuju lakše i brže liječenje pacijenata te predlaganje načina liječenja. Neki od takvih sustava će se u ovom radu analizirati, a jedan takav web sustav stvaranja preporuka prikladnih lijekova s obzirom na informacije o pacijentu će se izraditi u sklopu ovog diplomskog rada.

U ovom radu analizirat će se postupak predlaganja lijekova i postojeći sustavi slične namjene. Uz to, bit će prikazan postupak stvaranja web sustava za predlaganje lijekova od definiranja idejnog rješenja, arhitekture, postupaka testiranja i zahtjeva na sustav, kao i stvaranje programskog rješenja, provedba testiranja i analize.

Nakon analize postupaka i analize postojećih sustava za odabir prikladnih lijekova u drugom poglavlju, u trećem poglavlju prikazano je idejno rješenje sustava, definirani su funkcionalni i nefunkcionalni zahtjevi na sustava te objašnjeni postupci stvaranja preporuka, postupak odabira lijekova, programska arhitektura sustava i načini testiranja sustava koji će se primjenjivati u ovom radu. Nadalje u četvrtom poglavlju opisuju se ukratko korišteni alati i programski jezici koji će biti korišteni u izradi web sustava. Kroz primjere programskog koda, prikazano je i analizirano programsko rješenje na strani korisnika i poslužitelja. U petom poglavlju provedeno je testiranje ostvarenog web sustava, odnosno opisani korišteni alati te obavljeno testiranje korisničkog sučelja i korisničkog iskustva, stres test i testiranje API-ja, a rezultati testiranja analizirani. U šestom poglavlju dane su kratke upute za korištenje web sustava, definirani uvjeti ispitivanja sustava te prikazani rezultati tog ispitivanja s analizom.

2. SUSTAVI ZA ODABIR PRIKLADNIH LIJEKOVA NA TEMELJU PREPORUKA I PREGLED STANJA U PODRUČJU

Za razumijevanje problematike razvoja i testiranja web sustava za stvaranje preporuka prikladnih lijekova potrebno je znati što je lijek, na koji način se vrši odabira prikladnih lijekova i koji su problemi odabira. U nastavku su prikazani primjeri sustava za odabir lijekova i opisani potrebni postupci testiranja takvih sustava.

2.1. Problemi odabira lijekova

Znanost koja proučava tvari koje međusobno djeluju sa živim sustavima kroz kemijske procese se zove farmakologija pa tako u općenitom smislu, lijek bi se mogao definirati kao bilo koja tvar koja dovodi do izmjene u biološkim funkcijama kroz njezino kemijsko djelovanje [1]. Zbog tih izmjena u biološkim funkcijama organizma, problemi odabira prikladnih lijekova za određenog pacijenta se počinju ukazivati. Razne alergije na lijekova ili određene tvari u lijekovima otežavaju odabir i dodaju još jedan čimbenik na koji se mora paziti prilikom izdavanja lijekova za svakog pacijenta. Osim alergija mogu se pojaviti i razne nuspojave od uzimanja određenog lijeka koje mogu prouzrokovati daljnje bolesti a također nekada mogu biti vrlo opasne za zdravlje pacijent. Također, pogrešna doza lijeka, uzimanje u krivo vrijeme ili drugačiji način uzimanja lijeka može dovesti do nuspojava pa nekada i do smrti pacijenta u slučaju predoziranja. Ukratko, propisivanje krivog lijeka najčešće znači produljenje liječenja kao i povećanje troška liječenja, a nekada može dovesti do novih zdravstvenih problema.

Osim takvih problema od strane lijekova, doktori zbog gužvi u ordinacijama nekada se bore s nedostatkom vremena kojeg treba posvetiti svakom pacijentu i odabiru najbolje terapije liječenja te je tada nemoguće pratiti sve nuspojave lijekova i veliki broj interakcija među lijekovima. Također, doktori su često ograničeni određenim lijekovima koje mogu prepisati pacijentima bez da ih dodatno financijski opterete [2]. Osim interakcija s drugim lijekovima, lijekovi mogu također imati drugačije interakcije s drugim tvarima poput alkohola, dima cigareta, nutrijenata, hrane i sunca.

Nekada se problemi oko odabira lijekova za pacijenta mogu pojaviti i od strane pacijenta. Odluka o lijekovima koji bi se trebali prepisati pacijentu, kao i procjena o kakvoj se bolesti radi i kako najbolje postupiti, dolazi od anamneze pacijenta tako da je proces i odabir lijekova jednako važan i jednako kvalitetan kao što je i anamneza pacijenta. Anamneza je skup podataka o bolesniku koji sadrži sve informacije i okolnosti koje su prethodile trenutnom stanju [3, 4]. U njoj pacijent navodi povijest bolesti, koje lijekove uzima, nasljedne bolesti i dr.

Navedeni problemi prilikom odabira lijekova nekada dovode do medikacijske greške. Medikacijska greška se može definirati kao svaki događaj koji se može izbjeći a uzrokuje ili vodi do neodgovarajuće upotrebe lijeka ili do štete kod pacijenata, bili lijekovi upotrebljavani pod nadzorom liječnika ili samostalno od pacijenata [2].

Takva greška obilježava također kada:

- se prepiše pravi lijek, ali s pogrešnom dozom, načinom uzimanja ili je navedeno krivo vrijeme
- se daje pogrešan lijek za stanje pacijenta
- se prepiše pravi lijek koji s hranom ili drugim lijekovima ima štetne nuspojave
- prepisani lijek uzrokuje neželjene nuspojave

Medikacijska greška ne mora nužno uvijek biti greška liječnika koji izdaje lijek i način uzimanja, već može biti greška bilo kojeg drugog zdravstvenog radnika te pacijenta koji nepravilno uzima prepisani lijek.

Neki od rizičnih čimbenika za pogrešno prepisivanje, izdavanje i upotrebu lijekova su [5, 6, 7]:

1. Pogrešno izdavanje lijeka od strane farmaceuta
    - automatsko izdavanje lijekova
    - nedovoljno informiranje bolesnika o lijeku koji je dobio
    - pogrešno izdan lijek
2. Pogrešno prepisivanje lijeka od strane liječnika
    - neprikladno uzeta anamneza o lijekovima koje pacijent uzima ili je uzimao
    - dupliciranje lijekova
    - neprecizne upute o načinu uzimanja lijeka
    - neprikladno doziranje s obzirom na dob
3. Pogrešna upotreba lijeka od strane pacijenta
    - nekontrolirano uzimanje više lijekova
    - konzumacija alkohola uz konzumaciju lijekova te nepravilna ishrana
    - pogrešan izbor lijekova koji se kupuju bez recepta
    - neprikladno rečena anamneza doktoru o lijekovima koje uzima

2.2. Stvaranje preporuka za izbor prikladnih lijekova

U Sjedinjenim Američkim Državama provedeno je istraživanje koristi računalnih sustava za predlaganje i odabir lijekova prema kojemu su zaključili da se tijekom istraživanja, tj. tijekom korištenja takvog sustava došlo do smanjenja pogrešno propisanih lijekova za čak 81%. Propisivanje lijekova koji ozbiljno mogu naštetiti pacijentu je opalo za 86%. Velike razlike su se mogle vidjeti u svim glavnim tipovima grešaka kod izdavanja lijekova: pogrešne doze, pogrešno vrijeme uzimanja, pogrešni zamjenski lijekovi i alergije [8].

Kako bi se došlo do velikih smanjenja pogrešno propisanih lijekova, pogrešne doze, vremena uzimanja, pogrešnih zamjenskih lijekova te do alergija, potreban je određeni postupak stvaranja preporuka za izbor prikladnih lijekova. Takav postupak se mora temeljiti na određenim podacima i promatranju važnosti tih podataka kako bi se spriječilo predlaganje lijekova na koje bi pacijenti bili alergični, imali neželjene nuspojave i kontraindikacije.

Neki od današnjih sustava, koji će biti spomenuti kasnije, zahtijevaju od liječnika da unaprijed zna o kojoj se bolesti radi kod bolesnika prije korištenja sustava za preporuku lijekove, a zatim prema toj bolesti predlažu lijekove i način liječenja kao i dodatne informacije te za određena stanja, čak i fotografije kako to stanje izgleda kod pacijenta. Prema tome, potrebno je zaključiti da se stvaranje preporuka prikladnih lijekova zasniva na filtriranju lijekova prema određenim parametrima koji se mogu povezati s određenim lijekom. U ovom slučaju određeni parametri mogu biti bolesti, alergije te simptomi, no kod drugih ti parametri mogu biti sasvim drugačiji. Također, važno je napomenuti da niti jedan način stvaranja preporuka za izbor prikladnih lijekova nije u potpunosti točan, a vjerojatno niti neće biti i svugdje se mogu pojaviti greške te zbog toga se provode razna testiranja takvih sustava. Osim toga liječnici još uvijek pregledavaju predložene lijekove i odabiru lijek za kojeg misle da će najbolje djelovati na pacijenta i da će mu najbolje odgovarati te da neće imati nuspojave ili druge neželjene pojave. Čimbenik odabira, tj. provjere čovjeka je također ključan u odabiru prikladnog lijeka.

2.3. Značaj testiranja pri razvoju web sustava za preporuku prikladnih lijekova

Prema svim navedenim problemima odabira lijekova, može se zaključiti da je kreiranje jednog sustava za odabir prikladnih lijekova na temelju preporuka složen i zahtjevan posao sa stajališta kvalitetnog predlaganja lijekova. Sustav koji bi korisniku predlagao lijek s obzirom na određene simptome ne bi bilo teško razviti da se ne mora paziti na određene nuspojave, kontraindikacije, alergije pacijenata na određene lijekove, interakcije s drugim lijekovima i tvarima poput alkohola,

kvalitetno uzimanje anamneze od pacijenta kao i dovoljno informiranje pacijenta o određenom lijeku.

Također, treba pripaziti pri odabiru lijekova koji će se predložiti pacijentu kako se ne bi predlagali lijekovi koji imaju mali broj simptoma koji se podudaraju s unesenom anamnezom pacijenta te da se ne predlažu lijekovi koji mogu ozbiljno naštetiti pacijentu ispred nekih drugih lijekova koji mogu riješiti isti taj problem bez rizika po zdravlje pacijenta. Osim takvih lijekova, potrebno je obratiti pozornost i na predlaganje lijekova koji mogu izazvati ovisnost. U slučaju da bi se takvi lijekovi, koji mogu izazvati ovisnost i koji mogu naštetiti pacijentu, predlagali u sustavu, bilo bi dobro naglasiti pacijentu o kakvom se lijeku radi te koje su nuspojave kako bi mogao odlučiti je li mu uopće potreban takav lijek.

Zbog navedenih slučajeva može se primijetiti važnost testiranja takvog sustava za predlaganje prikladnih lijekova jer nisu svi pacijenti jednaki te se različitim pacijentima daju različiti lijekovi nekada za istu bolest. Prema tome, sustavi moraju biti testirani za različite slučajeve pacijenata kao i za rubne slučajeve ili čak za slučajeve koji se u stvarnosti ne bi mogli dogoditi kako bi mogli testirati i vidjeti kako se sustav ponaša u tim slučajevima. Testiranjem se u bilo kojem smislu i u bilo kojoj primjeni, smanjuje vjerojatnost pojavljivanja pogreške, što je ključno kada se radi o predlaganju lijekova i liječenju ljudi.

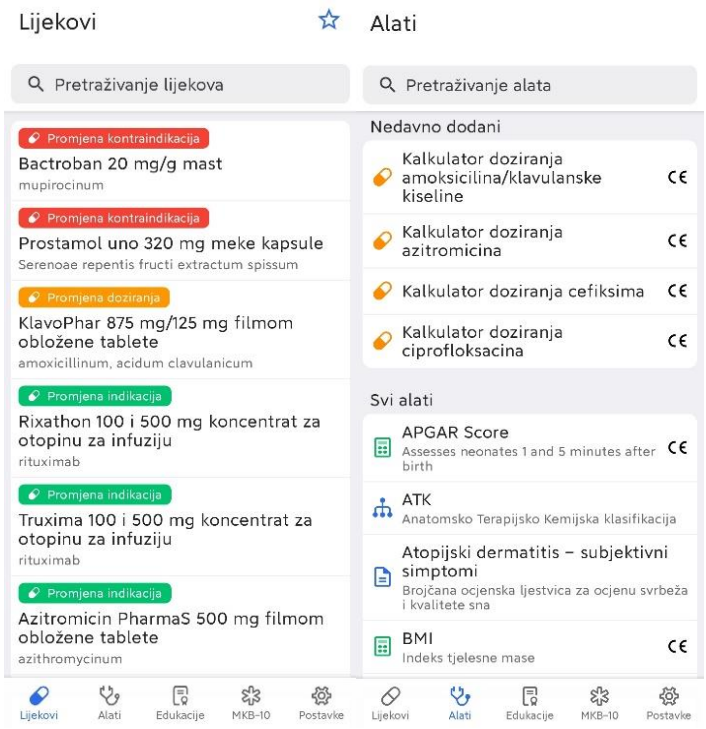
2.4. Postojeća rješenja za stvaranje preporuka prikladnih lijekova

S obzirom na navedene probleme odabira lijekova sada se može jasnije vidjeti važnost kao i koristi sustava koji bi olakšao uzimanje anamneze pacijenta te prema njoj predlagao lijekove za određenog pacijenta. Takvi sustavi već postoje i koriste se već godinama, a ovdje će biti spomenuti neki od sustava.

Neki od sustava koji se koriste danas za odabir lijekova su:

- Sustav Europske agencije za lijekove
- Baza lijekova Mediately
- Sustav Medscape za odabir lijekova prema simptomima

Na slici 2.1 prikazana je početna stranica s lijekovima, na kojoj se mog u vidjeti izmjene kontraindikacija, doziranja i indikacija za pojedine lijekove, i stranica s alatima aplikacije baza lijekova Mediately.

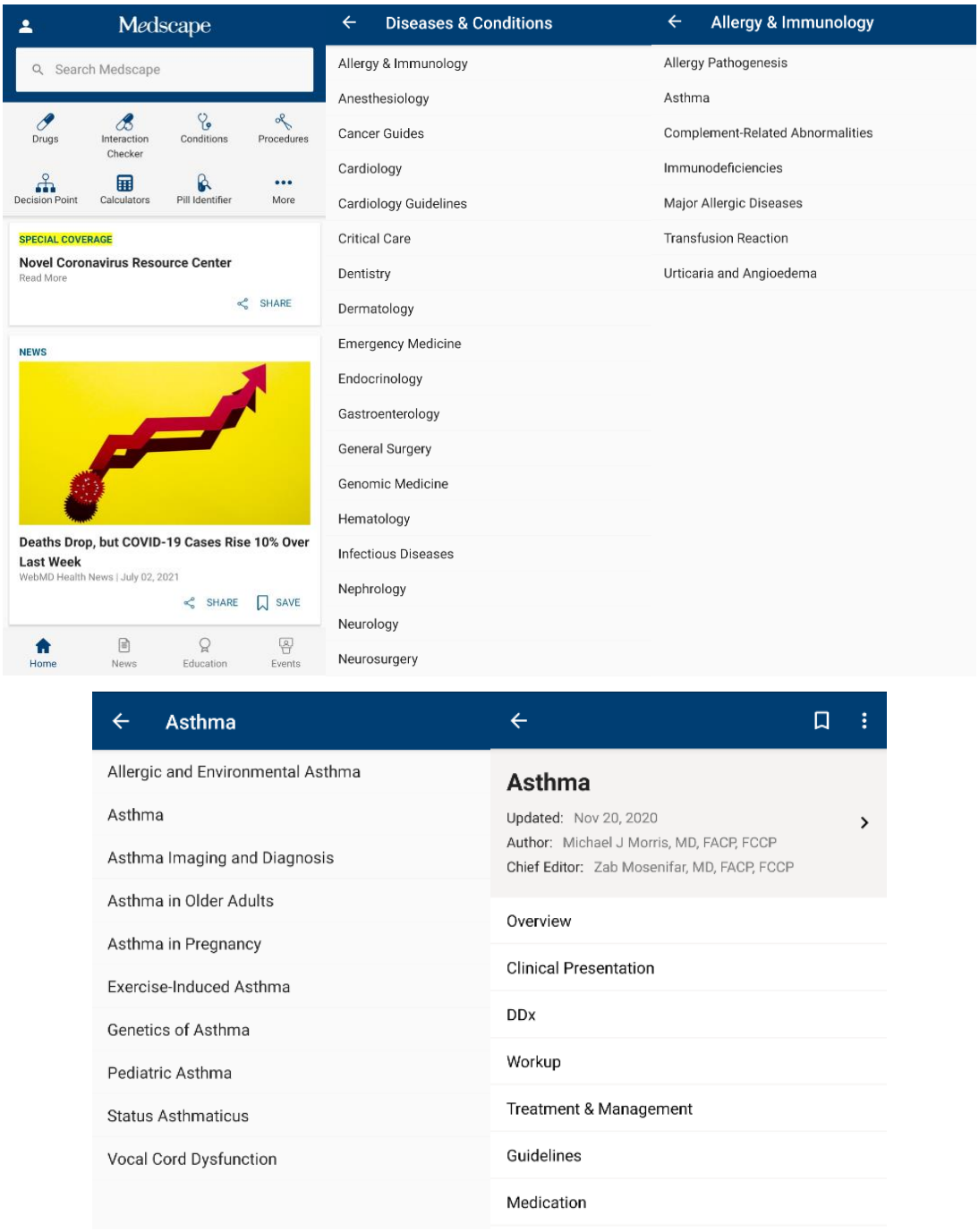


Sl. 2.1. *Prikaz aplikacije Baza lijekova Mediately* [9]

Određeni sustavi, u ovom slučaju baza lijekova Mediately i sustav Medscape, imaju inačice web sustava kao i aplikacije za mobilne uređaje, dok sustav Europske agencije ima samo web sustav.

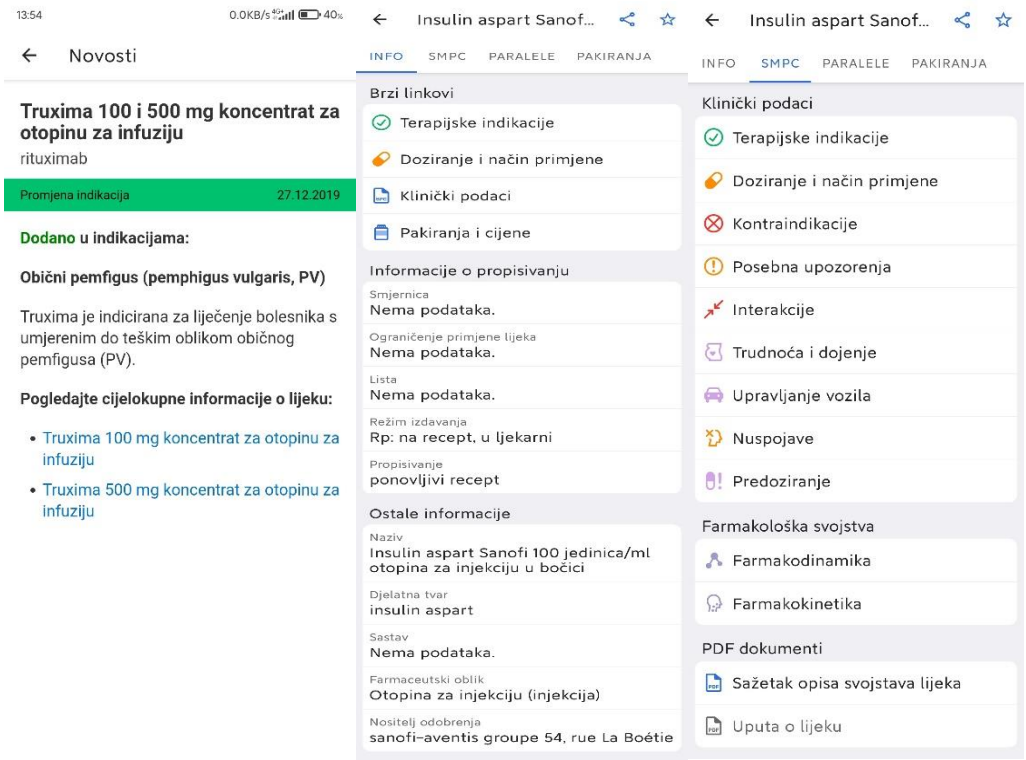
U sustavu Medscape se može pretraživati po područjima medicine i bolestima unutar tog područja gdje se dobiju načini liječenja, lijekovi te simptomi po kojima se može prepoznati određenu bolest. Kod sustava Medscape, prikazanog na slici 2.2, može se vidjeti proces odabira bolesti za koju se dobiju načini liječenja, lijekovi koji se mogu koristiti za liječenje te bolesti i druge kategorije, dok se kod baze lijekova Mediately, prikazane na slici 2.1 i 2.3, može pregledavati pojedine lijekove, njihove detalje i izmjene, interakcije s drugim lijekovima i tvarima, nuspojave, razne druge alate vezane za lijekove, razne kalkulatore i dr., ali u njoj se ne može pregledavati načini liječenja određenih bolesti.

Sustav Europske agencije za lijekove je sličan bazi lijekova Mediately u smislu da se pregledavaju lijekovi i informacije za koje se bolesti koristi određeni lijek, no on nema dodatne alate za lijekove poput kalkulatora za doziranje i dr. te nema ni informacije o nuspojavama, kontraindikacijama i interakcijama s drugim lijekovima kao što baza lijekova ima. Osim navedenih informacija, u sustavu Europske agencije za lijekove se također mogu pretraživati lijekovi po bolestima i lijekovi za životinje i biljke. Prema tome, može se reći da je sustav Europske agencije za lijekove više orijentiran pružanju osnovnih informacija o lijeku te pružanju informacije o tome smije li se lijek koristiti u Europskoj uniji.



Sl. 2.2. *Prikaz odabira bolesti i ponuđenih kategorija za bolest u Medscape aplikaciji* [10]

Još jedna sličnost baze lijekova Mediately i sustava Europske agencije za lijekove je to što se u oba sustava mogu vidjeti promjene informacija o lijekovima, izmjene njihove primjene, doziranja i dr. U baze lijekova Mediately, na slici 2.3, mogu se vidjeti promjene za određene lijekove te se mogu vidjeti i razne indikacije, kontraindikacije, nuspojave kao i interakcije s drugim tvarima i lijekovima, način primjene i doziranje lijeka. Također, važno je napomenuti da HALMED (Hrvatska agencija za lijekove i proizvode) koristi sustav Europske agencije za lijekove i preuzima podatke od njih te pruža informacije o lijekovima i druge podatke bazi lijekova Mediately.



Sl. 2.3. *Prikaz izmijene lijeka i detalja lijeka unutar aplikacije Baza lijekova Mediately* [9]

3. IDEJNO RJEŠENJE, MODEL I GRAĐA WEB SUSTAVA ZA STVARANJE PREPORUKA PRIKLADNIH LIJEKOVA

Za lakše razumijevanje i praćenje razvoja web sustava za preporuku prikladnih lijekova potrebno je definirati osnove na kojima će se zasnivati web sustav, poput postavljanja zahtjeva te postupaka stvaranja preporuka. Osim toga, dobro definirani zahtjevi, model i građa sprječavaju da se dodaju nepotrebne funkcionalnosti i usredotočuju na razvoj bitnih funkcionalnosti kao i na cjelokupni proces razvoja web sustava.

3.1. Funkcionalni i nefunkcionalni zahtjevi na web sustav

Funkcionalni i nefunkcionalni zahtjevi [11] spadaju pod prethodno navedene stavke čijim se definiranjem olakšava proces razvoja web sustava. Njihovim se definiranjem štedi vrijeme razvoja web sustava tako da se unaprijed zna koje funkcionalnosti se moraju nalaziti u sustavi i koji dodatni zahtjevi moraju biti zadovoljeni, a koji nisu smješteni unutar funkcionalnih zahtjeva. Osim toga, definiranjem ovih zahtjeva spriječit će se dodavanje nepotrebnih funkcionalnosti i dodatnih zahtjeva koji zapravo nisu potrebni.

3.1.1. Funkcionalni zahtjevi

Funkcionalni zahtjevi definiraju što sustav programske podrške treba raditi tako da svaki zahtjev definira funkciju sustava ili njegovog modula. Prema tome, kompleksnost sustava programske podrške se može dijelom odrediti po njegovim funkcionalnostima, a dijelom po njegovim nefunkcionalnim zahtjevima koji će biti spomenuti kasnije.

Funkcionalni zahtjevi web sustava koje će biti potrebno zadovoljiti su:

- Registracija i prijava korisnika
- Spremanje podataka o prijavi unutar kolačića
- Odjava korisnika kada istekne vrijeme tokena unutar kolačića
- Odvojeno sučelje za administratora i korisnika
- Unošenje pojedinosti o pacijentu od strane korisnika (alergija na lijekove, dob, spol)
- Unos simptoma pacijenta za prijedlog lijekova
- Unos, uređivanje i brisanje lijekova
- Uređivanje i brisanje korisnika od strane administratora
- Brisanje od strane administratora i pregled pacijenata od strane administratora i korisnika
- Prikladno rješenje za bazu podataka
- Dohvaćanje prijedloga lijekova za korisnika prema anamnezi

Registracija i prijava na web sustav predstavljaju početak sustava. Prilikom registracije korisnici će upisivati svoje podatke (ime, prezime, e-poštu i lozinku) unutar forme koja će se pritiskom na gumb obraditi i prema unesenim podacima će biti pokrenut proces kreiranja korisnika. Također, prilikom prijave korisnici će unositi svoje podatke (e-poštu i lozinku) kako bi pristupili sustavu i svom profilu. Lozinka korisnika će biti kriptirana, a podaci vezani za korisnika spremljeni u tokene koji će pri prijavi i registraciji biti predani korisniku. Tokeni će se na korisničkoj strani spremiti u kolačiće i uz to će biti omogućeno spremanje podataka lokalno na korisničkoj strani. Time će se izbjeći nepotrebna prijava korisnika pri svakom novom posjetu sustavu. Korisniku će se pri svakom posjetu sustavu pregledati token koji se nalazi unutar kolačića te provjeriti je li valjan ili ne. Ovisno o rezultati, korisnik će moći nastaviti koristiti sustav ili će biti odjavljen te će se trebati ponovno prijaviti u sustav.

Sustav će imati odvojeno sučelje za običnog korisnika i administratora. Korisnik će imati svoje sučelje pomoću kojega će moći unositi podatke za pacijenta poput dojenja ili trudnoće, dobi, alergija i uz to će, nakon unošenja podataka, moći pristupiti dijelu sustava za predlaganje lijekova s obzirom na anamnezu i postavljene postavke pacijenta. Unutar tog dijela sustava nalazit će se jedna vrsta forme koju će korisnik moći popuniti s podacima, a na pritisak gumba poslati zahtjev s unesenim podacima za dohvaćanjem prijedloga lijekova koji će vratiti odziv s poslužiteljske strane s popisom predloženih lijekova. S druge strane, administrator će prijavom ući u administratorsko sučelje gdje će imati pristup popisu korisnika, popisu lijekova kao i popisu unesenih pacijenata. Administrator će korisnike moći uređivati i brisati, lijekove će, uz uređivanje i brisanje s popisa, moći dodavati na popis dok će unesene pacijente moći pregledavati i brisati s popisa.

Do sada navedene funkcionalnosti zahtijevaju prikladnu bazu podataka koja će moći spremiti razne vrste podataka i koja će biti modelirana za odvojeno spremanje korisnika, lijekova, pacijenata, alergija, simptoma i relacija među njima u zasebne tablice. Svaki lijek će imati određene podatke poput simptoma, nuspojava, kontraindikacija i dodatnih informacija po kojima se može pronaći.

3.1.2. Nefunkcionalni zahtjevi

Nefunkcionalni zahtjevi su zahtjevi koji ne opisuju što će programska podrška raditi, već opisuju na koji će način programska podrška raditi [12, 13, 11]. Oni su uglavnom izvedeni iz funkcionalnih zahtjeva programske podrške jer je tako najbolje prikazano koji su dodatni zahtjevi potrebni s obzirom na funkcionalnosti. Neki od parametara koji govore kakav sustav treba biti su operativni

troškovi, performanse, pouzdanost, održivost, prenosivost itd. U predloženom sustavu trebat će se usmjeriti na nefunkcionalne zahtjeve koji će biti najpotrebniji s obzirom na funkcionalne zahtjeve.

Nefunkcionalni zahtjevi koje će biti potrebno ispuniti su:

- Sigurnost podataka
- Pouzdanost
- Prilagodljivost
- Proširivost

Prvi nefunkcionalni zahtjev koji se može primijetiti iz navedenih funkcionalnih zahtjeva je sigurnost podataka. Osjetljivi podaci korisnika, poput lozinke, trebaju biti kriptirani te pristup poslužiteljskoj strani, tj. krajnjim točkama poslužitelja, treba biti zaštićen od pristupa. U slučaju da treća strana želi pristupiti krajnjim točkama poslužitelja bez valjanog tokena pristupa, dobiti će grešku kao odgovor. Pouzdanost web sustava je također bitan zahtjev jer nije poželjno da usred korištenja web sustava korisnička ili poslužiteljska strana ispadne iz rada. Taj čimbenik će biti bitan kada se web sustav bude postavio na udaljenog poslužitelja jer lokalno izvođen gotovo nikada neće ispasti iz rada. Naposljetku, važno je spomenuti prilagodljivost i proširivost. Prilagodljivost web sustava u ovom slučaju bi bila da se može izvoditi na različitim platformama, tj. web preglednicima. Funkcionalnosti koje rade na jednom pregledniku trebaju raditi na drugom, a proširivost kao zahtjev će zahtijevati pisanje što više komponenti koje se mogu iskoristiti ponovno i na više mjesta te će se prilikom razvoja trebati pripaziti na omogućavanje dodavanja novih funkcionalnosti, poput dodavanja filtriranja lijekova prema bolestima ako bude potrebno.

3.2. Postupci stvaranja preporuka prikladnih lijekova

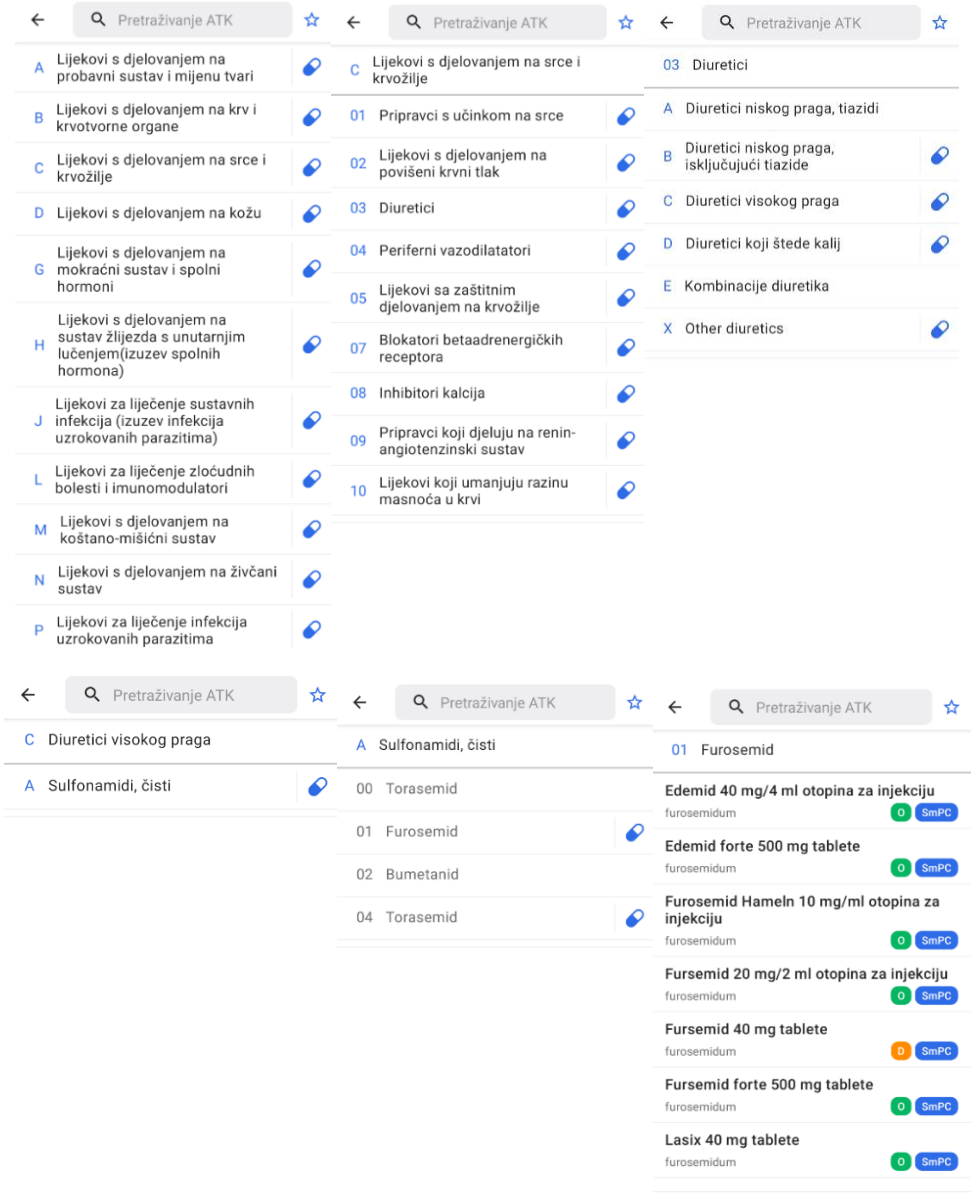
Prethodno navedeni zahtjevi kako bi se sustav trebao ponašati dovode do odabira načina pomoću kojega bi se trebalo moći predlagati lijekove na efikasan način s obzirom na anamnezu pacijenta kao i druge čimbenike.

3.2.1. Višekriterijsko filtriranje

Jedan od načina filtriranja se naziva višekriterijsko donošenje odluka i ono je jedno od najpoznatijih grana donošenja odluka. Po nekim autorima višekriterijsko donošenje odluka se može podijeliti na višeciljno donošenje odluka, višeatributno donošenje odluka ili višekriterijska analiza [14]. Ova dva navedena tipa se često koriste u označavanju iste klase modela [15]. Model višeciljnog odlučivanja je primjeren za dobro strukturirane probleme, tj. probleme kod kojih je poznato sadašnje stanje i željeno konačno stanje kao i način postizanja željenog stanja, dok je

model višeatributnog odlučivanja ili višekriterijske analize primjeren za loše strukturiranje probleme, tj. probleme kod kojih su ciljevi vrlo složeni te su često nejasno opisani, kod kojih postoje brojne neizvjesnosti [16].

Prema prikazanom primjeru filtriranja unutar aplikacije baze lijekova na slici 3.1 i prema opisu višekriterijskog odlučivanja može se zaključiti da je zapravo višekriterijsko odlučivanje najbolje rješenje za filtriranje sadržaja, tj. lijekova u ostvarenom web sustavu. Pomoću njega će se prema određenim simptomima i ograničenjima pacijenta, koji će u ovom slučaju predstavljati kriterije, odlučivati koje lijekove predložiti.



Sl. 3.1. *Prikaz filtriranja lijekova u Mediately bazi lijekova po određenim kriterijima* [9]

Kako bi se mogli lijekovi učinkovito predlagati, ograničenja pacijenta će morati biti prethodno postavljena kako bi se, prilikom dohvaćanja liste lijekova, ta ograničenja mogla uzeti u obzir i pomoću njih izvršiti početno filtriranje lijekova tako da se maknu lijekovi koji se ne bi trebali nalaziti u listi. Početno filtriranje, tj. filtriranje bez unesenih simptoma, uzeti će u obzir alergije i doji li pacijent ili je trudan, dok će se pravo filtriranje izvršiti kada se prilikom dohvaćanja liste lijekova predaju i simptomi po kojima će se dohvatiti lijekovi koji u sebi sadržavaju najmanje jedan simptom s liste te koji odgovaraju ograničenjima postavljenim za pacijenta.

Lista dohvaćenih lijekova će trebati biti sortirana po količini simptoma koje zadovoljava lijek s obzirom na predane simptome. Prema tome, lijek koji zadovoljava najviše simptoma će biti na prvom mjestu kako bi se izbjeglo ručno pretraživanje lijekova i uspoređivanje simptoma.

3.2.2. Filtriranje na temelju sadržaja

Sustavi preporuka temeljeni na sadržaju kreiraju preporuke na takav način da analiziraju sadržaj tekstualnih informacija i pronalaze pravilnosti u sadržaju [17]. Prema tome, za razliku od sustava s kolaborativnim filtriranjem koji koristi ocjene korisnika kako bi kreirao prijedloge i predviđanja, ovaj sustav se oslanja na sadržaj predmeta kako bi kreirao predviđanja [18].

Također, riječ „Sadržaj“ u nazivu se odnosi na opisne atribute predmeta koji se koriste za davanje preporuka. U metodama temeljenim na sadržaju, ocjene i ponašanje korisnika pri kupnji kombiniraju se s informacijama o sadržaju dostupnim u predmetu. Osim toga, u takvim metodama, opisi predmeta, koji su označeni ocjenama, se koriste kao podaci za obuku kako bi se stvorio problem klasifikacije koja je svakom korisniku jedinstvena ili problem regresijskog modeliranja. Za svakog korisnika podaci za obuku se podudaraju s opisima predmeta koje je kupio ili ocijenio. Takvi podaci za obuku se koriste za kreiranje klasifikacije ili regresijskog modela koji je jedinstven za svakog korisnika (ili aktivnog korisnika). Taj model koji je jedinstven korisniku se koristi za predviđanje hoće li se tom pojedincu svidjeti određeni predmet za koji nije poznata kupovina ili je nepoznata ocjena takvog predmet od strane korisnika [19].

Filtriranje na temelju sadržaja ima određene prednosti u kreiranju prijedloga za nove predmete kada nema dovoljno podataka o ocjenama za taj predmet jer postoji mogućnost da je aktivni korisnik ocijenio druge predmete sa sličnim atributima, tj. sa sličnim sadržajem. Zbog tih prednosti će nadzirani model moći koristiti ocjene drugih predmeta zajedno sa atributima novog predmeta kako bi davao preporuke čak i kada nema povijesti ocjena za taj predmet.

Također, uz prednosti se trebaju spomenuti i nedostatci takvog filtriranja, tj. metoda temeljenih na sadržaju. Neki od nedostataka su:

- zbog podataka vezanih svakog korisnika pri predlaganju smanjuje se raznolikost predloženih predmeta što je nepoželjno u npr. web trgovinama
- iako je ovo filtriranje efektivno u pružanje prijedloga za nove predmete, ono nije efektivno za pružanje prijedloga predmeta novim korisnicima

Zbog svega navedenog o filtriranju na temelju sadržaja može se zaključiti da ovakav način neće odgovarati u razvoju web sustava za stvaranje prijedloga prikladnih lijekova jer se ovdje ne radi o lijekovima koji se predlažu na temelju nekakvih prethodno propisanih lijekova kao što bi to bilo u nekakvoj web trgovini, već se trebaju predlagati lijekovi s obzirom na unesene parametre.

3.2.3. Kolaborativno filtriranje

Kolaborativno filtriranje je jedno od najuspješnijih pristupa izgradnji sustava preporuka. Ono radi tako da koristi poznate preferencije grupe korisnika za izradu preporuka ili predviđanja nepoznatih preferencija za druge korisnike [17].

Tipovi tehnika kolaborativnog filtriranja su [17]:

- Na temelju memorije
    - jednostavna implementacija, ne treba se uzimati u obzir sadržaj predmeta koji se predlaže, novi podaci se mogu dodati jednostavno i inkrementalno
    - performanse se smanjuju kada su podaci rjeđi, ovisno o ocjenama korisnika, ne mogu predlagati za nove korisnike i predmete
- Na temelju modela
    - bolje rješenje za rijetkost, skalabilnost i druge probleme, pruža intuitivno obrazloženje za prijedloge, poboljšava performanse predviđanja
    - mora se odabrati između skalabilnosti i performansi predviđanja, skupa izrada modela
- Hibridni preporučitelji
    - poboljšava performanse predviđanja, prelaze preko ograničenja kolaborativnog filtriranja, filtriranja na temelju sadržaja i drugih preporučitelja
    - potrebne eksterne informacije koje obično nisu dostupne, imaju povećan trošak i kompleksnost za implementaciju

Prema navedenom, mnogo je izazova za zadatke kolaborativnog filtriranja jer ne moraju svi korisnici sudjelovati u stvaranju preferencija. Iz tog razloga algoritmi kolaborativnog filtriranja moraju imati sposobnost načina rada s vrlo rijetkim podacima jer ih neće stvarati svaki korisnik. Osim toga, mora imati i sposobnost skaliranja sa sve većim brojem korisnika i predmeta, davanja zadovoljavajućih preporuka u kratkom vremenskom razdoblju i rješavanja drugih problema poput sinonimije (sklonost da isti ili slični predmeti imaju različite nazive), *shilling* napadi, podatkovni šum i problemi sa zaštitom privatnosti.

Shilling napad je napad na sustav preporuka s ciljem utjecanja na rangiranje preporuka kako bi se utjecalo na korisnički odabir [20]. Napad se također zove ubrizgavanje profila ili *profile injection.* Izvršava se tako da neki korisnici naprave lažne korisničke profile koji se sastoje od pristranih ocjena i time utječu na rangiranje preporuka i prikaz tih preporuka drugim korisnicima ispred nekih drugih koji bi bili prikladniji za njihove preferencije.

Prema tome, algoritmi za predlaganje često djeluju u izazovnim okruženjima, pogotovo kod velikih internet trgovina poput eBay i Amazon gdje je veliki naglasak na sprečavanju *shilling* napada.

Iz spomenutog opisa kolaborativnog filtriranja, zaključuje se da u ovom web sustavu neće biti potrebno izvesti ovakav način filtriranja lijekova jer se ne mogu povući direktne paralele između svakog pacijenta, tj. korisnika. Takvo predlaganje lijekova potencijalno bi moglo predložiti lijekove sličnom profilu pacijenta iako takav lijek za pojedinog pacijenta možda neće djelovati jednako, a mogu se pojaviti i drugi problemi poput nuspojava i kontraindikacija jer svaki organizam može drugačije reagirati na lijek.

3.3. Postupak odabira lijekova s obzirom na mogućnosti i ograničenja pacijenta i lijeka

Iz prošlog poglavlja, gdje se moglo vidjeti koji način filtriranja podataka je najbolje rješenje za web sustav, dalo se zaključiti da će u sustavu morati postojati parametri vezani za pojedinog pacijenta i pojedini lijek koji bi se ponašali kao ograničenja prema kojima će se predlagati lijekovi unutar web sustava.

Neki od najvažnijih parametara pacijenta, koji mogu utjecati na odabir lijeka, i lijeka, koji će se predložiti pacijentu, navedeni su u tablici 3.1:

Tab. 3.1. *Prikaz bitnih parametara koji utječu na odabir lijeka*

Pacijent	Lijek
Dob	Bolest za koje se koristi
Težina	Dob za koju se koristi
Trudnoća i dojene	Interakcije s drugim lijekova
Simptomi	Nuspojave
Koje druge lijekove koristi	Kontraindikacije
Alergije	Dodatne informacije

Najviše ograničenja pri odabiru lijeka nalazi se na strani pacijenta što i nije čudno jer se prema njegovim problemima, tj. određenim parametrima odabiru lijekovi koji bi bili dobri za rješavanje problema pacijenta.

Prema tome, dob, težina i trudnoća kao i dojenje bi bili najupečatljiviji parametri kod pacijenta. Pacijentu koji je mlađi od određene dobi za koji je predložen određeni lijek, koji je ispod potrebne težine ili pacijentu koji je trudan ili doji ne bi se trebali predlagati lijekovi koji zahtijevaju određenu težinu, dob ili da pacijent nije trudan jer takvi lijekovi mogu naštetiti pacijentu, izazvati nuspojave ili kontraindikacije kao i naštetiti još nerođenom djetetu.

Osim tih parametara, jedni od najvažnijih parametara na strani pacijenta bile bi alergije na određene tvari ili lijekove i simptomi koje pacijent posjeduje. Prema unesenim simptomima pacijentu će se predlagati lijekovi koji bi najviše odgovarali za te simptome, naravno, rijetko će se događati da će se simptomi savršeno podudarati sa simptomima bolesti koje određeni lijek liječi te će se morati na osnovu podudaranja dijela simptoma predlagati lijekovi. Uz to predlaganje, alergije na određene lijekove ili tvari će služiti kao pomoć u filtriranju lijekova koji će biti prikazani pacijentu. Osim toga, drugi lijekovi koje koristi pacijent također mogu pomoći oko filtriranja lijekova tako da se provjere s kojim lijekovima imaju neželjene interakcije i s obzirom na njihov odnos se prikazuju ili ne prikazuju takvi lijekovi.

Za razliku od pacijentske strane, potrebno je prema određenim parametrima i predlagati lijekove. Oni su zapravo analogni s parametrima pacijenta jer se preko njih traže paralele i usporedbe kako bi se predložili takvi lijekovi. S obzirom na to, lijek mora imati određene bolesti iz kojih će se uzeti simptomi za koje je djelotvoran te će se prema tim simptomima usporediti s korisničkim unosom simptoma pa će na takav način funkcionirati predlaganje lijekova. Zbog toga, ostali parametri koji su navedeni da su bitni za lijek, u velikoj mjeri odgovaraju korisničkom unosu.

Ostali parametri poput nuspojava, kontraindikacija i dodatnih informacija će biti predstavljeni korisniku koji bude odabirao lijek tako da može dodatno pročitati informacije te odlučiti rješava li odabrani lijek njegove probleme.

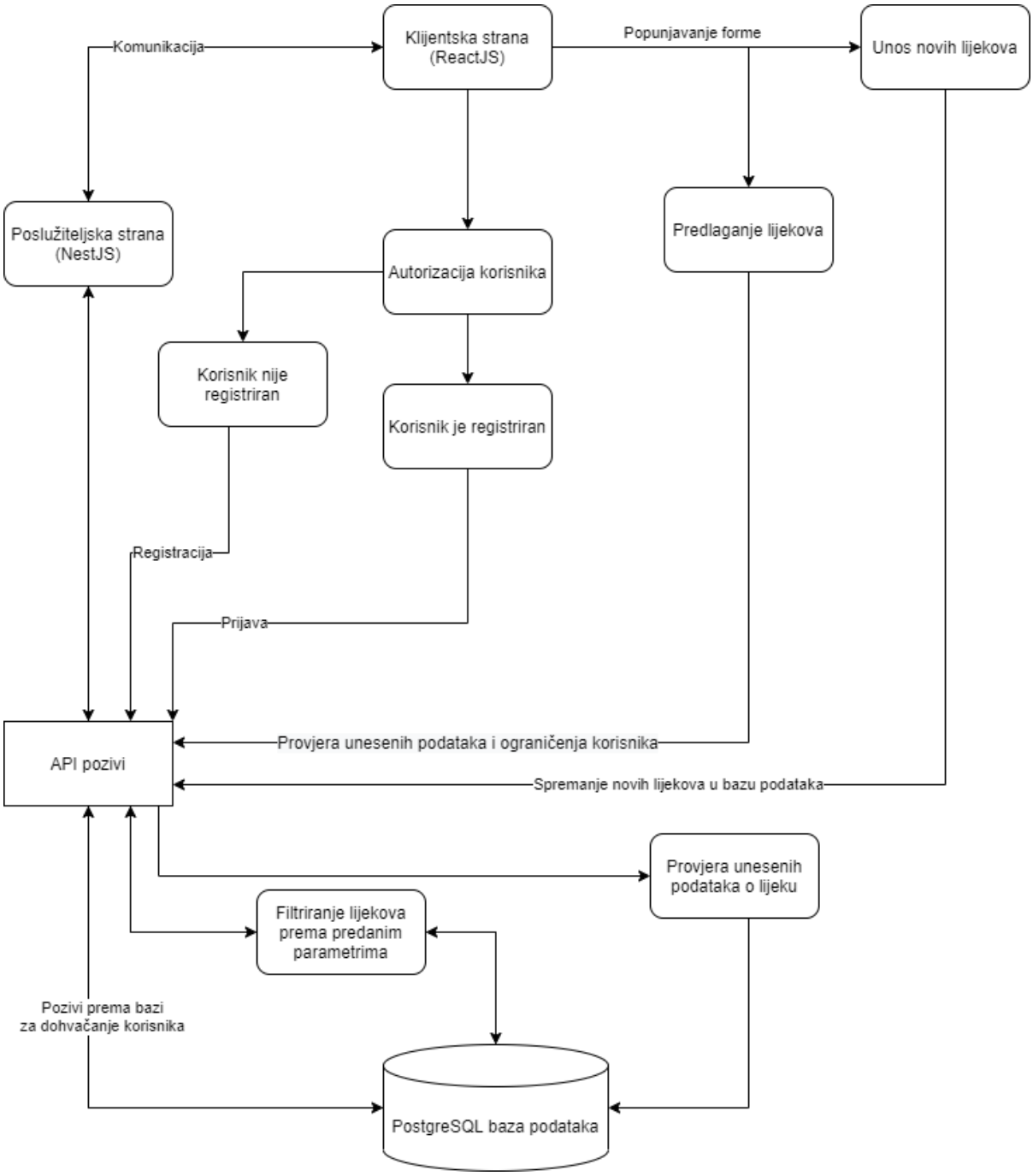
Prema svim navedenim parametrima, može se primijetiti da u teoriji postoji puno parametara koji su bitni pri odabiru prikladnog lijeka za pacijenta i da se neki od tih parametara moraju ručno gledati kako bi se mogao odlučiti prikladan lijek. Primjer tome bi bile interakcije lijekova i kontraindikacije napisane u opisu lijeka koje bi se pregledavale zasebno za svaki lijek. Zbog toga, u ovom web sustavu će se proširiti i zahvatiti što je više moguće parametara navedenih u tablici 3.1 kako bi sustav ostao što jednostavniji za upotrebu i za izradu.

3.4. Programska arhitektura web sustava

Primjer arhitekture web sustava prikazan je prema slici 3.2 blok dijagramom. Najveći dio posla u web sustavu prema blok dijagramu će snositi komunikacija između klijentske i poslužiteljske strane kao i API pozivi na poslužiteljskoj strani. Klijentska strana web sustava će pozivati funkcije s poslužiteljske strane koja će obavljati većinu posla i ovisno o funkciji koja je pozvana, spremati, uređivati, brisati ili dohvaćati određene podatke iz baze podataka. Podaci o završenom pozivu će biti vraćeni preko poslužiteljske strane klijentskoj strani koja će prema njima odrađivati daljnje akcije koje su potrebne.

Autorizacija korisnika obavljat će se na poslužiteljskoj strani tako da se spremaju novi korisnici u bazu podataka prilikom registracije, ako unesena e-pošta već ne postoji u bazi. Prilikom prijave provjeravat će se unesena e-pošta i lozinka koja će se kriptirati i usporediti s kriptiranom vrijednošću spremljenom unutar baze podataka te će se odgovarajući podaci o korisniku proslijediti natrag klijentskoj strani. U slučaju da e-pošta već postoji u bazi podataka prilikom registracije, vratit će se odgovarajuća greška s detaljima zašto nije moguće registrirati korisnika.

Korisnik će, nakon autorizacije, na web sustavu imati pristup formi za kreiranje pacijenta, predlaganje lijekova za pacijenta i popis prošlih pacijenata koje je unio. Pomoću formi za predlaganje lijekova i unos pacijenta unosit će se određeni podaci i parametri koji se potrebni za filtriranje lijekova, a zatim će se ti parametri proslijediti prema funkciji na poslužiteljskoj strani u obliku API poziva. Filtriranje će se izvršiti na poslužiteljskoj strani tako da se pretražuju lijekovi koji najviše odgovaraju predanim parametrima te će se takva lista predati natrag korisniku koji će moći pregledavati svaki od lijekova i čitati pojedinosti o njima.



Sl. 3.2. *Prikaz primjera arhitekture web sustava*

3.5. Postupci ručnog i automatiziranog testiranja web sustava

U prošlom poglavlju navedena je važnost testiranja web sustava i razlozi iz kojeg bi se trebalo testirati web sustav za predlaganje lijekova. Zbog toga, u ovom poglavlju će se govoriti o različitim načinima testiranja koji će biti provedeni nad web sustavom kako bi se osigurao ispravan rad te spriječilo ispadanje sustava iz rada. Testovi koji će se trebati provesti nad sustavom su ručno

testiranje klijentske i poslužiteljske strane kao i automatizirano testiranje obje strane pomoću različitih biblioteka.

Automatizirano testiranje se smatra bitnim dijelom svakog razvoja programske podrške. Automatizacija olakšava brzo i jednostavno ponavljanje pojedinačnih testova ili paketa testova tijekom razvoja. Uz to, ona također pomaže povećati pokrivenost koda testovima i pruža bržu povratnu informaciju programeru [21].

3.5.1. Testiranje korisničkog sučelja

Testiranje korisničkog sučelja će biti provođeno tijekom cijelog razvoja korisničkog sučelja web sustava. Najveći dio testiranja provodit će se ručno tijekom izrade pojedinih komponenti aplikacije, dok će se automatizirano testiranje provoditi nakon što se svaka od komponenti napiše i nakon što se testovi za tu komponentu budu također napisali. Kao rezultat testiranja dobiva se prikaz pokrivenosti koda testovima, pokrivenosti određenih linija koda te bolju i pouzdaniju aplikaciju.

Prema dokumentaciji ReactJS-a preporučeni alati za testiranje React aplikacije su Jest te React Testing Library. Ova oba alata uz Enzyme alat će se koristiti za testiranje korisničkog sučelja, a još neki drugi alati poput Istanbul-a će pratiti pokrivenost određenih linija koda testovima.

Jest alat služi da pokretanje testova te omogućuje pristup DOM-u kroz jsdom, a jsdom služi za simulaciju preglednika jer se testovi obično izvršavaju u okolini koja nema pristup pravoj površini za prikazivanje poput preglednika [22]. Također, Jest alat omogućuje pregled i generiranje pokrivenosti koda testovima, prikaz stanja testova te razloge zašto neki testovi nisu prošli te omogućuje jednostavno oponašanje implementacije funkcija. Za razliku od jsdom-a, DOM predstavlja Document Object Model i to je model aplikacije kojeg preglednik kreira prilikom učitavanja stranice te se sastoji od objekata koji predstavljaju dijelove aplikacije. S tim objektnim modelom JavaScript dobiva sve što mu je potrebno za stvaranje dinamičkog HTML-a [23].

Enzyme je uslužni program za JavaScript testiranje za React koji olakšava testiranje rezultata React komponenti. Također se može manipulirati, prelaziti i na neki način simulirati vrijeme izvođenja s obzirom na izlaz.

Korištenje svih ovih alata prilikom testiranja korisničkog sučelja će omogućiti bolji i pouzdaniji rad aplikacije i izbjegavanje problema u kodu koji se možda ne vide odmah prilikom pisanja koda. Uz to, pomoću ovih alata će se nadzirati rad svake pojedine komponente te da svaka komponenta radi kako je zamišljena pisanjem testova.

3.5.2. Stres test

Stres testiranje je oblik namjerno intenzivnog ili temeljitog ispitivanja koji se koristi za određivanje stabilnosti određenog sustava [24]. Takvo testiranje se provodi jer svaki web poslužitelj ima maksimalni kapacitet učitavanja i kada se taj teret prekorači, web poslužitelj počinje reagirati i proizvoditi greške. Svrha stres testa je pronaći maksimalno opterećenje koju web poslužitelj može podnijeti. Testira se trenutak pucanja sustava koji se namjerno izaziva velikim opterećenjem programske podrške. Time se utvrđuje maksimum rada sustava i postavljaju se ograničenja kako ne bi dolazilo do prelaska postavljenih ograničenja tijekom korištenja aplikacije.

Aplikacija koja se koristi za testiranje stresa je JMeter. Ona može otkriti maksimalni broj istodobnih korisnika s kojima se stranica može nositi. Pri tome nudi niz grafičkih analiza izvješća o izvedbi. U ovom web sustavu će se koristiti navedena aplikacija kada se sustav bude postavio na poslužitelja poput Heroku gdje će se testirati maksimalan broj istovremenih korisnika koji mogu posjetiti stranicu.

3.5.3. Testiranje API-ja

Prilikom razvoja poslužiteljske strane bit će provedeni testovi na sličan način kao kod testiranja korisničkog sučelja, no u ovom slučaju provodit će se provjere funkcionalnosti i sigurnosti krajnjih točaka. Provođenje takvog testiranja je potrebno kako bi bila osigurana funkcionalnost i pouzdanost rada krajnjih točaka. Važnost takvom testiranju također pridonosi to što će svi podaci unutar web sustava ovisiti o poslužiteljskoj strani. Zbog toga je bitno da poslužiteljska strana i pozivi krajnjih točaka na njoj rade kako su zamišljene. Prema tome, svrha testiranja poslužiteljske strane, tj. API-ja, je pronalazak novih informacija ili problema koji se nisu mogli primijetiti tijekom razvoja [25].

Testiranje poslužiteljske strane će se također odvijati pomoću alata Jest i SuperTest. NestJS ima ugrađenu podršku za navedene alate koji će pomoći u testiranju krajnjih točaka API-ja web sustava. Takvo testiranje će se provoditi automatizirano, pisanjem testova, i ručnim testiranjem korištenjem klijentske strane i posebnog alata za slanje zahtjeva.

Na slici 3.3 se može vidjeti primjer jednog paketa testova u kojem je definiran test za jednu krajnju točku na API-ju. Uz to, definirane su potrebne varijable za testiranje kao i akcije koje se izvršavaju prije i nakon pokretanja testa koje osiguravaju pravilno pokretanje, izvođenje i završavanje testova.

```
import * as request from 'supertest';
import { Test } from '@nestjs/testing';
import { CatsModule } from '../../src/cats/cats.module';
import { CatsService } from '../../src/cats/cats.service';
import { INestApplication } from '@nestjs/common';

describe('Cats', () => {
  let app: INestApplication;
  let catsService = { findAll: () => ['test'] };

  beforeAll(async () => {
    const moduleRef = await Test.createTestingModule({
      imports: [CatsModule],
    })
      .overrideProvider(CatsService)
      .useValue(catsService)
      .compile();

    app = moduleRef.createNestApplication();
    await app.init();
  });

  it(`/GET cats`, () => {
    return request(app.getHttpServer())
      .get('/cats')
      .expect(200)
      .expect({
        data: catsService.findAll(),
      });
  });

  afterAll(async () => {
    await app.close();
  });
});
```

Sl. 3.3. *Primjer testa modula i krajnje točke* [21]

4. PROGRAMSKO RJEŠENJE WEB SUSTAVA ZA STVARANJE PREPORUKA PRIKLADNIH LIJEKOVA

Kako bi se mogao razviti web sustav za stvaranje preporuka prikladnih lijekova s obzirom na razne parametre vezane za pacijenta, poput alergija na lijekove i njegove anamneze, mora se razdvojiti sustav na dio kojeg će korisnik vidjeti te na dio koji će sadržavati logiku registracije i prijave, unošenja lijekova i drugih akcija potrebnih za lijekove, filtriranja lijekova i dr. Prema tome, sustav se sastojati od korisničke strane koja je rađena u ReactJS, dok je poslužiteljska strana odrađena u NestJS i PostgreSQL. Uz navedene programske jezike u razvoju aplikacije se koristio Linux operacijski sustav.

4.1. Korišteni programski jezici i alati

Kao što je važno napraviti dobar odabir s programskim jezicima za izradu aplikacije, tako je jednako važno odabrati dobre alate za razvoj aplikacije koji mogu ubrzati i olakšati razvoj. Zbog toga je važno opisati i navesti neke mogućnosti alata i programskih jezika koji su bili korišteni u razvoju.

4.1.1. Korišteni programski jezici

ReactJS je JavaScript biblioteka koja se koristi za izradu korisničkih sučelja. Zasnovana je na ideji izrada zasebnih komponenti koje se brinu o svojim stanjima, a cilj je da komponente budu napisane tako da se mogu ponovno iskoristiti na različitim mjestima te da se mogu slagati s drugim komponentama u složenije komponente [22]. Također, React je jedna od najvećih biblioteka za izradu korisničkog sučelja s velikom zajednicom koja već dugo radi na toj biblioteci. Zbog tih posebnosti ReactJS je izabran za izradu korisničkog sučelja koje će biti raspodijeljeno na komponente, a uz to je i biblioteka već dovoljno zrela i razvijena te pruža mogućnosti za izradu takve aplikacije kao i provođenja *unit* testova pomoću Jest okvira, automatskog i ručnog testiranja korisničkog sučelja i dr.

NestJS je JavaScript okvir za izradu skalabilnih i učinkovitih Node.js aplikacija na poslužitelju. Koristi progresivni JavaScript te je izrađen da potpuno podržava TypeScript i objedinjuje elemente objektno orijentiranog programiranja, funkcijskog programiranja i funkcijsko reaktivnog programiranja [21]. NestJS je izabran jer je jedan od najbrže rastućih okvira za izradu poslužiteljske strane te ima odličnu dokumentaciju kao i veliku zajednicu koja radi na ovome okviru. Uz njega odabran je PostrgeSQL sustav za upravljanje bazama podataka jer je otvorenog koda, razvija se već dugo godina, pouzdan je i ima veliku zajednicu koja stoji iza njega te koja može pomoći u slučaju da je potrebna pomoć ili odgovor na neko pitanje.

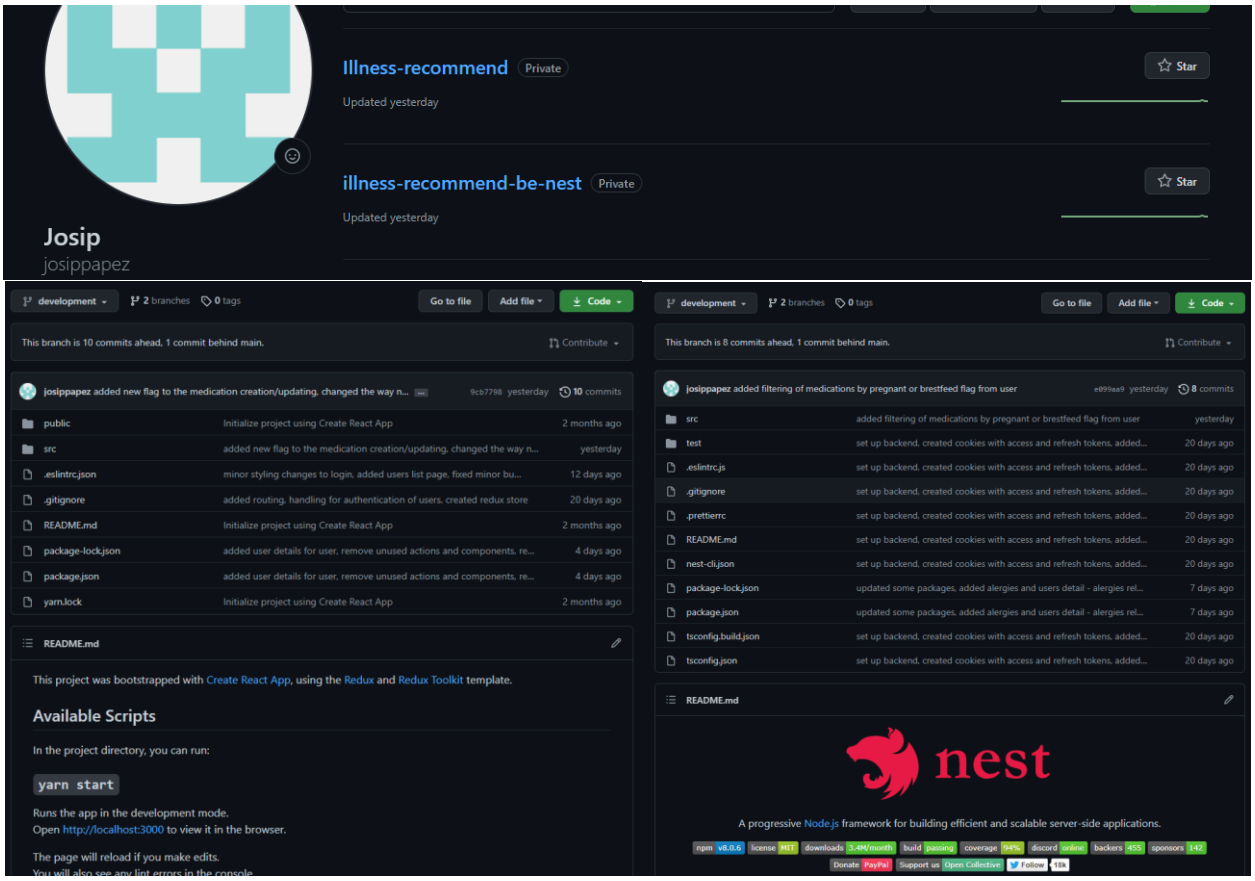
4.1.2. Alati korišteni u izradi aplikacije

Prvi i najbitniji alat za izradu koji se koristio za razvoj poslužiteljske i klijentske strane je Visual Studio Code. Ovaj uređivač izvornog koda, razvijen od tvrtke Microsoft, je dostupan za Windows, Linux i macOS razvojni sustav. U sebi ima ugrađenu podršku za JavaScript, TypeScript i Node.js te ima bogati sustav proširenja za druge jezike te za dodatna poboljšanja u smislu funkcionalnosti. Zbog ugrađene podrške za JavaScript, TypeScript i proširenja koja će olakšati pisanje koda i razvoj aplikacije, Visual studio Code se čini kao dobar odabir jer pokriva gotovo sve što je bilo potrebno za razvoj aplikacije. U programu postoje mnogo ugrađenih funkcionalnosti koje će ubrzati pisanje koda. Neke od ugrađenih funkcionalnosti koje su se koristile:

- Visual Studio IntelliCode/IntelliSense – pruža isticanje sintakse i pametno automatsko dovršavanje linija koda s obzirom na vrstu varijabli, definicija funkcija i uvezenih modula, također pruža opise određenih funkcija te prikaz parametara koji se predaju funkcijama
- Emmet – alat za web programere koji ubrzava pisanje HTML i CSS koda pretvarajući kratice u fragmente koda
- Navigacija koda i datoteka – pruža brzi odlazak na određenu datoteku ili na implementaciju/definiciju određene funkcije
- Verzioniranje koda – zasebna kartica u kojoj se mogu pratiti izmjene u kodu i datotekama, spremiti izmjene, pregledati detalje o grani i sve uz repozitorij u kojem se trenutno radi, dodati komentare te spremiti ih na jedan od pružatelja za verzioniranje koda pomoću Git-a poput GitHub, Bitbucket, GitLab i drugih
- Terminal – pruža mogućnost izvršavanja naredbi koje su potrebne za pokretanje aplikacije, spremanje izmjena i dodavanje novih modula, testiranje i mnoge druge

Uz navedene ugrađene alate, korištena su proširenja za manja poboljšanja u pisanju koda poput raznih isječaka koji omogućuju brzo pisanje velike količine ponavljajućeg koda, razna proširenja za uljepšavanje i održavanje forme koda kako bi osigurali pisanje jednakog i čitljivog koda, proširenje za pokretanje lokalnog poslužitelja i još manji broj alata koji olakšavaju pisanje koda i snalaženje u njemu.

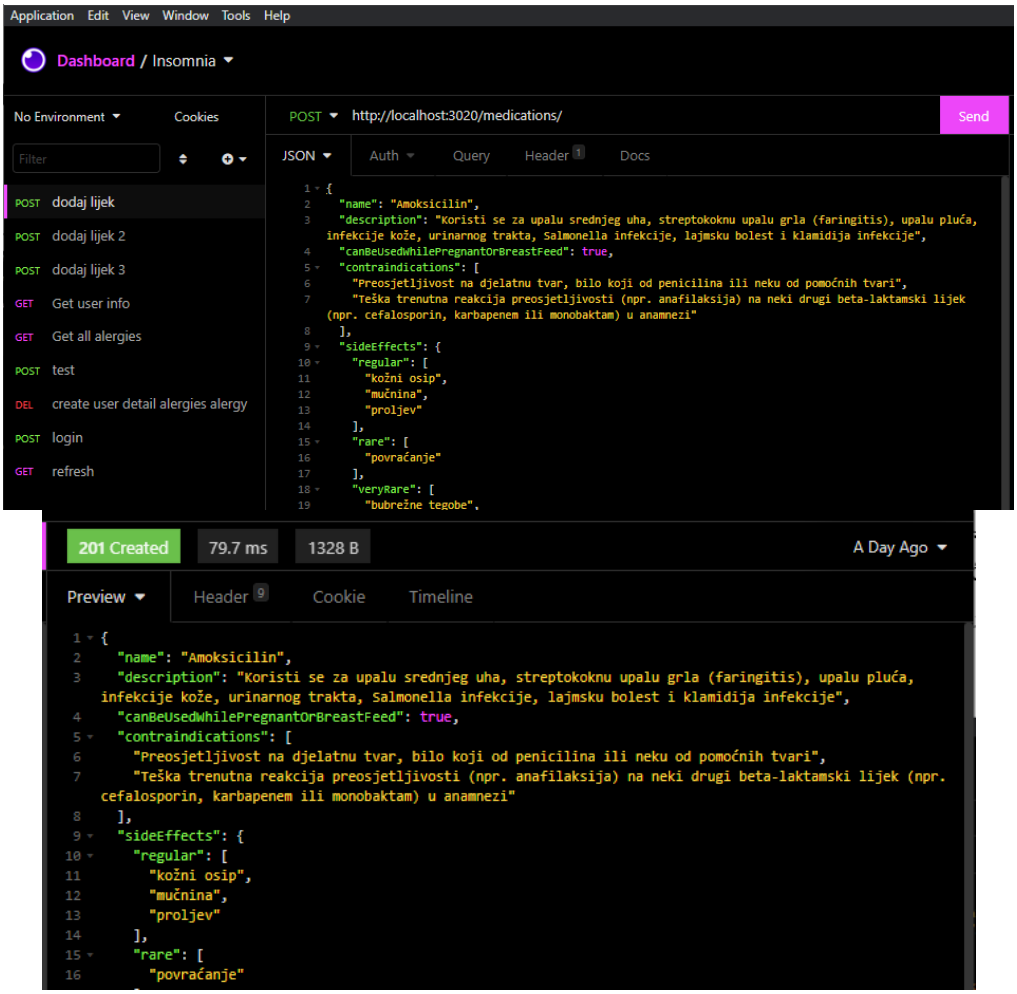
Drugi alat koji je jednako bitan kao i prvi je GitHub. To je online pružatelj za verzioniranje koda pomoću Git-a. Na njemu je spremljena povijest izmjena i trenutno stanje koda. Klijentska i poslužiteljska strana imaju zaseban repozitorij na kojem su spremljene kao što je vidljivo na slici 4.1 gdje su spremljeni kao privatni repozitoriji.



Sl. 4.1. *Prikaz repozitorija klijentske i poslužiteljske strane*

Treći alat korišten za testiranje poziva i izvođenje poziva prema poslužiteljskoj strani se naziva Insomnia. Insomnia je klijent i alat za dizajn suradničkog API-ja [26]. Pomoću njega su testirani pozivi prema poslužiteljskoj strani i provjeravani odzivi te njihova valjanost kao što je vidljivo na slici 4.2.

Uz alata Insomnia korišten je i alat DBeaver. DBeaver je besplatni alat za baze podataka za programere, administratore baze podataka, analitičare i ostale koji trebaju raditi s bazama podataka. Podržava sve popularne baze podataka i može se koristiti na više platformi poput Windows, Mac i Linux operacijskog sustava [27]. On omogućuje direktno uređivanje baze podataka kroz grafičko sučelje i omogućuje spajanje na više različitih baza, stvaranje sigurnosnih kopija baze, prebacivanje podataka između različitih baza i dr.



Sl. 4.2. *Prikaz alata Insomnia s listom zahtjeva te prikazom poziva i odziva*

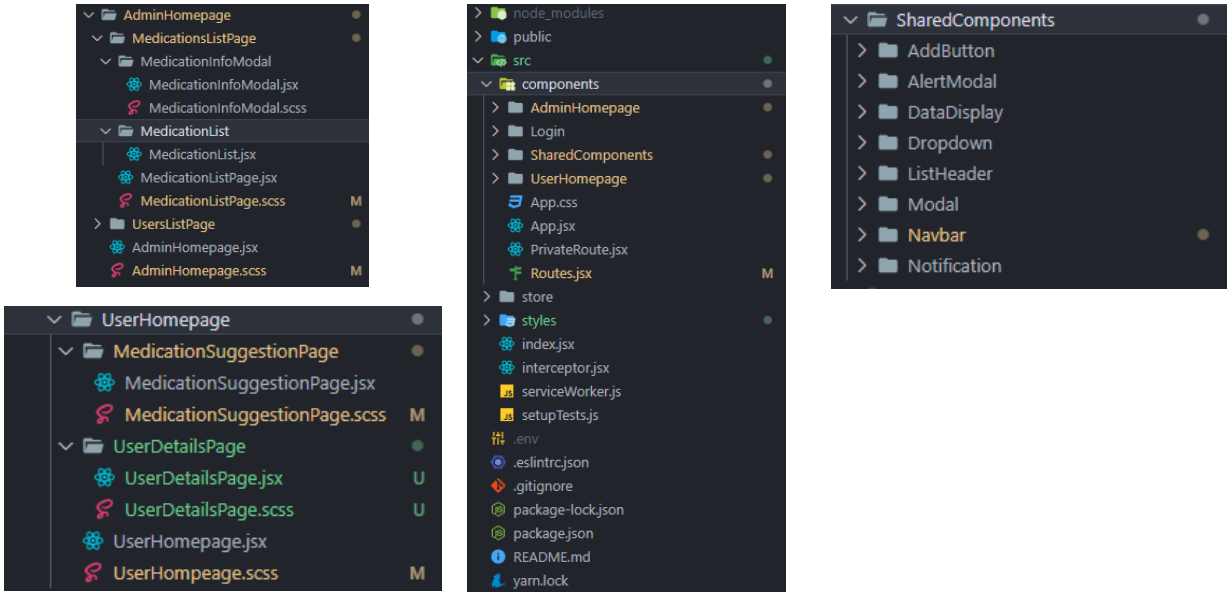
Osim GitHub-a, koristili su se mnogi drugi online alati i stranice poput SVGator za kreiranje animacija unutar svg slika, Method Draw Vector Editor za kreiranje i uređivanje svg slika, Flat icon i unDraw stranice za preuzimanje besplatnih svg slika.

Sl. 4.3. *Prikaz Method Draw Vector Editor stranice za kreiranje i uređivanje svg slika*

Naposljetku za stres test aplikacije koja je postavljena na udaljenog poslužitelja koristio se JMeter. Nije bilo potrebe ranije koristiti alat jer se ne ispituje koliko stranica može podnijeti zahtjeva dok je pokrenuta lokalno, već dok je pokrenuta na udaljenom poslužitelju.

4.2. Programsko rješenje na strani korisnika

Aplikacija na korisničkoj strani podijeljena je u komponente gdje svaka komponenta predstavlja jedan manji ili veći dio stranice. Tako se u aplikaciji koristi komponenta za prikaz teksta i podataka, kao što je prikazano na slici 4.6, gdje naziv polja za unos i polje za unos predstavljaju jednu komponentu. Prema tome, aplikacija je podijeljena na prikaz prijave i registracije, korisnički prikaz i prikaz administratora. Svaki od prikaza sadrži u sebi komponente koje se vezane za taj određeni prikaz poput liste korisnika koja je vezana za administratorski prikaz stranice. Osim takvih komponenti, postoje i zajedničke komponente od kojih se neke koriste među svim navedenim prikazima poput Datadisplay komponente koja služi za prikaz podataka i naslova vezanih za podatke.



Sl. 4.4. *Prikaz strukture projekta i komponenti na klijentskoj strani*

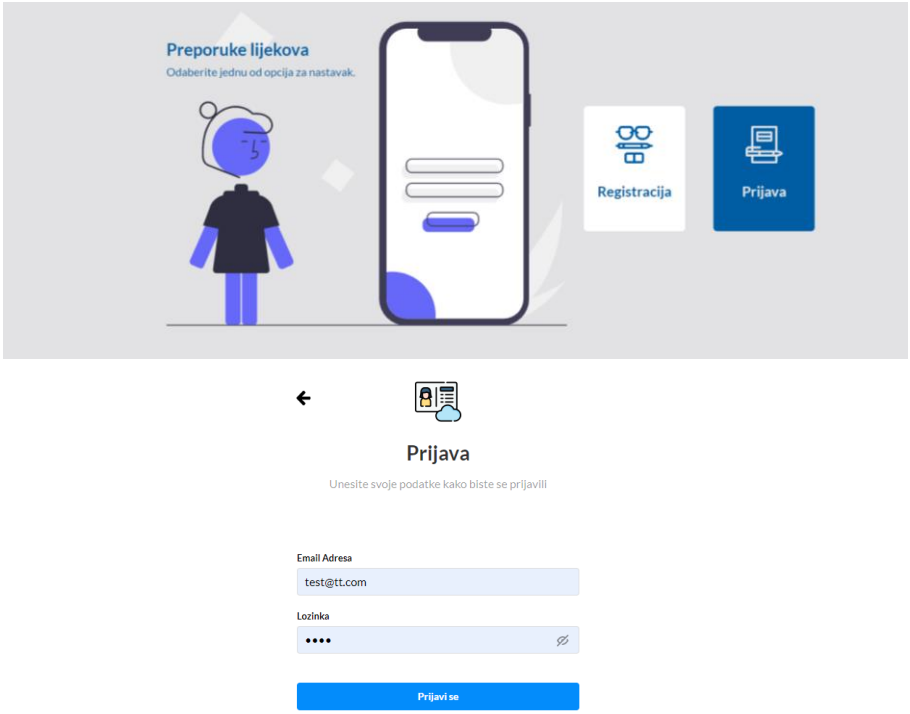
4.2.1. Naslovna stranica

Dolaskom korisnika na naslovnu stranicu prikazuje mu se odabir za registraciju ili prijavu u obliku dvije tipke pomoću kojih dolazi do odgovarajućih formi. Prema slici 4.6 prikazan je izgled naslovne stranice i forme za prijavu, a prema slici 4.5 se može vidjeti prikaz koda koji predstavlja prikaz tipki na naslovnoj stranici.

```jsx
<div className="login__button-container">
  <button
    className="login__button-container__register-btn"
    type="button"
    onClick={() => {
      setShowRegistration(true);
      setShowForm(true);
    }}
  >
    <div className="login__button-container__register-btn__text">
      Registracija
    </div>
  </button>
  <button
    className="login__button-container__login-btn"
    type="button"
    onClick={() => setShowForm(true)}
  >
    <div className="login__button-container__login-btn__text">
      Prijava
    </div>
  </button>
</div>
```

Sl. 4.5. *Prikaz tipki za prikaz forme za prijavu ili registraciju*

Forme se prikazuju tako da se stanje prikaza forme promjeni pritiskom na određenu tipku te se padajućom animacijom prikazuje forma. Prema slici 4.5 se može vidjeti da svaka tipka u sebi ima *onClick* funkciju koja osluškuje akciju pritiska na tipku i izvršava određeni kod koji je postavljen unutar poziva te akcije. U ovom slučaju se prilikom pritiska na tipku „Registracija“ postavljaju *showRegistration* i *showForm* stanja u istinito što omogućava prikazivanje forme.



Sl. 4.6. *Prikaz naslovne stranice i forme za prijavu*

Unutar gotovo svakog HTML-u elementa mogu se osluškivati razne akcije koje korisnik izvršava te se pritisak na tipku, unos teksta u polje, prelaskom miša preko elementa ili neka druga funkcija

izvršava na sličan način gdje se unutar nje mogu uhvatiti podaci o događaju i obraditi na željeni način.

Prema tome, ispunjavanjem forme, tj. upisivanjem podataka u polja za unos, spremaju se podaci u odgovarajuća stanja prikazana na slici 4.7 koja se, pritiskom na tipku „Prijavi se“ ili pritiskom na tipku Enter na tipkovnici, predaju funkciji u formi. Funkcija u formi šalje poslužiteljskoj strani podatke na obradu gdje, s obzirom na podatke, ona vraća odziv s korisnikom i tokenima koji sadrže podatke za pristup stranici i krajnjim točkama.

```
function Login() {
  const user = useSelector(state => state.user);
  const errorMessage = useSelector(state => state.user.error);
  const dispatch = useDispatch();

  const [name, setName] = useState(null);
  const [email, setEmail] = useState(null);
  const [password, setPassword] = useState(null);
  const [showForm, setShowForm] = useState(null);
  const [showRegistration, setShowRegistration] = useState(null);
  const [displayPassword, setDisplayPassword] = useState(false);
  const buttonRef = useRef(null);

  useEffect(() => {
    if (
      user &&
      user.id &&
      Cookies.get('Refreshtoken') &&
      Cookies.get('Accesstoken')
    ) {      You, 2 weeks ago • added login page styling and added rou
      window.location.replace('/');
    }
  }, [user]);
```

```
export const loginUser = (email, password) => {
  return (dispatch, getState) => {
    axios({
      method: 'POST',
      url: `${process.env.REACT_APP_API_URL}/authentication/log-in`,
      withCredentials: true,
      data: {
        email,
        password,
      },
    })
      .then(response => {
        dispatch(userLoggedIn(response.data));
      })
      .catch(error =>
        dispatch(
          loginError({
            message: error.response.data.successMessage,
            error: error.response.data,
            status: error.response.status,
          })
        )
      );
  };
};
```

Sl. 4.7. *Početna stanja unosa i prikaza formi te prikaz funkcije koja izvršava prijavu*

Nakon primanja točnog odziva od poslužiteljske strane, korisnika se sprema u spremnik stanja iz kojeg se mogu kasnije povlačiti podatci o korisniku i drugi podaci. Uz podatke iz odziva dobivaju se tokeni koji su potrebni za pristup stranici te izvršavanje poziva prema krajnjim točkama. Tokeni se prilikom valjanog odziva za prijavu spremaju u kolačiće koji se nalaze na klijentskoj strani i šalju se sa svakim budućim zahtjevom u zaglavlju s oznakom *Authorization* kako bi poslužiteljska strana mogla znati podatke o korisniku i, prema tome, poslati odgovor.

Sl. 4.8. *Prikaz odziva zahtjeva za prijavu*

Također, takvi spremljeni kolačići se koriste za pristup stranici te za provjeru smije li određeni korisnik pristupiti i vidjeti stranicu. Primjer provjere kolačića je prikazan na slici 4.7 gdje se

provjerava postoji li spremljeni korisnik u lokalnom spremniku stanja i postoje li odgovarajući kolačići koji su dobiveni kao odziv na prijavu ili registraciju, a ako su oba zahtjeva zadovoljena, korisnik neće moći pristupiti stranici za prijavu već će biti odmah preusmjeren na početnu stranicu.

Provjera kolačića izvršava se pomoću paketa JavaScript Cookie koji pruža API pomoću kojeg se rukuje s kolačićima. Osim provjere postoji li kolačić, moguće je obrisati i spremiti kolačić pomoću istog API-ja.

Postoji i provjera prilikom naknadnih pristupa stranici, tj. kada se korisnik vraća na stranicu nakon nekog vremena prikazana na slici 4.9. Prilikom te provjere se provjerava postoji li korisnik u spremniku stanja, postoje li kolačići te je li kolačić za pristup stranici istekao. U slučaju da ne postoji korisnik u bazi, da ne postoji kolačić za pristup stranici ili da je kolačić za obnovu tokena, pod nazivom „Refreshtoken“, za pristup stranici istekao, onda se korisniku odbija pristup stranici, svi lokalno spremljeni podaci će biti obrisani i korisnik će biti preusmjeren na naslovnu stranicu prije nego što mu se prikaže ostatak stranice. U slučaju da postoji token za obnovu pristupa koji nije istekao i u slučaju da postoji korisnik u bazi kao i token za pristup stranici, onda se korisniku prikazuje stranica i dio stranice kojoj želi pristupiti.

```
const PrivateRoute = props => {
  const user = useSelector(state => state.user);
  const [response, setResponse] = useState(null);
  const [alreadyFetched, setAlreadyFetched] = useState(null);

  useEffect(() => {
    renderFunc();
  }, []);

  const renderFunc = async () => {
    let accessToken;
    const accessTokenCookie = Cookies.get('Accesstoken');
    const refreshTokenCookie = Cookies.get('Refreshtoken');
    if (accessTokenCookie) {
      accessToken = token.decode(accessTokenCookie).exp;
    }
    if (
      (refreshTokenCookie &&
        token.decode(refreshTokenCookie).exp <= moment.utc().unix()) ||
      !accessTokenCookie ||
      !user.id
    ) {
      logOutAndWipeLocalStorage();
      return;
    }
    if (accessToken && accessToken > moment.utc().unix()) {
      setResponse(true);
      setAlreadyFetched(false);
    } else if (!alreadyFetched && refreshTokenCookie) {
      const response = await refreshAuthentication();
      if (response.status !== 200) {
        logOutAndWipeLocalStorage();
      }
      const newToken = Cookies.get('Accesstoken');
      const decodedToken = token.decode(newToken);
      if (decodedToken == null || decodedToken.exp <= moment.utc().unix()) {
        logOutAndWipeLocalStorage();
      }
      setAlreadyFetched(true);
      renderFunc();
    } else {
      logOutAndWipeLocalStorage();
    }
  };
```

```
  return response ? (
    <>
      <Navbar />      You, 3 days ago • added not
      <div className="page-container">
        <div className="navbar-container"></div>
        <div className="page-content">
          <Route {...props} />
        </div>
      </div>
    </>
  ) : response === false ? (
    <Redirect to="/login" />
  ) : null;
};
```

Sl. 4.9. *Prikaz komponente za provjeru prilikom pristupa stranici*

Naposljetku, ako postoji korisnik u bazi stanja, token za obnovu pristupa koji nije istekao i token za pristup stranici koji je istekao, tada će se pokušati obnoviti token za pristup stranici prije nego što se prikaže stranica kojoj korisnik želi pristupiti. U slučaju da obnova tokena bude uspješna, korisniku se prikazuje stranica, no ako obnova bude neuspješna ili ako se dobije novi token za pristup stranici koji je istekao, u tom slučaju se korisniku odbija pristup i brišu se svi lokalno spremljeni podaci te se korisnik preusmjerava na naslovnu stranicu.

4.2.2. Korisnički prikaz aplikacije

Korisnička strana sadrži odvojeni prikaz za običnog korisnika i administratora. Običan korisnik ulaskom, tj. prijavom ili registracijom, na stranicu ima pristup svojim korisničkim detaljima te prijedlogu lijekova na navigacijskoj traci prikazanoj na slici 4.10.

Sl. 4.10. *Prikaz početne stranice korisnika nakon prijave*

4.2.2.1. Detalji o pacijentu

Prelaskom na dio stranice s detaljima o pacijentu, korisniku se prikazuje forma za unos pojedinosti. Na toj formi korisnik može postavljati vrijednosti od kojih će neke utjecati na prijedlog lijekova za pacijenta, a neke vrijednosti na prikaz podataka za pojedini lijek. Na slici 4.12 se može vidjeti navedena forma gdje se unosi broj godina, težina i alergije pacijenta kao i podataka doji li pacijent ili je trudan. Odabrane alergije i podatak doji li pacijent ili je trudan će utjecati na lijekove koji će biti predloženi na dijelu stranice za predlaganje lijekova. Alergije se dodaju na listu za pacijenta tako da se pritiskom na unos s tekstom „Odaberi ili upiši“ odaberu odgovarajuće alergije s padajućeg izbornika koje se odnose na unesenog pacijenta.

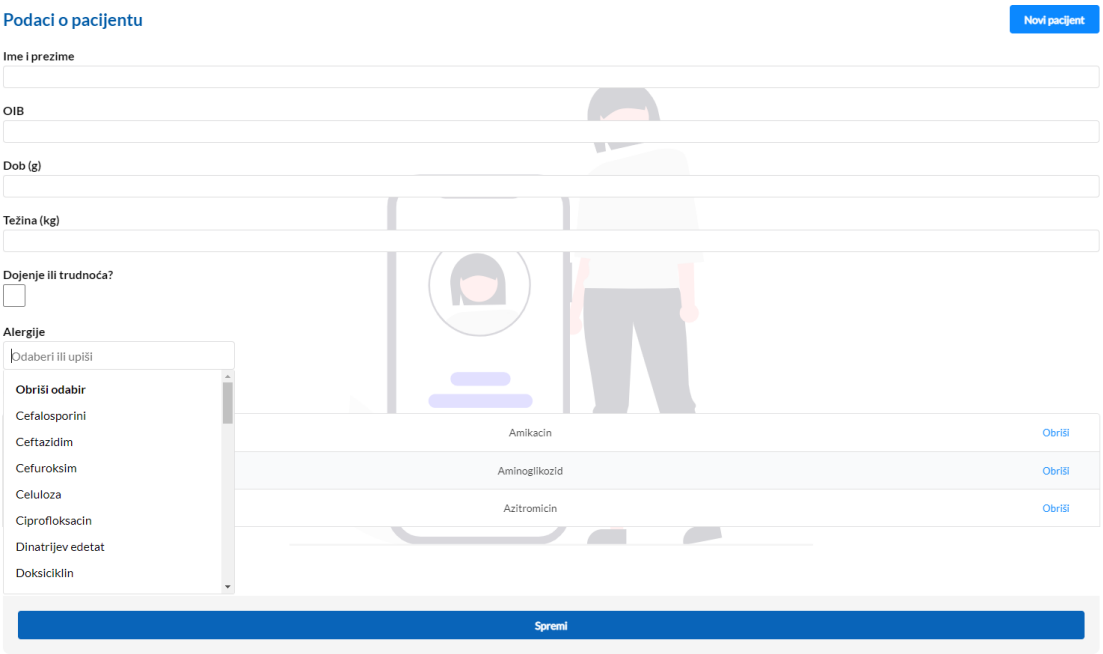
Komponenta prikazana na slici 4.11 s nazivom Dropdown je ručno rađena komponenta koja prima parametre liste, teksta unutar unosa, funkcije za odabir i još mnoge druge koje upravljaju ponašanjem komponente poput odabira više stavki odjednom, mogućnosti za filtriranje liste te druge. Odabir stavki s izbornika funkcionira tako da se prilikom odabira provjerava ima li

pritisnuta stavka identifikacijski broj ili ne. Provjera se vrši jer se u izborniku također nalazi „Obriši odabir“ stavka koja nema identifikacijski broj te se nakon spomenute provjere alergija sprema na zadnje mjesto u listi.

```
<Dropdown
  customclass="alergies-dropdown"
  inputNewData
  multiselect
  addButtonShouldBeShown={false}
  inputNewDataPlaceholder="Odaberi ili upiši"
  handleSelect={item => {
    if (item.id) {
      setPatientDetailsInfo({
        ...PatientDetailsInfo,
        alergies: [...PatientDetailsInfo.alergies, item],
      });
    }
  }}
  list={
    alergies
      ? [
        ...alergies.filter(alergy =>
          PatientDetailsInfo.alergies.length > 0
            ? !PatientDetailsInfo.alergies.find(
              userInfoAlergy =>
                userInfoAlergy.id === alergy.id
            )
            : alergy
        ),
      ]
      : []
  }
  headerTitle="Odaberi ili upiši"
  defaultHeaderOption="Odaberi ili upiši"
/>
```

Sl. 4.11. *Prikaz korištenja komponente padajućeg izbornika*

Korisnik može odabirati postojeće alergije s padajućeg izbornika, ali ne može unositi svoje jer je aplikacija napravljena tako da popis alergija i simptoma ovisi o svim simptomima i alergijama unesenim za sve lijekove.



Sl. 4.12. *Prikaz stranice s detaljima o korisniku*

Prema tome ako je u bazi unesen jedan lijek i on ima simptom vrućice te postavljenu alergiju na penicilin, onda će u padajućem izborniku za simptome i alergije biti prikazani ti podaci koji su uneseni za taj jedan lijek. Analogno tome ako se doda još simptoma i alergija, oni će također biti dodani automatski u bazu te će korisnik moći pristupiti tim simptomima i alergijama tako da pošalje zahtjev za dohvaćanje cijele liste alergija koja je prikazana na slici 4.13 pod nazivom *getAllAlergies*.

```
const PatientDetailsPage = props => {
  const dispatch = useDispatch();
  const PatientDetails = useSelector(state => state.patient);
  const alergies = useSelector(state => state.alergies.alergies);

  const [PatientDetailsInfo, setPatientDetailsInfo] = useState(
    PatientDetails && PatientDetails.currentPatientInfo
      ? { ...PatientDetails.currentPatientInfo, userId: props.userId }
      : {
          userId: props.userId,
          oib: null,
          name: null,
          age: null,
          alergies: [],
          pregnantOrBreastFeed: false,
          weight: null,
        }
  );

  useEffect(() => {
    dispatch(getAllAlergies());
  }, []);

  useEffect(() => {
    if (PatientDetails && PatientDetails.currentPatientInfo) {
      setPatientDetailsInfo(PatientDetails.currentPatientInfo);
    }
  }, [PatientDetails]);
```

Sl. 4.13. *Prikaz početnih vrijednosti i funkcija koje se izvršavaju dolaskom na stranicu s detaljima pacijenta*

Osim dohvaćanja svih alergija, dolaskom na ovu stranicu se postavljaju već postojeći detalji o pacijentu, ako postoji pacijent spremljen u lokalnoj bazi stanja, kako se ne bi izgubili uneseni podaci za trenutnog pacijenta kada se napusti ovaj dio stranice. Nakon uspješnog unošenja svih podataka za pacijenta, korisnik pritiskom na tipku „Spremi“ kreira unos za pacijenta u bazi i prikazuje mu se nova stavka u izborniku za prijedlog lijekova. Prema slici 4.14 se može vidjeti kako tipka „Spremi“ služi za izvršavanje dvije odvojene funkcije čije izvršavanje ovisi o postojanju pacijenta. U slučaju da pacijent postoji u bazi stanja, tada će se pritiskom na tipku izvršavati funkcija za ažuriranje pacijenta, dok se u suprotnom kreira novi pacijent.

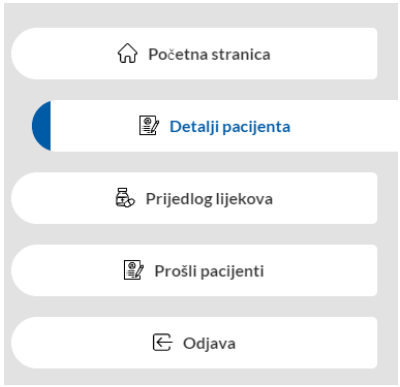
Prilikom izvršavanja poziva za kreiranje i ažuriranje pacijenta, kao što je prethodno rečeno, postavlja se *Authorization* zaglavlje unutar njih u kojem se nalazi token za pristup stranici i krajnjim točkama. Taj token omogućava valjani odziv s poslužiteljske strane tako da se provjerava njegova valjanost. U slučaju da nije valjan, ne postoji ili ako uloga korisnika ne odgovara ulozi postavljenoj za krajnju točku, tada se dobije odziv da korisnik ne može pristupiti toj krajnjoj točki.

Sl. 4.14. *Prikaz tipke za spremanje i ažuriranje pacijenta te funkcija za kreiranje i ažuriranje pacijenta*

4.2.2.2. Predlaganje lijekova

Nakon dodavanja novog pacijenta korisniku se prikazuje nova stavka na navigacijskoj traci pomoću koje može pristupiti dijelu stranice za prijedlog lijekova za pacijenta. Dodavanje nove stavke na navigacijsku traku se izvršava kao što je prikazano na slici 4.15 gdje se unutar „Navbar“ komponente dodaje nova komponenta „Navlink“ s rutom „/medication-suggestion“ i nazivom „Prijedlog lijekova“. Osim dodavanja nove stavke u navigacijsku traku, potrebno je dodati spomenutu novu rutu korisniku u listu ruta kojima on može pristupiti. Takvo dodavanje je potrebno napraviti jer postoje odvojene rute za administratora i korisnika i želi se ograničiti da korisnik ima pristup isključivo određenim rutama, a isto tako i administrator. Dodavanje nove rute u listu ruta

kojima korisnik može pristupiti se može vidjeti na slici 4.15 gdje se *userRoutes* vrijednost mijenja u ovisnosti o tome postoji li pacijent ili ne.

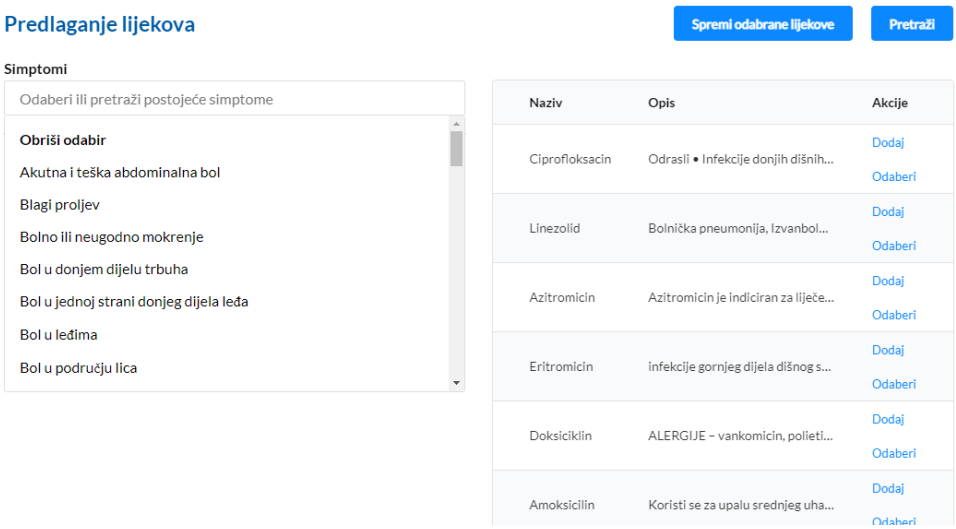



```
{patient && (
  <NavLink
    className="navbar-item"
    to="/medication-suggestion"
    activeClassName="active"
  >
    <img
      src={
        theme.darkTheme
          ? MedicationSuggestionImageWhite
          : MedicationSuggestionImage
      }
      alt="home"
      className="navbar-icon"
    />
    Prijedlog lijekova
  </NavLink>
)}
```

```
const Routes = () => {
  const user = useSelector(state => state.user);
  const theme = useSelector(state => state.theme);
  const patient = useSelector(state => state.patient.currentPatientInfo);

  const adminRoutes = [
    { route: '/users', component: <UsersListPage theme={theme} /> },
    { route: '/medications', component: <MedicationListPage theme={theme} /> },
    {
      route: '/patients-details-history',
      component: <PatientsListPage userRole={user.role} theme={theme} />,
    },
  ];
  const userRoutes = patient
    ? [
      {
        route: '/patient-details',
        component: <PatientDetailsPage userId={user.id} theme={theme} />,
      },
      {
        route: '/medication-suggestion',
        component: (
          <MedicationSuggestionPage userId={user.id} theme={theme} />
        ),
      },
      {
        route: '/patients-details-history',
        component: <PatientsListPage userRole={user.role} theme={theme} />,
      },
    ]
    : [
      {
        route: '/patient-details',
        component: <PatientDetailsPage userId={user.id} theme={theme} />,
      },
      {
        route: '/patients-details-history',
        component: <PatientsListPage userRole={user.role} theme={theme} />,
      },
    ];
  const isAdmin = user => {
    return user.role === 'admin';
  };
```

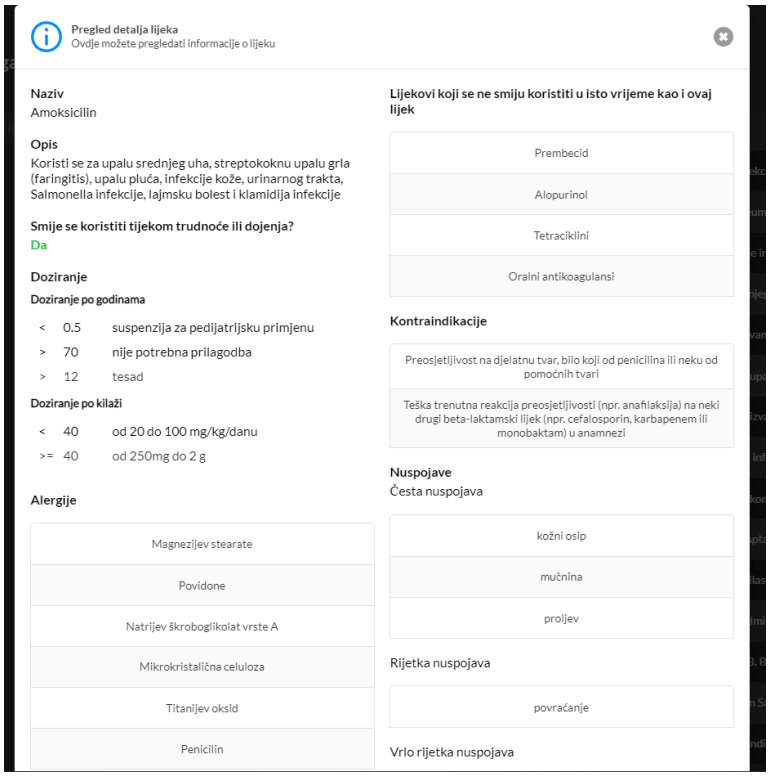
Sl. 4.15. *Navigacijski izbornik nakon dodavanja pacijenta, implementacija prikaza stavke te prikaz ruta za korisnika i administratora*

Ovom dijelu stranice, pomoću novo kreirane rute, može pristupiti isključivo korisnik i prikazana je na slici 4.16 s popisom simptoma i dohvaćenom listom lijekova. Na dijelu stranice za predlaganje lijekova korisnik može unositi simptome kao anamnezu pacijenta i prema unosu tih simptoma te postavljenim pojedinostima na stranici detalja pacijenta, pritiskom na tipku „Pretraži“, koja je vidljiva na slici 4.16, izvršava se poziv funkcije *getMedicationsBySymptomsAndAlergies* koja se vidi na slici 4.18. Toj funkciji se predaje lista odabranih simptoma, a njezinim završetkom se dobiva odziv s listom lijekova od poslužiteljske strane koji je prošao višekriterijsko filtriranje za predane parametre.



Sl. 4.16. *Prikaz stranice za prijedlog lijekova s dohvaćenom listom lijekova*

U slučaju da korisnik ne preda niti jedan simptom i zatraži listu lijekova, kao odziv će dobiti listu lijekova na kojoj se ne nalaze lijekovi koji u sebi imaju postavljene alergije kao i pacijent ili koji se ne slažu s odabirom o dojenju i trudnoći. Nakon prikaza liste lijekova, korisnik može pregledati detalje o svakom lijeku pritiskom na tipku „Odaberi“ koja se nalazi pokraj svakog lijeka. Pritiskom na tipku, odabrani lijek se dodaje u stanje koje se predaje detaljnom prikazu te se otvara prikaz o lijeku u kojem korisnik može vidjeti informacije poput opisa, cijele liste alergija i simptoma, kontraindikacije, nuspojave lijeka te druge informacije.



Sl. 4.17. *Prikaz detalja o lijeku*

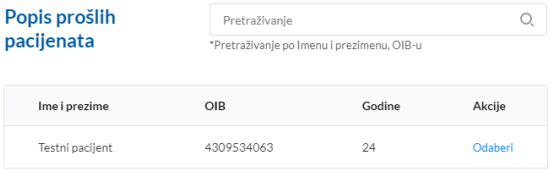
Druga tipka s nazivom „Dodaj“ služi za dodavanje lijeka na listu odabranih lijekova za kreiranog pacijenta. Dodani lijek se može naknadno obrisati s liste odabranih lijekova za pacijenta te se lista odabranih lijekova i simptoma može spremiti u unos za pacijenta pritiskom na tipku „Spremi odabrane lijekove“. Tipka za spremanje odabranih lijekova izvršava poziv iste funkcije za ažuriranje detalja o pacijentu kao kod tipke „Spremi“ na stranici s detaljima o pacijentu, kao što je vidljivo na slici 4.18.

```
<button
  type="button"
  className="save-patient-info"
  onClick={() => {
    dispatch(
      updatePatientDetails(PatientDetailsInfo, selectedSymptoms)
    );
  }}
>
  Spremi odabrane lijekove
</button>
<button
  type="button"
  className="search-medications-button"
  onClick={() => {
    dispatch(getMedicationsBySymptomsAndAlergies(selectedSymptoms));
  }}
>
  Pretraži
</button>
```

Sl. 4.18. *Tipke za spremanje odabranih lijekova i pretraživanja lijekova po pojedinostima pacijenta*

4.2.2.3. Lista prošlih pacijenata

Naposljetku, kao zadnji dio korisničkog prikaza i zadnja ruta kojoj korisnik može pristupiti je lista prošlih pacijenata koje je kreirao korisnik. Na njoj se, osim prošlih pacijenata koje je korisnik kreirao, može vidjeti trenutni pacijent kojeg korisnik pregledava. Ova lista je namijenjena praćenju stanja prošlih pregleda pacijenata gdje se osim pregleda cijele liste, pacijenti mogu pretraživati pomoću unosa koji je vidljiv na slici 4.19.

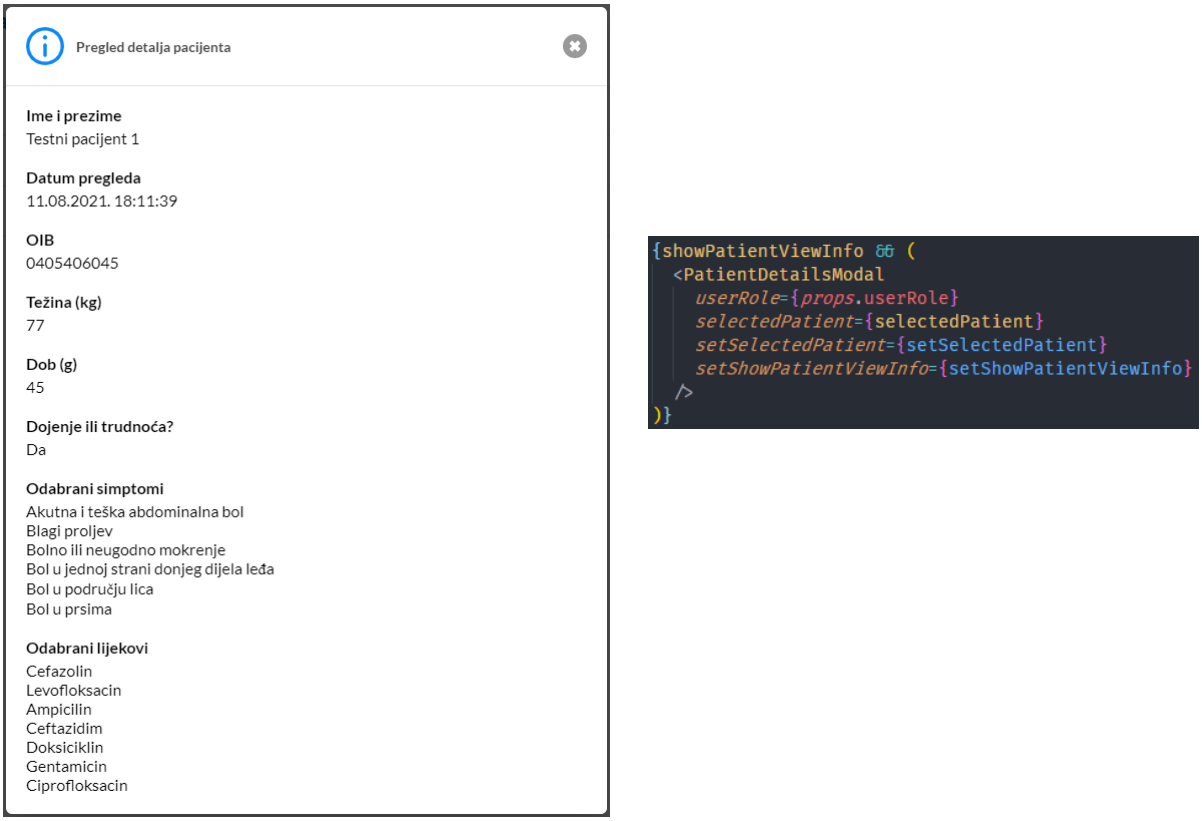



```
export const searchPatientsByText = query => {
  return (dispatch, getState) => {
    axios({
      method: 'GET',
      url: `${process.env.REACT_APP_API_URL}
      /patients-details/search?search=${query}`,
      headers: {
        Authorization: `Bearer ${Cookies.get('Accesstoken')}`,
      },
    })
      .then(response => {
        dispatch(setPatientsList(response.data));
      })
      .catch(error =>
        dispatch(
          setErrorCurrentPatientInfo({
            message: error.message,
            error: error.message,
            status: error.status,
          })
        )
      );
  };
};
```

Sl. 4.19. *Prikaz liste unosa o pacijentima i funkcije za pretraživanje pacijenata po unosu*

Pretraživanje se vrši po imenu i prezimenu te OIB-u tako da se funkciji vidljivoj na slici 4.19 predaje tekst prema kojem će se pretraživati pacijent koja će s obzirom na unos vratiti novu listu pacijenata. Unos prema kojem se vrši pretraživanje, ne samo kod pacijenata, već i kod drugih popisa koji sadrže pretraživanje, se predaje kao upit unutar URL parametra funkcije koja poziva krajnju točku za pretraživanje.

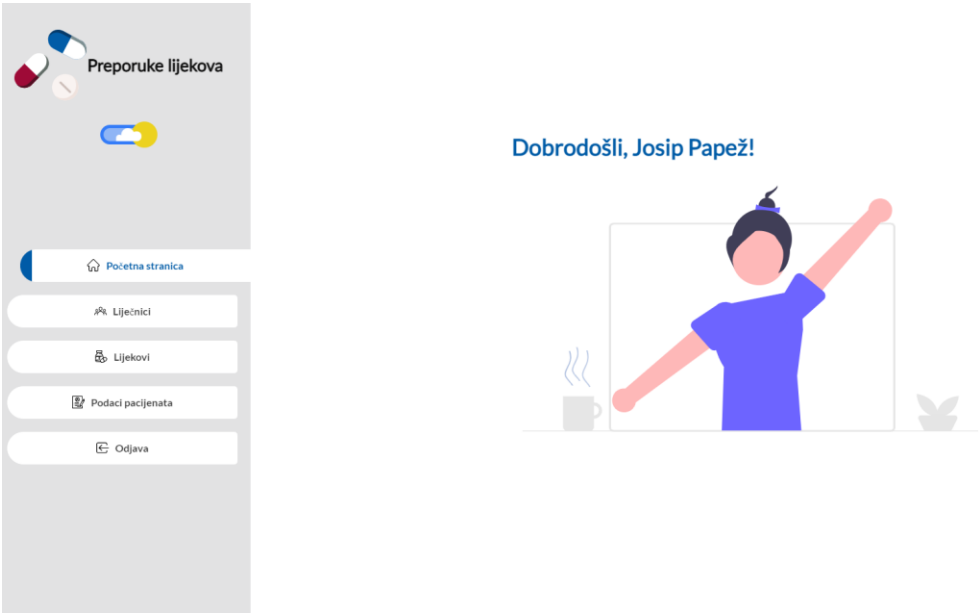
Pritiskom na tipku „Odaberi“, pokraj unosa o pacijentu, otvara se detaljan prikaz unosa tako da se spremi objekt unosa o pacijentu u lokalno stanje te se preda detaljnom prikazu koje prikaže unesene podatke. Na detaljnom prikazu vidljiv je datum pregleda, podaci o pacijentu, lijekovi, kao i simptomi i alergije koje su bile odabrane za pacijenta. Korisnik može pregledavati podatke dok administrator ima mogućnost brisanja unosa o pacijentu uz pregled podataka.



Sl. 4.20. *Detaljan prikaz informacija o unosu za pacijenta i implementacija detaljnog prikaza*

4.2.3. Administratorski prikaz aplikacije

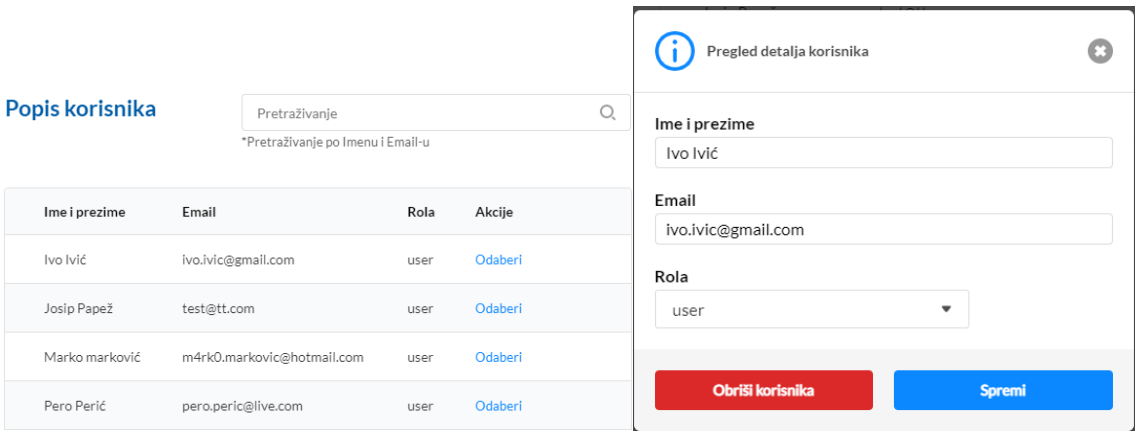
Administrator ulaskom, tj. prijavom, na stranicu ima pristup listi liječnika, tj. korisnika, koji su registrirani na stranici, popisu lijekova te listi svih unosa pacijenata. Tim dijelovima stranice pristupa pomoću navigacijske trake prikazane na slici 4.21.



Sl. 4.21. *Prikaz početne stranice administratora nakon prijave*

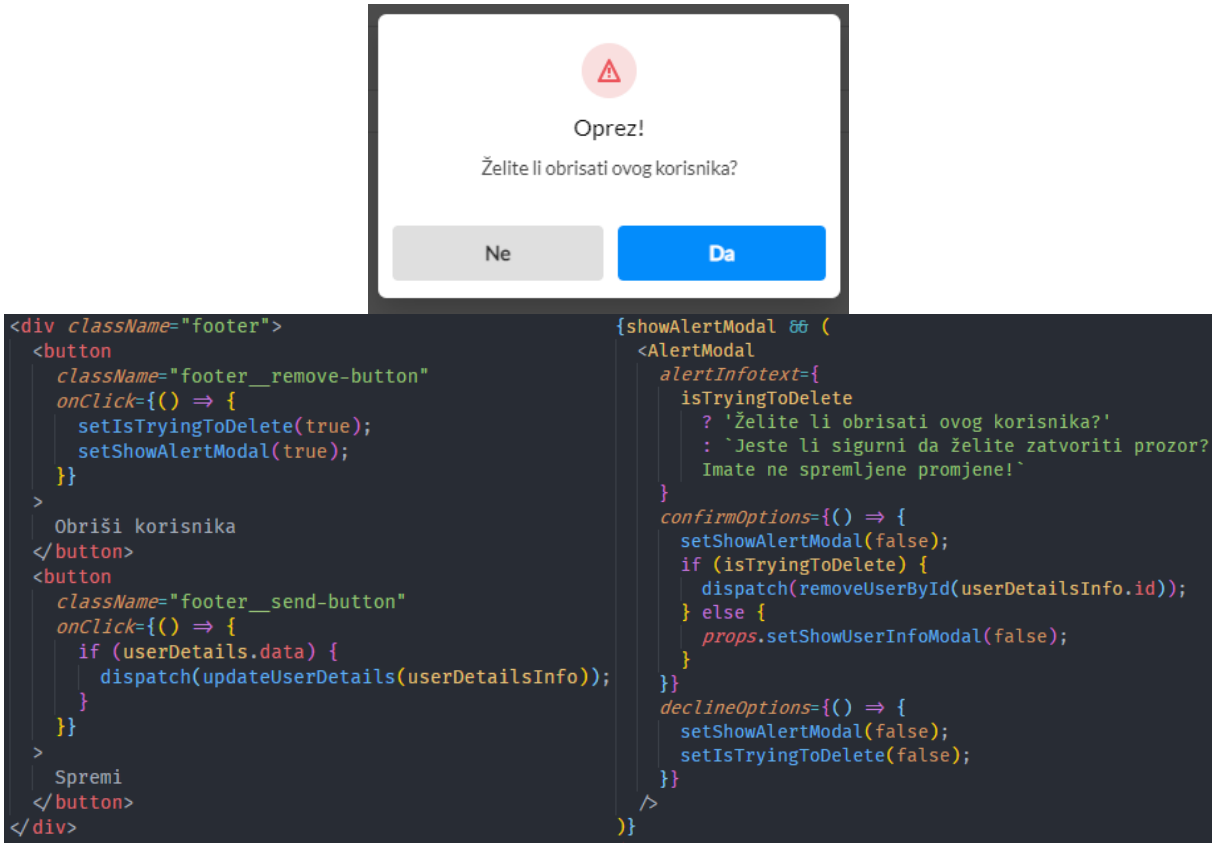
4.2.3.1. Popis korisnika

Prelaskom na stranicu s korisnicima, tj. liječnicima, prikazuje se popis sa svim registriranim korisnicima koji nisu administratori. Pritiskom na tipku „Odaberi“, koja je prikazana na slici 4.22 s desne stane, administrator može svakom korisniku pregledati detalje koje je unio i može ih izmijeniti pa tako korisniku može izmijeniti ulogu, promijeniti e-poštu te ime i prezime. Nakon pritiska na spomenutu tipku otvara se detaljniji prikaz korisnika gdje se administratoru prikazuje forma s korisničkim podacima. Uz pregled i izmjene korisnika, administrator može obrisati korisnika sa stranice pri čemu se korisnik briše iz baze podataka te se mora ponovno registrirati kako bi pristupio stranici. Brisanje korisnika se vrši tako da se poslužiteljskoj strani pošalje identifikacijski broj korisnika, kojeg svaki korisnik ima, i prema tom broju poslužiteljska strana obavlja brisanje svih podataka za tog korisnika.



Sl. 4.22. *Prikaz liste korisnika i detalja korisnika na administratorskom prikazu*

Uz brisanje korisnika je dodan prikaz potvrde, koji je vidljiv na slici 4.23 u slučaju da administrator greškom pritisne na tipku „Obriši korisnika“. Unutar implementacije upozorenja *AlertModal* se mogu vidjeti funkcije koje će se izvršiti pritiskom na „Da“ ili „Ne“ opciju. Funkcija za „Da“ opciju je vidljiva unutar *confirmOptions* parametra, a funkcija za „Ne“ opciju unutar *declineOptions* parametra.




```
<div className="footer">
  <button
    className="footer__remove-button"
    onClick={() => {
      setIsTryingToDelete(true);
      setShowAlertModal(true);
    }}
  >
    Obriši korisnika
  </button>
  <button
    className="footer__send-button"
    onClick={() => {
      if (userDetails.data) {
        dispatch(updateUserDetails(userDetailsInfo));
      }
    }}
  >
    Spremi
  </button>
</div>
```

```
{showAlertModal && (
  <AlertModal
    alertInfotext={
      isTryingToDelete
        ? 'Želite li obrisati ovog korisnika?'
        : `Jeste li sigurni da želite zatvoriti prozor?
        Imate ne spremljene promjene!`
    }
    confirmOptions={() => {
      setShowAlertModal(false);
      if (isTryingToDelete) {
        dispatch(removeUserById(userDetailsInfo.id));
      } else {
        props.setShowUserInfoModal(false);
      }
    }}
    declineOptions={() => {
      setShowAlertModal(false);
      setIsTryingToDelete(false);
    }}
  />
)}
```

Sl. 4.23. *Prikaz upozorenja pri brisanju korisnika i implementaciji prikaza*

Osim pregleda detalja korisnika, administrator ima mogućnost pretraživanja korisnika. Korisnici se pretražuju prema e-pošti te imenu i prezimenu s obzirom na vrijednost pretraživanja. Vrijednost pretraživanja se, uz spremanje unutar komponente za pretraživanje, također šalje i roditeljskoj komponenti u kojoj se komponenta za pretraživanje koristi što je vidljivo na slici 4.24 sa *setSearchQuery* funkcijom. Ona se poziva unutar „Search“ komponente pritiskom na tipku Enter na tipkovnici i time se postavlja vrijednost unosa u roditeljskoj komponenti u *searchQuery* stanje.

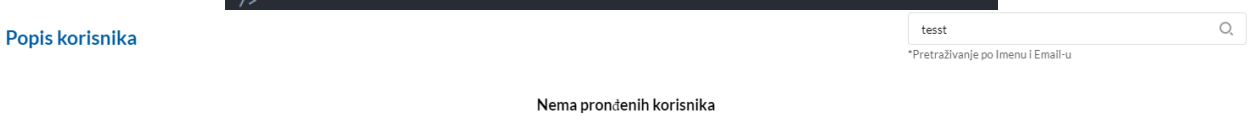
```jsx
<DataDisplay
  dataHeader="Popis korisnika"
  headerBolded
  centerHeaderVertically
  headerFontSize={23}
  headerTextColor={props.theme.darkTheme ? '#fff' : '#005BA7'}
  dataFullWidth
  TopSpacing={30}
  floatDataRight
  data={
    <Search
      searchingByInfo="*Pretraživanje po Imenu i Email-u"
      fetchData={() => {
        return getAllUsers();
      }}
      fetchDataByName={text => {
        return searchUsersByText(text);
      }}
      setSearchQuery={setSearchQuery}
    />
  }
/>
```

```jsx
const Search = props => {
  const [searchTerm, setSearchTerm] = useState('');
  const dispatch = useDispatch();

  return (
    <div className="search">
      <form
        className="search__form"
        onSubmit={e => {
          e.preventDefault();
          if (props.setSearchQuery) {
            if (searchTerm === '') {
              props.setSearchQuery(null);
              dispatch(props.fetchData());
            } else {
              props.setSearchQuery(searchTerm);
              dispatch(props.fetchDataByName(searchTerm));
            }
          }
        }}
      >
        <input
          type="text"
          disabled={props.disabled}
          placeholder="Pretraživanje"
          className="search-input"
          value={searchTerm}
          onChange={e => setSearchTerm(e.target.value)}
        />
      </form>
      <small className="search-by-text">{props.searchingByInfo}</small>
    </div>
  );
};
```

Sl. 4.24. *Prikaz implementacije pretraživanja i komponente za pretraživanje*

Spremanjem unosa pretraživanja može se dodatno utjecati na izgled komponente u kojoj se koristi pretraživanje ili se mogu izvršavati razni pozivi funkcija s obzirom na vrijednost pretraživanja. U ovom slučaju u web sustavu se koristi vrijednost pretraživanja kako bi se prikazala različita informacija u slučaju da nema pronađenih korisnika za vrijednost po kojoj se pretražuje.

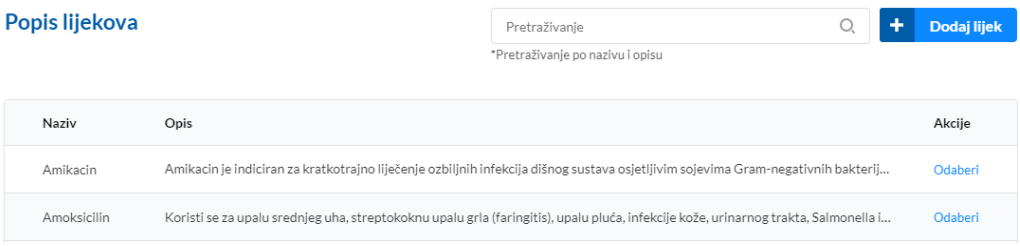
```jsx
<DataDisplay
  TopSpacing={40}
  dataFullWidth
  data={
    <div className="users-list-page__list__display-list">
      {usersList && usersList.users && usersList.users.length > 0 ? (
        <UsersList
          usersList={usersList.users}
          setUserId={setUserId}
          setShowUserInfoModal={setShowUserInfoModal}
        />
      ) : (
        <div className="users-list-page__list__display-list__not-found">
          {searchQuery
            ? 'Nema pronđenih korisnika'
            : 'Nema unesenih korisnika'}
        </div>
      )}
    </div>
  }
/>
```



Sl. 4.25. *Prikaz različitih poruka u odnosu na vrijednost pretraživanja*

4.2.3.2. Popis lijekova

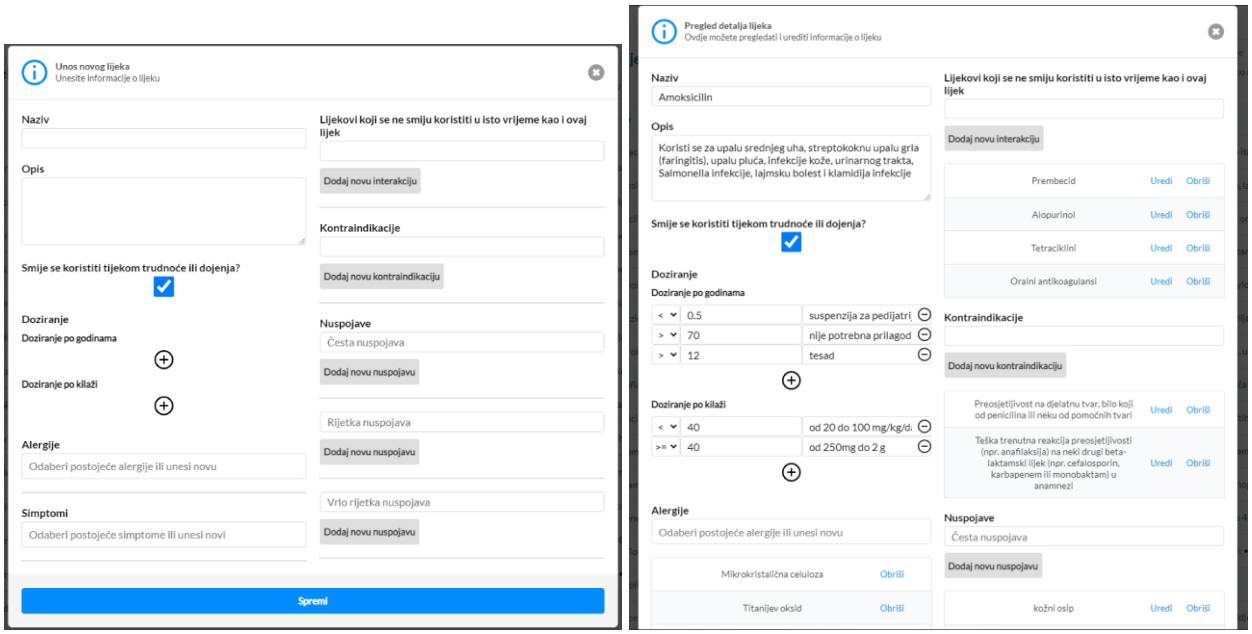
Osim liste korisnika administrator ima pristup listi lijekova. Prelaskom na stranicu s listom lijekova koja se nalazi na navigacijskoj traci pod nazivom „Lijekovi“, administratoru se prikazuje popis svih unesenih lijekova u bazi podataka. Na ovom dijelu stranice mogu se pretraživati lijekovi po nazivu i opisu, dodavati novi lijekovi na popis, uređivati postojeći lijekovi te brisati lijekovi s popisa. Implementacija liste lijekova je prikazana na slici 4.26 gdje je vidljiv prikaz podataka te funkcije koja otvara detaljan prikaz lijeka.




```
const MedicationList = props => {
  const dispatch = useDispatch();
  const [headers, setHeaders] = useState([
    {
      header: 'Naziv',
      headerKey: 'name',
    },
    {
      header: 'Opis',
      headerKey: 'description',
    },
    {
      header: 'Akcije',
    },
  ]);
  return (
    <table style={{ width: '100%' }} className="list-table">
      <ListHeader setHeaders={setHeaders} headers={headers} />
      <tbody className="list-table__item-row">
        {props.medicationList.map(medication => {
          return (
            <tr className="spacer  item-row" key={medication.id}>
              <td style={{ width: '10%', minWidth: '50px' }}>
                {medication.name}
              </td>
              <td
                style={{
                  maxWidth: 0,
                  overflow: 'hidden',
                  textOverflow: 'ellipsis',
                  whiteSpace: 'nowrap',
                }}
              >
                {medication.description}
              </td>
              <td style={{ width: '10%', minWidth: '50px' }}>
                <button
                  id="link-to-medication-page"
                  onClick={() => {
                    dispatch(medicationInfoFetched({ data: null }));
                    props.setSelectedMedication(medication);
                    props.setShowMedicationInfoModal(true);
                  }}
                >
                  Odaberi
                </button>
              </td>
            </tr>
          );
        })}
      </tbody>
    </table>
  );
};
```

Sl. 4.26. *Prikaz dijela liste lijekova i implementacije liste lijekova*

Pritiskom na tipku „Odaberi“ pokraj lijeka kojeg se želi pregledati, otvara se detaljan prikaz lijeka u kojem se mogu pregledavati detalji o lijeku te uređivati ti isti podaci, dok se pritiskom na tipku „Dodaj lijek“ otvara forma za dodavanje novog lijeka. Podaci unutar forme se popunjavaju tako da se formi preda lijek na kojeg se pritisne i prema predanim podacima se postavi početno stanje forme čime se popune polja u kojima se takvi početni podaci trebaju nalaziti.



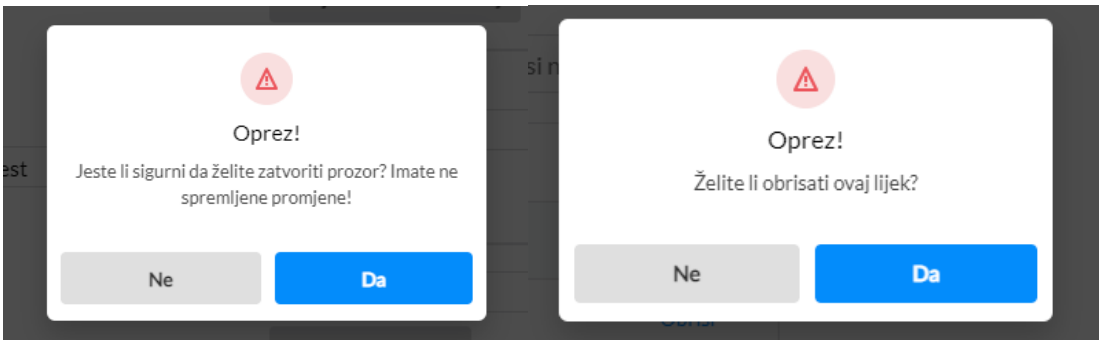
Sl. 4.27. *Prikaz forme za dodavanje novog lijeka i uređivanje unesenog lijeka*

U formama za dodavanje novog lijeka i za uređivanje postojećeg lijeka administrator postavlja alergije i simptome za određeni lijek koji će se prikazati korisniku u listama za odabir simptoma i alergija. Prilikom upisivanja nove alergije i simptoma vrši se provjera postoji li unos u postojećoj listi alergija ili simptoma i prema tome se prikazuje tipka za dodavanje novog unosa. Provjera postojanja alergije i simptoma po nazivu prikazana je na slici 4.28 gdje se tipka „Dodaj“ prikazuje nakon 0.5s u slučaju da unos ne postoji u listi, dok se u slučaju postojanja vrijednost *showAddButton* postavlja na „false“ stanje, tj. stanje u kojem se ne prikazuje tipka.

```
useEffect(() => {
  let timeout;
  if (addButtonShouldBeShown) {
    if (filterInput && filterInput !== '') {
      timeout = setTimeout(() => {
        if (
          !fullList.find(
            item => item.name.toLowerCase() === filterInput.toLowerCase()
          )
        ) {
          setShowAddbutton(true);
        }
      }, 500);
    } else {
      setShowAddbutton(false);
    }
  }
  return () => {
    clearTimeout(timeout);
  };
}, [filterInput]);
```


Sl. 4.28. *Prikaz provjere postojećih unosa i prikaz tipke za dodavanje novog unosa*

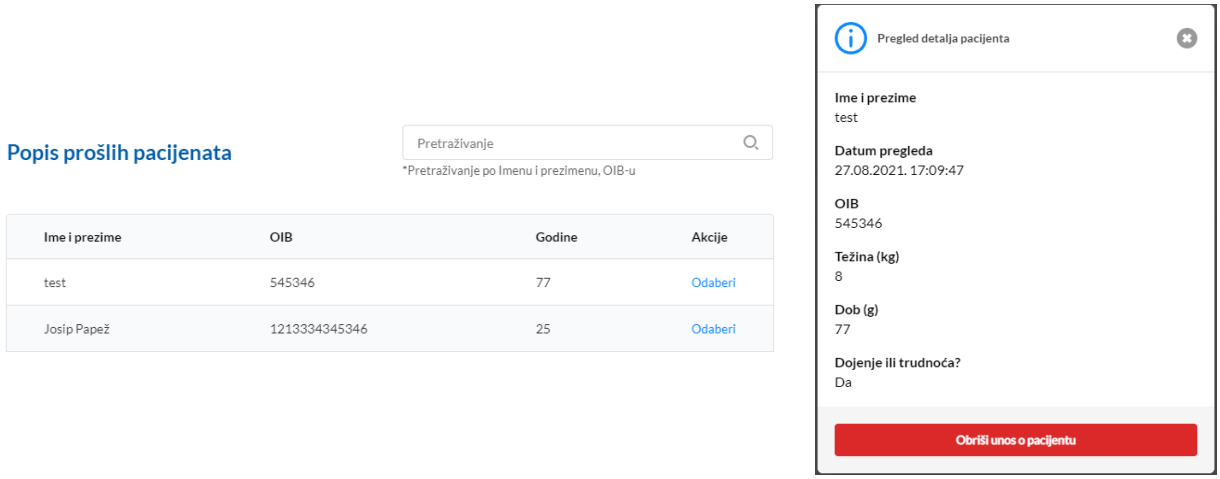
Prilikom izmjena podataka dodana je i provjera u slučaju da administrator želi izaći iz detaljnog pregleda prije nego što je spremio podatke. Osim provjere izmjene podataka, postoji i provjera prilikom brisanja lijeka u slučaju da se greškom pritisne tipka za brisanje lijeka. Implementacija upozorenja je jednaka kao i kod brisanja korisnika, jedina razlika je što se kod slučaja s izmjenama, provjerava početno stanje forme i početno stanje lijeka. U slučaju da stanja nisu jednaka, stanje za prikaz upozorenja se pri pokušaju izlaska iz forme postavlja na istinito i time se forma prikazuje.



Sl. 4.29. *Prikaz upozorenja pri izlasku iz detaljnog pregleda i prilikom brisanje lijeka*

4.2.3.3. Popis unesenih podataka o pacijentima

Naposljetku, zadnji dio administratorskog prikaza uključuje popis unesenih podataka o pacijentima koji podrazumijeva sve unose o pacijentima od svih liječnika koji su registrirani na stranici. Kao i neke prethodno opisane liste, ova lista u sebi sadrži unos za pretraživanje pacijenata prema ime i prezimenu te OIB-u. Osim toga, svaki pojedini pacijent se može pregledati pritiskom na tipku „Odaberi“ u tablici gdje se odabrani pacijent sprema u lokalno stanje liste te se predaje kao objekt komponenti za detaljan prikaz. Komponenta za detaljan prikaz se otvara i prikazuje podatke iz objekta o pacijentu nakon što zaprimi valjani objekt koji sadrži unos o pacijentu i nakon što se postavi zastavica na istinito stanje koja označava prikazivanje komponente.



Sl. 4.30. *Prikaz liste pacijenata i pregled detalja o pacijentu*

Uz prikazivanje detalja o pacijentu, na komponenti za detaljan prikaz se nalazi tipka za brisanje unosa o pacijentu koju može vidjeti isključivo administrator. Pritiskom na tipku za brisanje otvara se prikaz upozorenja, poput onoga na slici 4.29, čijom potvrdom se šalje identifikacijski broj unosa o pacijentu, koji se nalazi unutar objekta predanog prilikom otvaranja detaljnog prikaza, funkciji za brisanje unosa vidljivoj na slici 4.31.

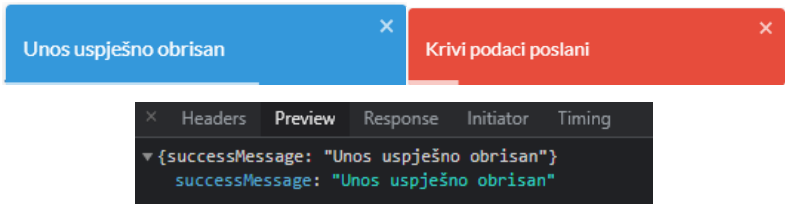
```
export const removePatientDetailsById = id => {
  return (dispatch, getState) => {
    axios({
      method: 'DELETE',
      url: `${process.env.REACT_APP_API_URL}/patients-details/delete`,
      headers: {
        Authorization: `Bearer ${Cookies.get('Accesstoken')}`,
      },
      data: { id },
    })
      .then(response => {
        dispatch(
          setErrorCurrentPatientInfo({
            message: response.data.successMessage,
            error: null,
            status: response.status,
          })
        );
        dispatch(
          setErrorCurrentPatientInfo({
            message: null,
            error: null,
            status: null,
          })
        );
        dispatch(getAllPatientsForAdmin());
      })
      .catch(error =>
        dispatch(
          setErrorCurrentPatientInfo({
            error: error.response.data,
            status: error.status,
          })
        )
      );
  };
};
```

Sl. 4.31. *Prikaz funkcije za brisanje unosa o pacijentu*

4.2.4. Prikaz obavijesti

Prilikom svih poziva prema poslužiteljskoj strani gdje se spremaju, uređuju, kreiraju ili šalju podaci, tj. unosi, korisniku se prikazuje obavijest o izvršenom zahtjevu, bio on uspješan ili ne. Obavijesti su izvedene pomoću „React-Toastify“ paketa, koji služi za prikazivanje kartice s tekstom na određeno vrijeme prikazane na slici 4.32, te praćenje statusa odziva na razne zahtjeve. Praćenje se izvršava tako da se pri svakoj promjeni određenih dijelova baze stanja provjerava status unutar tog dijela baze znanja, kao što je vidljivo na slici 4.33 gdje se provjerava *user, userInfo, medicationInfo* i *currentPatientInfo* te se pri određenim statusima prikazuju različite poruke i obavijesti. Spomenuti paket, koji se koristi za prikaz obavijesti, ima u sebi ugrađeno nekoliko različitih načina za prikaz obavijesti te različite izglede za različite obavijesti. Prema tome, na slici 4.33 se može vidjeti da će se obavijest za uspješno obavljene zahtjeve prikazati

sredini gornjeg dijela ekrana, automatski će se zatvoriti nakon 3000 milisekundi i može se zatvoriti pritiskom na nju.



Sl. 4.32. *Obavijesti o uspješnom i neuspješnom izvršenju te prikaz uspješnog odziva*

Statusi se postavljaju ručno prilikom odziva na zahtjeve kao što je vidljivo u *then* dijelu funkcije prikazane na ranije prikazanoj slici 4.31, gdje se pozivom *setErrorCurrentPatientInfo* predaju poruka, greška i status zahtjeva. Mijenjanjem statusa unutar tog dijela baze stanja se okida provjera unutar komponente za prikaz obavijesti i time se prikazuje obavijest. Također, kako obavijest ne bi bila stalno prikazana, status unutar tog dijela baze stanja mora biti vraćen na početnu vrijednost i time se prekida okidanje prikazivanja obavijesti.

```
const Notification = () => {
  const user = useSelector(state => state.user);
  const userInfo = useSelector(state => state.user.userInfo);
  const currentPatientInfo = useSelector(state => state.patient);
  const medicationInfo = useSelector(
    state => state.medicationList.medicationInfo
  );

  const statusArray = [user, userInfo, medicationInfo, currentPatientInfo];
  useEffect(() => {
    statusArray.map(statusInfo => {
      if (
        statusInfo &&
        (statusInfo.status === 200 || statusInfo.status === 201)
      ) {
        toast.info(statusInfo.message, {
          position: 'top-center',
          autoClose: 3000,
          hideProgressBar: false,
          closeOnClick: true,
          pauseOnHover: true,
          draggable: true,
          progress: undefined,
        });
      }
      if (
        statusInfo &&
        statusInfo.status &&
        statusInfo.status !== 200 &&
        statusInfo.status !== 201
      ) {
        toast.error(
          statusInfo.message ? statusInfo.message : 'Nešto je pošlo po krivu!',
          {
            position: 'bottom-center',
            autoClose: 3000,
            hideProgressBar: false,
            closeOnClick: true,
            pauseOnHover: true,
            draggable: true,
            progress: undefined,
          }
        );
      }
    });
  }, [statusArray]);
```

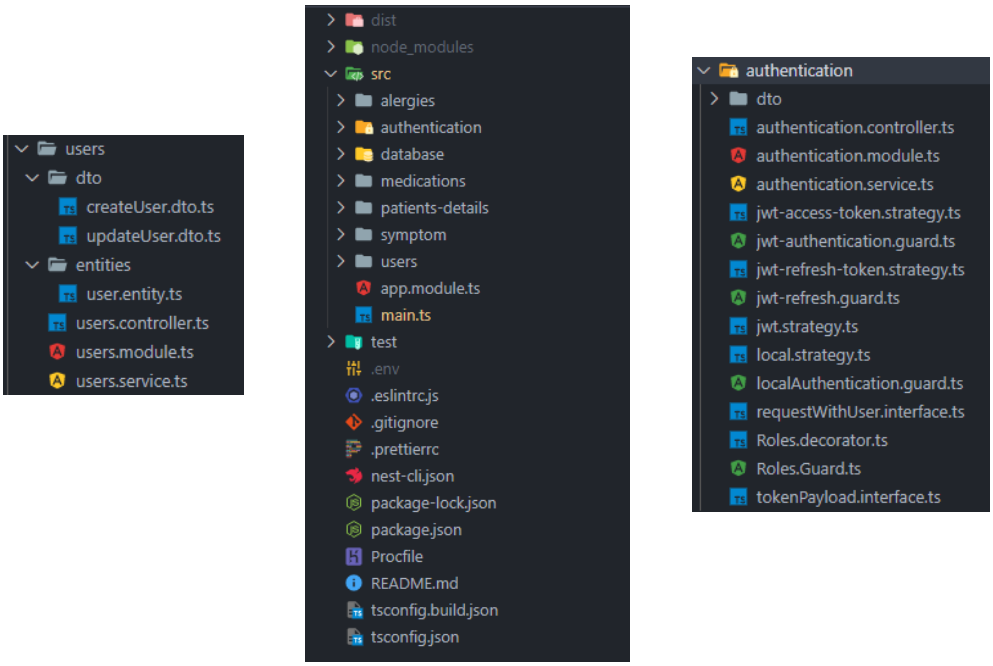
Sl. 4.33. *Prikaz komponente za kreiranje obavijesti*

4.3. Programsko rješenje na strani poslužitelja

Filtriranje lijekova, logika registracije i prijave korisnika kao i unos novih lijekova, izmjena, dohvaćanje i brisanje lijekova te druge potrebne funkcionalnosti su odrađene na poslužiteljskoj strani. Na njoj su posebno definirane krajnje točke kojima će moći pristupati isključivo administrator, korisnik ili oboje te će prema tome korisnici i administratori moći pristupiti različitim dijelovima sustava. Poslužiteljska strana, kao što je ranije navedeno, je napravljena u NestJS okviru te pomoću PostgreSQL sustava za upravljanje bazama podataka.

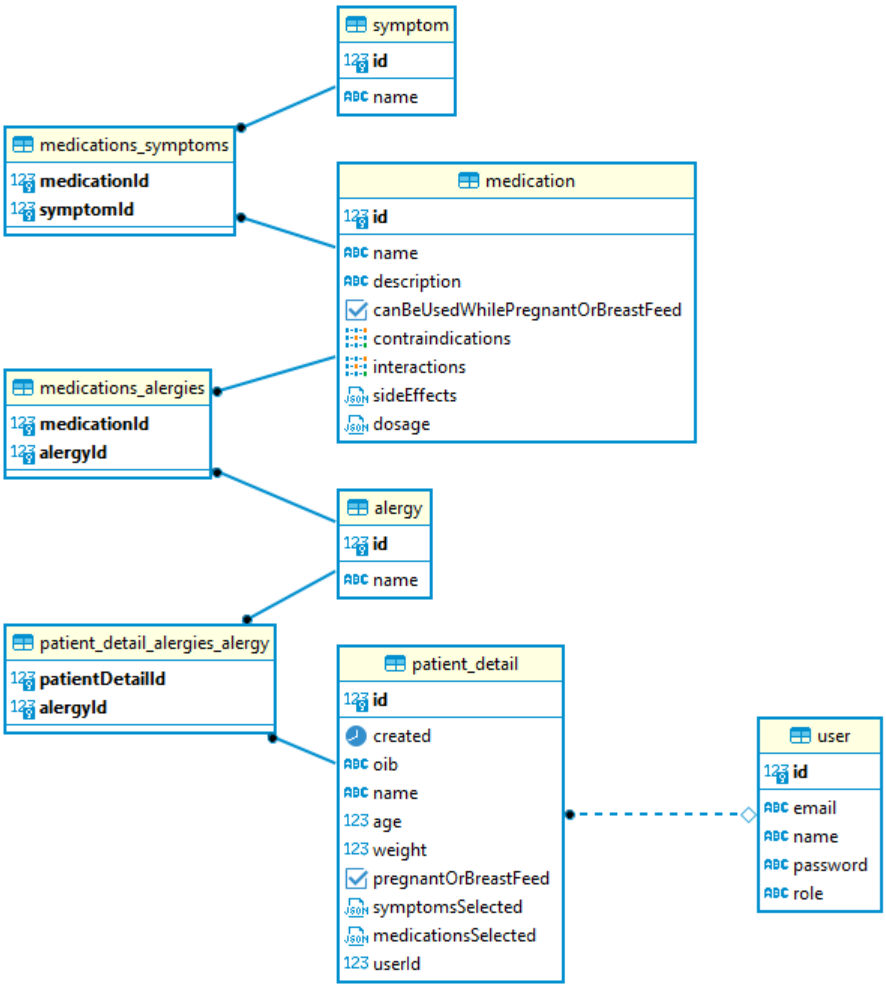
Prema slici 4.34 se može vidjeti struktura poslužiteljske strane, njezini modula i usluga unutar njih, kao i entiteta koji se prikazuju u bazi u obliku tablice. Glavna datoteka koja služi za pokretanje poslužiteljske strane je „main.ts“, a „app.module.ts“ povezuje sve module i pokreće ih prilikom pokretanja poslužiteljske strane.

Svaka komponenta u sebi ima datoteke kontrolora, pružatelja usluge, tj. servisa, entiteta i modula. Kontrolori su odgovorni za obradu dolaznih zahtjeva i vraćanje odgovora klijentu, pružatelji usluge su obične JavaScript klase koje su deklarirane kao davatelji usluga u modulu i koje odrađuju složenije zadatke, entiteti predstavljaju tablice prikazane u bazi podataka, a moduli povezuju kontrolore, usluge i entitete određene komponente te služe za učinkovito organiziranje komponenti. Prema tome, na spomenutoj slici se može vidjeti da unutar „users“ komponente postoje sve navedene datoteke koje su potrebne za dodavanje „users“ tablice, funkcionalnosti i krajnjih točaka pomoću kojih će se moći upravljati podacima u tablici.



Sl. 4.34. *Prikaz strukture projekta, komponenti i entiteta na poslužiteljskoj strani*

Osim navedenih datoteka, unutar komponenti se mogu nalaziti i dodatne datoteke s nastavkom „guard“ koje imaju odgovornost određivanja hoće li obrađivač rute obraditi zahtjev ili ne. Prema tome, „guard“ datoteke se koriste prilikom implementacije svake rute kako bi bila zaštićena određenim sigurnosnim postupcima poput provjere valjanosti tokena, uloga korisnika ili podataka koji su poslani prije nego što se izvedu pozivi prema bazi i manipulacija podacima.



Sl. 4.35. *Relacijski dijagram entiteta*

4.3.1. Ovjera korisnika

Ovjera korisnika za prijavu ili registraciju se obavlja na *authentication* ruti slanjem podataka na *register* ili *log-in* podrutu kao što je prikazano na slici 4.36 gdje se unutar *@Controller* dekoratera označava naziv rute. Unutar klase za ovjeru navode se metode i podrute koje će moći pozivati korisnik te na koje će moći slati podatke. Također, unutar konstruktora navode se varijable koje će se moći koristiti unutar klase kontrolora. U ovom slučaju koristit će se *authenticaionService* varijabla koja je tipa usluge za ovjeru korisnika, tj. *AuthenticaionService*.

Svaka funkcija može biti sinkrona ili asinkrona što je prikazano sa *async* oznakom pokraj naziva funkcije. Također, funkcija koja se želi pozivati na određenoj podruti mora imati dekorater koji označava metodu za poziv (Post, Get, Patch, Remove) i željeni naziv podrute. Primjer registracije bi prema spomenutom bilo slanje podataka Post metodom na rutu „*./authentication/register*“.

U funkcijama koje će se izvršavati pri korisničkim pozivima mogu se definirati i tipovi podataka kakve mora primiti funkcija unutar tijela zahtjeva kako bi se uspješno izvršila. Primjer tome je slika 4.36 gdje je definiran izgled objekta kakav mora biti poslan na krajnju točku za registraciju kako bi se uspješno izvršila, a primjena tog objekta je vidljiva na istoj slici. U slučaju da dođe drugačiji objekt ili s drugačijim vrijednostima od definiranih, greška će biti vraćena korisniku.

```
@Controller('authentication')
export class AuthenticationController {
  constructor(private readonly authenticationService: AuthenticationService) {}

  @Post('register')
  async register(@Body() registrationData: RegisterDto) {
    return this.authenticationService.register(registrationData);
  }

  @HttpCode(200)
  @UseGuards(LocalAuthenticationGuard)
  @Post('log-in')
  async logIn(@Req() request: RequestWithUser, @Res() response: Response) {
    const { user } = request;
    const accessToken = this.authenticationService.getJwtAccessToken(user.id);
    const refreshToken = this.authenticationService.getJwtRefreshToken(user.id);
    user.password = undefined;
    return response.send({ user, accessToken, refreshToken });
  }
```

```
export class RegisterDto {
  email: string;
  name: string;
  password: string;
}

export default RegisterDto;
```

Sl. 4.36. *Kontrolor ovjere i krajnje točke za registraciju i prijavu te objekt za registraciju korisnika*

Ovakvi primjeri krajnjih točaka ostaju kroz cijeli projekt gotovo identični, jedina razlika među određenim krajnjim točkama su različite zaštite koje će se koristiti za krajnje točke. U ovom slučaju kod prijave koristi se *LocalAuthenticaionGuard* zaštita koja koristi prilagođenu provjeru prema e-pošti i lozinki te poziva *getAuthenticatedUser* iz *authenticationService* usluge, kao što je vidljivo na slici 4.37. Navedena funkcija prima e-poštu i lozinku te vrši provjeru lozinke za pronađenog korisnika po e-pošti. U slučaju da se lozinke podudaraju, korisnik se vraća u odzivu te se predaje natrag *log-in* podruti u kojoj se može koristiti pozivanjem *request.user* objekta. Funkcija za prijavu zatim kreira token za pristup stranici te token za obnovu tokena za pristup i zajedno sa primljenim korisnikom šalje podatke u odzivu. Status pri uspješnom izvršenju je 200 dok je status pri neuspješnom izvršenju 400.

Sl. 4.37. *Prikaz lokalne strategije za zaštitu i funkcije za dohvaćanje korisnika te provjeru lozinke*

Tokeni kod prijave korisnika se generiraju pomoću *JwtService* usluge koja u pozivu za kreiranje tokena, kao što je vidljivo na slici 4.38, prima podatke, tajni ključ i postavlja vrijeme istjecanja token. Izvršavanjem *sign* funkcije kreira se token koji u sebi sadrži predani podatak te sve potrebne informacije koje će biti obrađene na klijentskoj strani za pristup web sustavu.

Sl. 4.38. *Postupak stvaranja tokena*

Ruta za registraciju prima unutar parametara zahtjeva objekt za registraciju koji sadržava e-poštu, ime i prezime te lozinku. Nakon primanja podataka, oni se prosljeđuju *authenticationService* usluzi i funkciji *register* koja također prima isti takav objekt za registraciju. Unutar funkcije za registraciju koja se nalazi u usluzi, prvo se poziva provjera predanih podataka te se rezultati spremaju u varijablu. U slučaju da postoje greške, prekida se izvršavanje registracije i korisniku se vraćaju informacije što ne valja s podacima. Prema slici 4.39 se mogu vidjeti provjere koje se vrše nad podacima, gdje se provjeravaju jesu li svi podaci unutar objekta u tekstualnom obliku te jesu li zapravo i poslani. Sve te provjere se vrše pomoću biblioteke Joi koja služi za validaciju podataka u JavaScriptu te za kreiranje sheme podataka. Pomoću te biblioteke moguće je postaviti i vlastite poruke za određene greške što pomaže u razvoju aplikacije koja mora biti prevedena na neki drugi jezik.

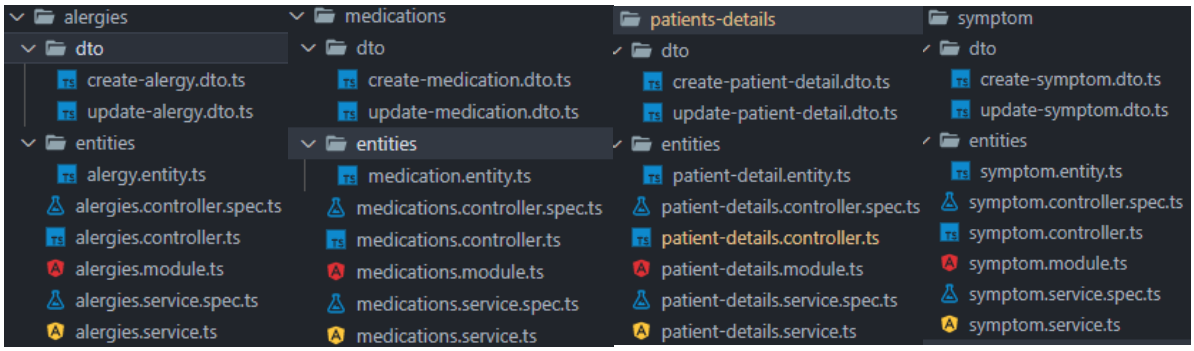
Sl. 4.39. *Prikaz podataka za provjeru i funkcije koja izvršava registraciju*

U slučaju da ne postoje greške prilikom provjere podataka, funkcija se nastavlja izvoditi unutar *try* dijela funkcije. Prije svega, lozinka koja je pristigla se *hashira* pomoću bcrypt biblioteke te se sprema zajedno sa ostalim podacima za registraciju u „users“ tablicu. Prilikom spremanja zamjenjuje se obična lozinka s *hash* vrijednošću te lozinke, kako bi se povećala sigurnost podataka te se novo kreiran korisnik, bez lozinke, vraća u odzivu natrag korisniku. Da se dogodila greška prilikom kreiranja ili nekog drugog dijela funkcije, *catch* blok unutar funkcije bi uhvatio tu grešku te ju vratio u dozivu s određenom porukom.

4.3.2. Relacije među tablicama korisnici, alergije, simptomi, pacijenti, lijekovi

Prema slici 4.35 je vidljiva povezanost svih tablica i relacija među njima. Zbog toga potrebno je opisati tablice zajedno kako bi se moglo bolje razumjeti proces kreiranja baze podataka i na koji način je olakšala proces predlaganja lijekova.

Prva tablica u bazi koja je kreirana je tablica s korisnicima. Prethodno je opisano kako funkcionira registracija i prijava te generiranje tokena kao i struktura aplikacije. U ovim tablicama se najbolje može vidjeti spomenuta struktura gdje svaka komponenta, tj. svaka funkcionalnost ima svoju tablicu i svoj modul te usluge koje su zadužene za podatke u tim tablicama. Na slici 4.40 je prikazana struktura ostatka aplikacije. Unutar svake od komponenti se nalaze entiteti koji predstavljaju pojedinu tablicu. Svaki entitet u sebi sadrži naziv, stupce koje će sadržavati, primarne ključeve, strane ključeve te opis podataka koji će se nalaziti u stupcima. Određeni entiteti u sebi sadrže i relacije u slučaju da imaju stupac čiji podaci se odnose na neku drugu tablicu.



Sl. 4.40. *Prikaz strukture komponenata alergije, simptoma, lijekova i pacijenata*

Najjednostavniji primjer entiteta u aplikaciji je entitet korisnika jer sadrži jednostavne podatke i tipove podataka te ima relativno malo stupaca. Na slici 4.41 se može vidjeti kako se mogu definirati stupci u tablici, na koji način se mogu postaviti ograničenja stupaca te koji stupac služi kao primarni ključ.

```
@Entity()
class User {
  @PrimaryGeneratedColumn()
  public id?: number;

  @Column({ unique: true })
  public email: string;

  @Column()
  public name: string;

  @Column()
  public password: string;

  @Column({ default: 'user' })
  public role: string;
}
```

Sl. 4.41. *Prikaz entiteta korisnika*

Za razliku od entiteta korisnika, u aplikaciji postoje i puno složeniji entiteti poput entiteta lijekova što je normalno jer se ipak najviše podataka mora spremati u lijekove. Kod entiteta lijekova vidljive su relacije koje uvelike olakšavaju povezivanje dvije tablice u bazi. Relacije mogu biti „više prema više“, „jedna prema jednoj“ i „više prema jednoj“. U entitetu lijeka koriste se dvije „više prema više“ relacije koje povezuju tablicu lijekove s tablicom alergije i simptoma. Tako definirana relacija označava da više lijekova može sadržavati više alergija i više simptoma, no to isto znači i za suprotno, tj. da više alergija i više simptoma mogu sadržavati više lijekova.

Prema idejnom modelu aplikacije takve relacije su točno ono što je potrebno kako bi se mogli opisati i povezati potrebni podaci koji će kasnije biti potrebni za filtriranje lijekova prema simptomima, alergijama i drugim podacima. Prema slici 4.42 se može vidjeti primjer kako se definira relacija „više prema više“ i time povezuju tablice. U ovom slučaju tablica simptoma i

tablica lijekova je povezana tako da se automatski kreira tablica relacije pozivom *@JoinTable* s nazivom „medications_symptoms“, gdje se sprema identifikacijski broj lijeka u stupac *medicationId* i identifikacijski broj simptoma u stupac *symptomId*. Analogno tome se kreira relacijska tablica za lijekove i alergije. Unutar „više prema više“ relaciji se definira trebaju li se automatski spremati podatci u relacijsku tablicu kada se postave podaci unutar tablice lijekova ili ne, što je vidljivo s oznakom *cascade* koja je postavljena na istinito stanje. Osim toga, postavljena je oznaka *eager* na istinito stanje, koja prilikom dohvaćanja lijekova vraća i sve simptome koji su vezani za taj lijek te oznaka *onDelete* na vrijednost „CASCADE“ što označava da se podaci vezani za lijek automatski brišu iz relacijske tablice kada se lijek obriše. Na isti način je definirano i za relacijsku tablicu s alergijama. Prema navedenom, može se primijetiti kako dodavanje relacija već dodaje jedan sloj filtriranja pri dohvaćanju podataka i kako bi se mogle relacije iskoristiti u kasnijem filtriranju lijekova prema podacima.

```
@Entity()
export class Medication {
  @PrimaryGeneratedColumn()
  public id?: number;
  @Column()
  public name: string;
  @Column()
  public description: string;
  @Column()
  public canBeUsedWhilePregnantOrBreastFeed: boolean;
  @Column('text', { array: true })
  public contraindications: string[];
  @Column('text', { array: true })
  public interactions: string[];
  @Column('json')
  public sideEffects: JSON;
  @Column('json', { default: { byAge: [], byWeight: [] } })
  public dosage: JSON;
  @ManyToMany(() => Symptom, (symptom) => symptom.medications, {
    cascade: true,
    eager: true,
    onDelete: 'CASCADE',
  })
  @JoinTable({
    name: 'medications_symptoms',
    joinColumn: {
      name: 'medicationId',
      referencedColumnName: 'id',
    },
    inverseJoinColumn: {
      name: 'symptomId',
      referencedColumnName: 'id',
    },
  })
  public symptoms: Symptom[];
  @ManyToMany(() => Alergy, (alergy) => alergy.medications, {
    cascade: true,
    eager: true,
    onDelete: 'CASCADE',
  })
  @JoinTable({
    name: 'medications_alergies',
    joinColumn: {
      name: 'medicationId',
      referencedColumnName: 'id',
    },
    inverseJoinColumn: {
      name: 'alergyId',
      referencedColumnName: 'id',
    },
  })
  public alergies: Alergy[];
}
```

Sl. 4.42. *Prikaz entiteta lijeka*

Relacije mogu biti jednosmjerne i dvosmjerne, a u slučaju lijekova, alergija i simptoma, koriste se dvosmjerne relacije. Dvosmjerne relacije predstavljaju relacije gdje se relacijski „više prema više“, tj. *@ManyToMany*, dekorater, pojavljuje na obje strane relacije. Primjer druge strane relacije je vidljiv na slici 4.43 gdje je definirana spomenuta relacija, ali ne postoji *@JoinTable* dekorater. Takav dekorater se nalazi isključivo na onoj strani relacije koja je vlasnik relacije.

```
@Entity()
export class Alergy {
  @PrimaryGeneratedColumn()
  public id: number;

  @Column({ unique: true })
  public name: string;

  @ManyToMany(() => Medication,
    (medications) => medications.alergies)
  medications: Medication[];
}
```

```
@Entity()
export class Symptom {
  @PrimaryGeneratedColumn()
  public id: number;

  @Column({ unique: true })
  public name: string;

  @ManyToMany(() => Medication,
  (medications) => medications.symptoms)
  medications: Medication[];
}
```

Sl. 4.43. *Prikaz entiteta alergije i simptoma*

Dvosmjerna relacija u ovom slučaju omogućuje lakše dohvaćanje svih lijekova koji su vezani za određenu alergiju. Na isti način je definirana relacija kod entiteta simptoma te se na takav način mogu dohvatiti svi lijekovi koji su povezani s određenim simptomom. Takav način dohvaćanja se koristi u funkciji za filtriranje lijekova uz još dodano filtriranje prema drugim podacima. Isto tako, ne nalaze se relacije isključivo kod lijekova. Entitet detalja o pacijentu također sadrži „više prema više“ relaciju prema alergijama te relaciju „više prema jednom“ prema korisniku. Ovakav način postavljanja relacije je potreban jer više pacijenata može imati istog liječnika, tj. korisnika, koji ih je unio te više pacijenata može imati više alergija na koje su alergični.

Alergije kod pacijenata se automatski dohvaćaju kao i kod lijekova jer se, prilikom filtriranja lijekova za pacijenta, moraju dohvatiti njegovi detalji iz baze pa tako i alergije koje su vezane s njim. Takvo automatsko dohvaćanje relacije je omogućeno prethodno navedenim *eager* parametrom unutar relacije. Također, ista funkcionalnost se može izvesti postavljanjem parametra relacije koju se želi dohvatiti unutar poziva funkcije za dohvaćanje podataka kao što je prikazano na slici 4.44.

```
async findOne(id: number) {
  return await this.patientDetailsRepository.findOne(id, {
    relations: ['alergies'],
  });
}
```

Sl. 4.44. *Prikaz dohvaćanja detalja pacijenta prema identifikacijskom broju*

Svako dohvaćanje pojedinačnih unosa podataka, npr. lijekova, pacijenata, simptoma i drugih, se obavlja prema identifikacijskom broju ili u nekim slučajevima prema nazivu. Dohvaćanje pojedinačnih podataka izvodi se *findOne* funkcijom kojoj se predaje parametar prema kojem se pronalaze podatci, a mogu se postaviti dodatne postavke za dohvaćanje poput relacija, koji podaci iz pronađenog objekta će se vratiti itd. Uz *findOne*, u uslugama se koriste i *find* funkcija za pronalaženje liste podataka koji odgovaraju predanom kriteriju. Kriteriji koji se mogu postaviti su jednaki kao i kod *findOne* funkcije uz dodatne funkcionalnosti popu *skip* i *take* koji služe za

vraćanje određenog dijela liste ili za postavljanje maksimalnog broja podataka koji se trebaju vratiti.

```
findAll() {
  return this.symptomRepository.find({
    order: { name: 'ASC' },
  });
}

findALlSymptomsByName(name: string) {
  return this.symptomRepository.find({ where: { name } });
}
```

Sl. 4.45. *Prikaz dohvaćanja svih simptoma i svih simptoma prema nazivu*

Opis ovih funkcija olakšava razumijevanje postupaka koji se izvode pri kreiranju i brisanju podataka iz baze. To je posebno naglašeno kod lijekova gdje spremanje lijeka u bazu, sličnim postupkom kao kod registracije korisnika, pozivanjem *save* funkcije nije dovoljno kako bi se povezali lijekovi s novim alergijama i simptomima. Također je naglašeno kod brisanja lijeka iz baze gdje se treba provjeriti prilikom brisanja postoje li drugi lijekovi za koje je određeni simptom ili alergija vezan. U slučaju postojanja, simptom ili alergija neće biti obrisani iz baze, a u suprotnom se zajedno sa lijekom s kojim su povezani brišu i time su liste alergija i simptoma ažurirane.

Na slici 4.46. se može vidjeti postupak spremanja lijeka u bazu. Prilikom spremanja, prva provjera koja se vrši je provjera naziva lijeka. Dva lijeka s istim imenom se ne mogu nalaziti u bazi i prema tome se vraća odgovarajuća greška korisniku. U nastavku, kao i kod registracije korisnika, se vrši provjera poslanih podataka, jesu li odgovarajućeg tipka i jesu li uopće poslani. U slučaju da neki od podataka ne odgovara, šalje se greška s porukom i nazivom polja koje ne odgovara shemi, a ako svi podaci prolazi provjeru, dolazi se zatim do dodavanja novih alergija i simptoma kako bi se lijek mogao povezati s njima.

Unutar podataka koji se šalju za kreiranje lijeka nalaze se i polja s alergijama i simptomima u kojem svaki objekt alergije ili simptoma u sebi ima naziv. U funkciji na slici 4.46 se provjerava postojanje tog naziva u bazi te se u slučaju ne postojanja dodaje novi unos alergije ili simptoma u tablicu. Nakon dodavanja novih alergija i simptoma lijek je spreman za dodavanje u tablicu. Pozivom *create* funkcije se kreira nova instanca entiteta u koju se spremaju podaci koji su definirani u objektu za kreiranje. Instanca novog lijeka se sprema u privremenu varijablu te se privremena varijabla zatim predaje *save* funkciji koja vrši spremanje lijeka u bazu.

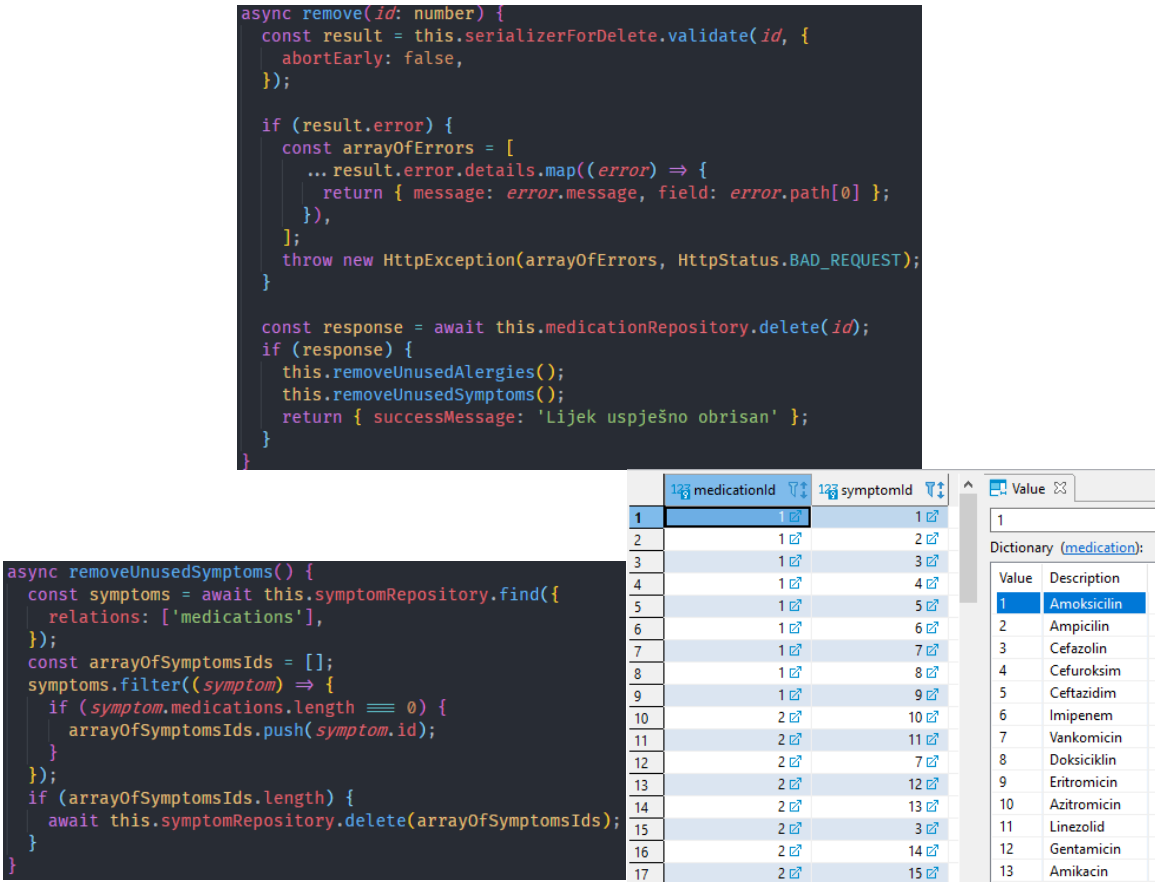
```
async create(createMedicationDto: CreateMedicationDto) {
  if (
    await this.medicationRepository.findOne({
      name: createMedicationDto.name,
    })
  ) {
    throw new HttpException(
      { message: 'Lijek s ovim nazivom već postoji', field: 'name' },
      HttpStatus.BAD_REQUEST,
    );
  }

  const result = this.serializer.validate(createMedicationDto, {
    abortEarly: false,
  });

  if (result.error) {
    const arrayOfErrors = [
      ...result.error.details.map((error) => {
        return { message: error.message, field: error.path[0] };
      }),
    ];
    throw new HttpException(arrayOfErrors, HttpStatus.BAD_REQUEST);
  }
  createMedicationDto.alergies.map(async (alergy) => {
    if (!this.alergyRepository.find({ name: alergy.name })) {
      await this.alergyRepository.save(alergy);
    }
  });
  createMedicationDto.symptoms.map(async (symptom) => {
    if (!this.symptomRepository.find({ name: symptom.name })) {
      await this.symptomRepository.save({ name: symptom.name });
    }
  });
  const newMedication = this.medicationRepository.create(createMedicationDto);
  await this.medicationRepository.save(newMedication);
  return { ...newMedication, successMessage: 'Lijek uspješno kreiran' };
}
```

```
export class CreateMedicationDto {
  id: number;
  name: string;
  description: string;
  canBeUsedWhilePregnantOrBreastFeed:
    boolean;
  contraindications: string[];
  interactions: string[];
  sideEffects: JSON;
  dosage: JSON;
  symptoms: Symptom[];
  alergies: { name: string }[];
}
```

Sl. 4.46. *Prikaz kreiranja lijeka i objekt za kreiranje*

Postupak brisanja lijeka je, u odnosu na postupak kreiranja, jednostavniji. Jedino što dodaje kompleksnost brisanju je provjera postoji li ovisnost nekog drugog lijeka o simptomu i alergiji koja je vezana za lijek koji se briše. Na slici 4.47 je prikazana funkcija za brisanje lijeka prema predano identifikacijskom broju. Prilikom brisanja provjerava se postoji li identifikacijski broj i je li ispravnog oblika koji je potreban za brisanje. Nakon provjere, lijek se briše iz baze te se zbog postavljene oznake *onDelete* na „CASCADE“ automatski brišu i vrijednosti iz tablica relacija koje su bile vezane za lijek.

Završetkom brisanja lijeka, pozivaju se funkcije za brisanje ne korištenih simptoma i alergija. U njima se dohvaća cijela lista alergija i simptoma s relacijom prema lijekovima te se provjerava postoje li simptomi i alergije koji nisu korišteni u nekom od lijekova. U slučaju postojanja simptoma koji nemaju niti jedan lijek u relaciji, dodaje se njegov identifikacijski broj u polje, a ako polje nakon provjere sadrži identifikacijske brojeve, poziva se funkcija *delete* koja briše simptome po identifikacijskom broju iz tablice simptoma. U slučaju da se funkcija poziva na *alergyRepository* varijabli, svaka alergija koja sadrži identifikacijski broj kao u polju bila bi obrisana. Takvim brisanjem se ažurira lista simptoma i alergija koje korisnik može koristiti za pacijenta kako bi uvijek imao valjane podatke koji su vezani za barem jedan lijek.



Sl. 4.47. *Prikaz brisanja lijeka i ne korištenih simptoma te tablice relacije lijekova i simptoma*

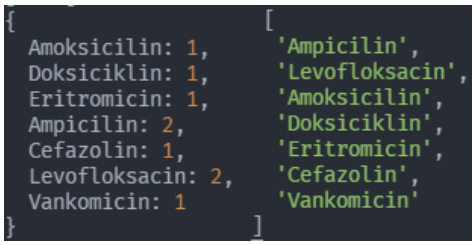
Takvi ažurirani podaci se zatim koriste u funkciji koja vrši filtriranje lijekova prema predanim simptomima. Kao što je ranije rečeno, dvosmjerna relacija kod simptoma se koristi u filtriranju te omogućuje lakši pronalazak relevantnih lijekova. Na slici 4.48 se može vidjeti primjena dohvaćanja lijekova kroz relaciju sa simptomima, korištenje filtriranja prema alergijama i informaciji doji li pacijent ili je trudan te dohvaćanje detalja o pacijentu prema identifikacijskom broju koje je ranije objašnjeno. Prema tome, *findBySymptoms* funkcija prima parametre liste simptoma koje korisnik šalje prilikom poziva funkcije za filtriranje na korisničkoj strani te identifikacijski broj pacijenta za kojeg se vrši filtriranje. U privremenu varijablu *patientDetails* spremaju se dohvaćeni detalji o pacijentu kako bi mogli biti korišteni kasnije u filtraciji te kako bi se iz njih izvukla lista alergija. Alergije se koriste za filtriranje lijekova bez obzira na to je li predana lista simptoma ili ne.

```
async findBySymptoms(symptoms: Symptom[], userId: number) {
  const patientDetails = await getManager()
    .getRepository(PatientDetail)
    .findOne({ id: userId });

  const alergiesByName = [
    ...patientDetails.alergies.map(
      (patientDetailAlergy) => patientDetailAlergy.name,
    ),
  ];
  if (symptoms && symptoms.length) {
    const symptomsFound = await this.symptomRepository.findByIds(
      [...symptoms.map((requestSymptom) => requestSymptom.id)],
      { relations: ['medications'] },
    );
    const extractedMedicationsFromSymptoms = symptomsFound.map(
      (symptom) => symptom.medications,
    );
    const flattenedArrayOfMedications = [].concat(
      [],
      ...extractedMedicationsFromSymptoms,
    );

    const uniq = flattenedArrayOfMedications
      .map((medication) => {
        return { count: 1, medication: medication };
      })
      .reduce((a, b) => {
        a[b.medication.name] = (a[b.medication.name] || 0) + b.count;
        return a;
      }, {});

    const sorted = Object.keys(uniq).sort((a, b) =>
      uniq[a] > uniq[b] ? -1 : uniq[a] < uniq[b] ? 1 : 0,
    );
    const medications = await this.medicationRepository.find({
      where: { name: In(sorted) },
    });
    const sortedMedications = sorted.map((medication) =>
      medications.find((med) => med.name === medication),
    );
    return sortedMedications.filter((medication) =>
      patientDetails.pregnantOrBreastFeed
        ? !medication.alergies.find((medicationAlergy) =>
            alergiesByName.includes(medicationAlergy.name),
          ) &&
          medication.canBeUsedWhilePregnantOrBreastFeed &&
          medication
        : !medication.alergies.find((medicationAlergy) =>
            alergiesByName.includes(medicationAlergy.name),
          ) && medication,
    );
  } else {
    const medications = await this.medicationRepository.find();
    return medications.filter((medication) =>
      patientDetails.pregnantOrBreastFeed
        ? !medication.alergies.find((medicationAlergy) =>
            alergiesByName.includes(medicationAlergy.name),
          ) &&
          medication.canBeUsedWhilePregnantOrBreastFeed &&
          medication
        : !medication.alergies.find((medicationAlergy) =>
            alergiesByName.includes(medicationAlergy.name),
          ) && medication,
    );
  }
}
```

Sl. 4.48. *Prikaz funkcije za filtriranje lijekova prema simptomima i detaljima pacijenta*

U slučaju da je predana lista simptoma, u privremenu *symptomsFound* varijablu dohvaćaju se svi simptomi prema identifikacijskom broju i s relacijom prema lijekovima. To omogućuje da svaki objekt simptoma u sebi ima i listu lijekova za koje se koristi. Zatim, iz takve liste simptoma s lijekovima za koje se koriste, izvlače se svi liste lijekova i spremaju u novo polje na takav način da *extractedMedicationsFromSymptoms* bude polje polja svih lijekova za predane simptome.

Kako bi se olakšalo korištenje i omogućilo lakše prolaženje kroz polje s lijekovima, ono se izravnava tako da ostane isključivo jedno polje sa svim lijekovima za predane simptome. Takvo se polje sprema u *flattenedArrayOfMedications* varijablu te se koristi u nastavku za računanje broja pojavljivanja određenog lijeka. Takvo računanje, koje je prikazano na slici 4.49 objektom s nazivima lijekova i njima pridruženim brojevima, zapravo predstavlja objekt s lijekovima koji zadovoljavaju simptome koji su predani. Što je veći broj pokraj naziva lijeka, to se više simptoma unutar lijeka podudara s predanim simptomom. Prema tome, unutar spomenutog objekta se nalaze svi lijekovi koje je potrebno poslati natrag korisniku.



Sl. 4.49. *Izračunata ponavljanja lijekova i sortirana lista lijekova po broju ponavljanja*

Kako bi vratili korisniku lijekove i prikazali ih smislenim redom, od onog lijeka koji ima najviše podudaranja prema onom lijeku koji ima najmanje, takav objekt se treba sortirati. Sortiranje se vrši uspoređivanjem broja ponavljanja te nakon sortiranja se dobije lista naziva lijekova kao na slici 4.49. Tako sortirani nazivi se predaju funkciji za dohvaćanje lijekova gdje se dohvaćaju isključivo lijekovi koji se nalaze u listi naziva. Dohvaćeni lijekovi se spremaju u varijablu *medications* te se koriste u nastavku kako bi unutar liste sortiranih naziva lijekova zamijenili nazivi lijekova s objektom lijeka za kojeg se naziv odnosi.

Zadnje filtriranje lijekova se odvija na kraju pri vraćanju lijekova korisniku gdje se koriste spremljeni podaci o pacijentu unutar varijable *patientDetails* i uspoređuju se s dohvaćenim lijekovima. U slučaju kada pacijent ima postavljeno stanje da doji ili je trudan, tada se to stanje također ubraja u filtriranje, a u suprotnom se provjeravaju isključivo alergije. Prema tome, ako pacijent ima postavljeno stanje da doji ili je trudan, ako se lijek smije koristiti tijekom trudnoće i ako se unutar lijeka ne nalaze alergije koje ima pacijent, onda se lijek vraća u listi lijekova za prijedlog. U slučaju da pacijent ne doji i nije trudan, tada se isključivo provjerava podudaranje alergija lijeka i pacijenta. Kada se podudari alergija na detaljima pacijenta s onom u lijeku, tada se taj lijek preskače i neće biti vraćen u odzivu na klijentsku stranu.

5. TESTIRANJE WEB SUSTAVA ZA STVARANJE PREPORUKA PRIKLADNIH LIJEKOVA

Nakon razvijanja web sustava za stvaranje preporuka prikladnih lijekova, potrebno je provjeriti rad funkcionalnosti na klijentskom i poslužiteljskoj strani kao i stabilnost aplikacije postavljene na udaljenog poslužitelja. Takvo testiranje omogućuje pronalazak grešaka u kodu i logici funkcionalnosti koje je teže primijetiti pregledavanjem koda i ručnim testiranjem. Prema tome, testiranja će se provoditi na klijentskoj i poslužiteljskoj strani kako bi se provjerile njihove funkcionalnosti, kao i stres test kada je sustav postavljen na udaljenog poslužitelja koji će, u ovom slučaju, biti Heroku.

5.1. Korišteni alati za testiranje

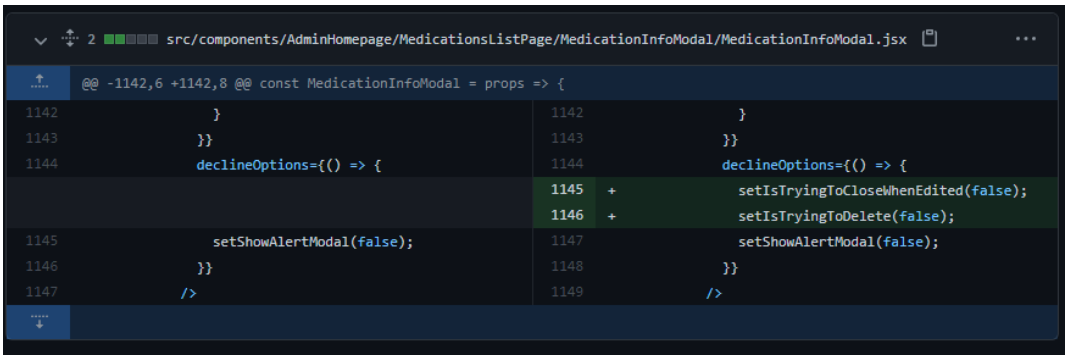
Za testiranje klijentske i poslužiteljske strane, koristit će se Visual Studio Code, njegov ugrađeni terminal koji će se koristiti za pokretanje testova, kao i razna proširenja dostupna unutar alata jer su obje strane pisane u JavaScript okviru. Proširenja omogućuju pokretanje lokalnog poslužitelja koji će biti korišten za pregled pokrivenosti koda testovima, korištenje kratica koje omogućuju brzo pisanje testova i koda te korištenje proširenja koji provjeravaju sintaksu tijekom pisanja koda i upozoravaju na greške. Prilikom stres testiranja web sustava postavljenog na udaljenog poslužitelja koristio se alat JMeter čije mogućnosti su spomenute ranije.

5.2. Testiranje korisničkog sučelja i testiranje korisničkog iskustva

Testiranje korisničkog sučelja i iskustva se odvijao na dva načina. Prvi je ručno testiranje funkcionalnosti i zahtjeva na korisničkoj strani, a drugo je automatizirano testiranje pisanjem testova i generiranjem izvješća. Ručnim testiranjem se mogu primijetiti veći propusti i greške kod korisničkog iskustva i funkcionalnosti, dok se pisanjem testova detaljnije provjeravaju funkcionalnosti te se mogu uhvatiti i manje greške koje možda nisu vidljive prilikom ručnog testiranja.

Ručnim testiranjem najveće su se izmjene provodile tijekom razvoja web sustava. To su većinom greške u logici koje se primjećuju prolaskom kroz tok korisnika koji će koristiti web sustav. Primjer tome se može vidjeti na slici 5.1 gdje se ispravila greška prilikom prikazivanja krivog upozorenja kada administrator pokušava izaći iz detaljnog prikaza o lijeku, a kada je izmijenio neki od podataka. U tom slučaju prikaz upozorenja treba pitati korisnika želi li napustiti prikaz, a u ovom slučaju je pitalo korisnika želi li obrisati unos.

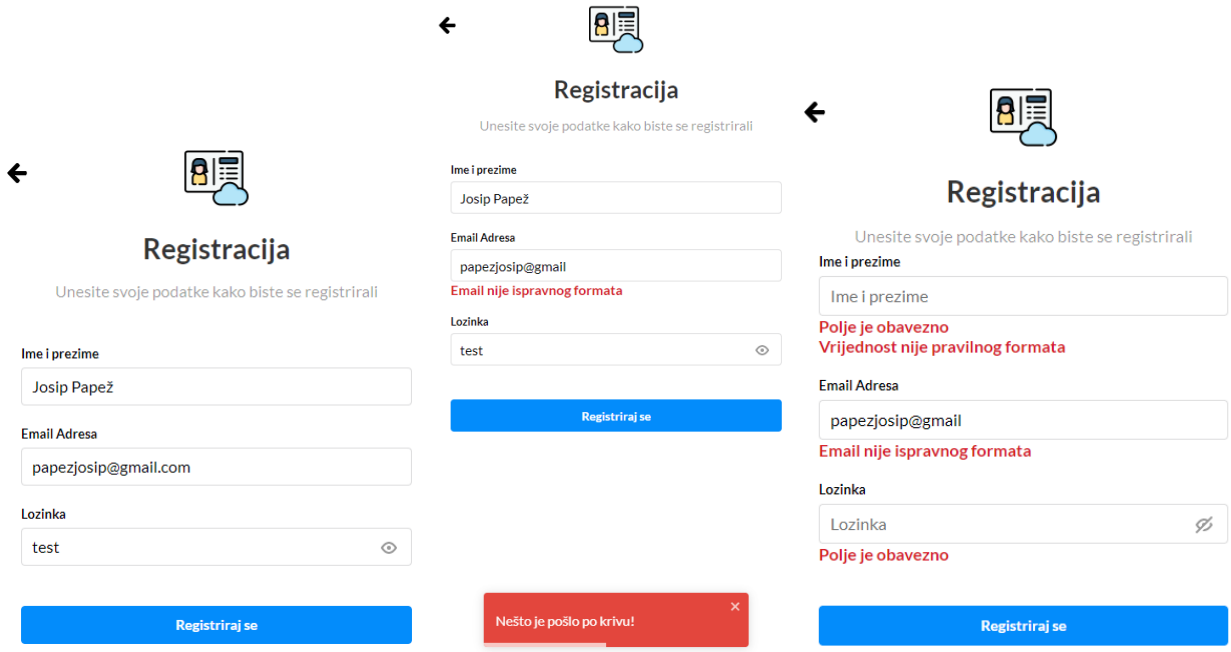
Greška se dogodila zato što vrijednost koja se postavi kada korisnik pokušava obrisati lijek ostane u istinitom stanju te prilikom svakog izlaza iz detaljnog prikaza pojavljuje se spomenuto upozorenje za brisanje. Zbog toga, greška se proširila i na redoslijed izvođenja funkcija prilikom pritiska na potvrdu unutar prikaza upozorenja. U slučaju da korisnik pritisne na potvrdu prilikom izlaska iz detaljnog prikaza, lijek bi bio obrisan jer je vrijednost za brisanje ostala u istinitom stanju. Prema tome, vraćanjem vrijednosti za brisanje i za zatvaranje detaljnog prikaza kada su podaci izmijenjeni na njihove početne vrijednosti je popravilo grešku.



Sl. 5.1. *Promjene za prikaz upozorenja na detaljnom prikazu lijeka*

Ovakvi tipovi greške i mnogi drugi su većinom popravljeni tijekom razvoja web sustava i mogu se pronaći u izmjenama spremljenim na GitHub repozitoriju. Za razliku od takvih grešaka popravljenih tijekom razvoja web sustava, vrlo malo grešaka je bilo koje su se naknadno popravile poput greške na slici 5.1.

Osim popravljanja grešaka, ručnim testiranjem se provjerava rad nekog manjeg ili većeg dijela web sustava. Najbolji način za provjeru ispravnosti funkcionalnosti je da se one provjere prolaskom kroz sustav jedna po jedna. Prema tome, prva funkcionalnost koja se provjerava je registracija kako bi korisnik mogao pristupiti stranici. Odabirom opcije za registraciju i unošenjem potrebnih podataka u formu za registraciju, kao što je prikazano na slici 5.2, ispunjavaju se svi uvjeti za registraciju korisnika. Pritiskom na tipku registriraj se, korisnik je registriran i automatski se prijavljuje u sustav. U slučaju da korisnik ne unese ispravnu e-poštu ili da ostavi bilo koje polje prazno, prikazuju se greške vezane za polja na registraciji u kojima se nalazi greška. Prilikom testiranja iskustva prijave i registracije, moglo se primijetiti kako se ne prikazuju greške za e-poštu i lozinku te su one naknadno dodane.



Sl. 5.2. *Ispunjena registracijska forma i prikaz grešaka na registracijskoj formi*

Ispis grešaka se provodi po nazivu polja za koje se odnosi. Primjer toga je vidljiv na slici 5.3 gdje je naknadno nakon testiranja dodana provjera grešaka za polje lozinke.

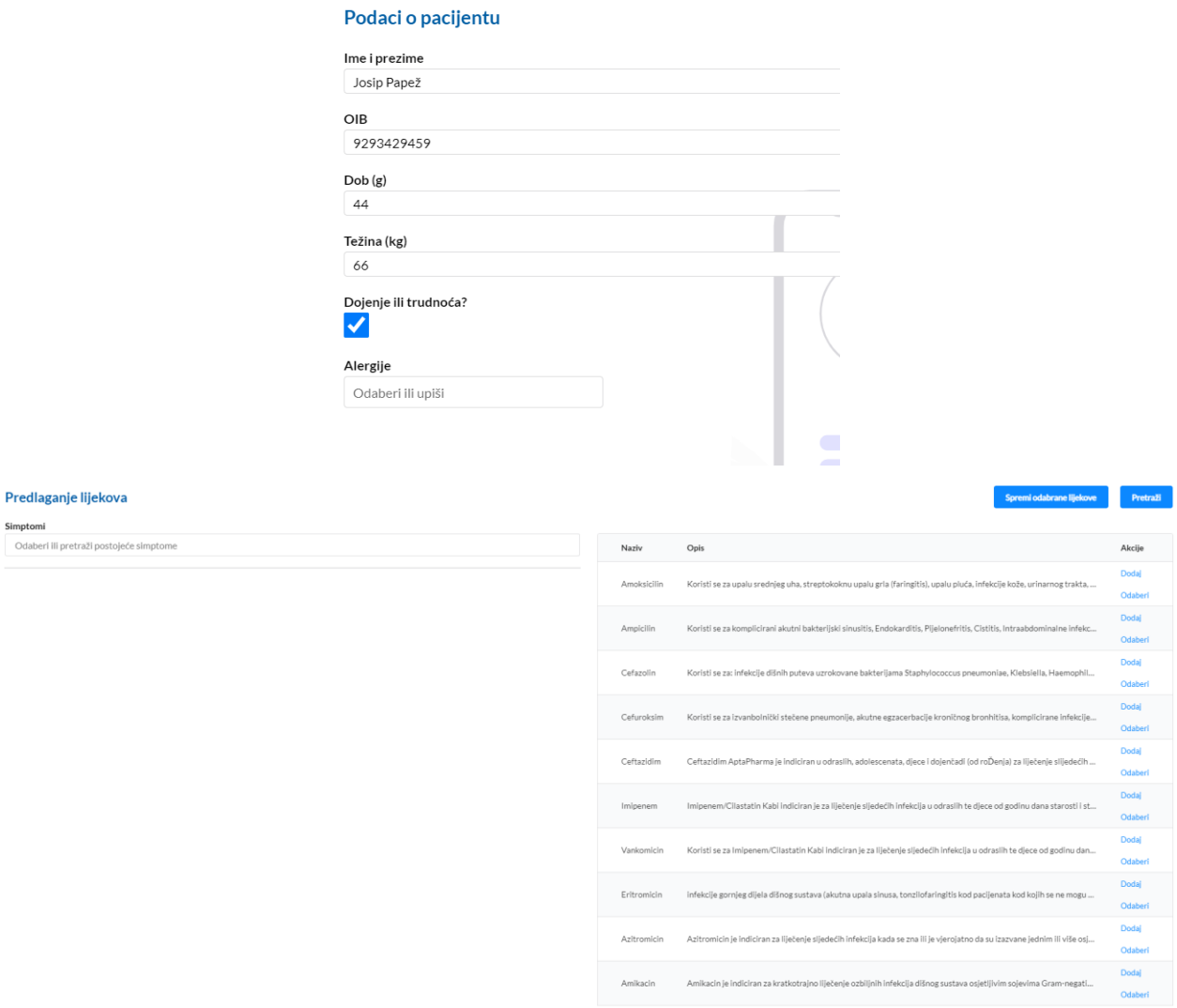
```
<div className="password-input">
  <input
    className="content-container__password"
    type={displayPassword ? 'text' : 'password'}
    value={password ? password : ''}
    onChange={e => {
      e.stopPropagation();
      e.preventDefault();
      setPassword(e.target.value);
    }}
    placeholder="Lozinka"
    name="password"
  />
  <div
    onClick={() => setDisplayPassword(!displayPassword)}
    className={classNames({
      'password-icon': true,
      'hide-password': !displayPassword,
      'show-password': displayPassword,
    })}
  />
</div>
```

```
<div className="password-input">
  <input
    className="content-container__password"
    type={displayPassword ? 'text' : 'password'}
    value={password ? password : ''}
    onChange={e => {
      e.stopPropagation();
      e.preventDefault();
      setPassword(e.target.value);
    }}
    placeholder="Lozinka"
    name="password"
  />
  <div
    onClick={() => setDisplayPassword(!displayPasswo
    className={classNames({
      'password-icon': true,
      'hide-password': !displayPassword,
      'show-password': displayPassword,
    })}
  />
+ {errorMessage &&
+   errorMessage.length > 0 &&
+   errorMessage.find(
+     error => error.field === 'password'
+   ) && (
+   <div className="error">
+     {
+       errorMessage.find(
+         error => error.field === 'password'
+       ).message
+     }
+   </div>
+ )}
</div>
```

Sl. 5.3. *Prikaz dodanog ispisa greške za polje lozinka*

Nakon prijave ili registracije, korisnik je preusmjeren na početnu stranicu kao što je u prethodnom poglavlju prikazano. Korisnik zatim može dodati novog pacijenta za kojeg želi unositi podatke i filtrirati lijekove. Kao i kod registracije, u slučaju da korisnik ostavi obavezna polja prazna ili upisane krive podatke prikazat će se odgovarajuće greške na poljima za koje se greške odnose.

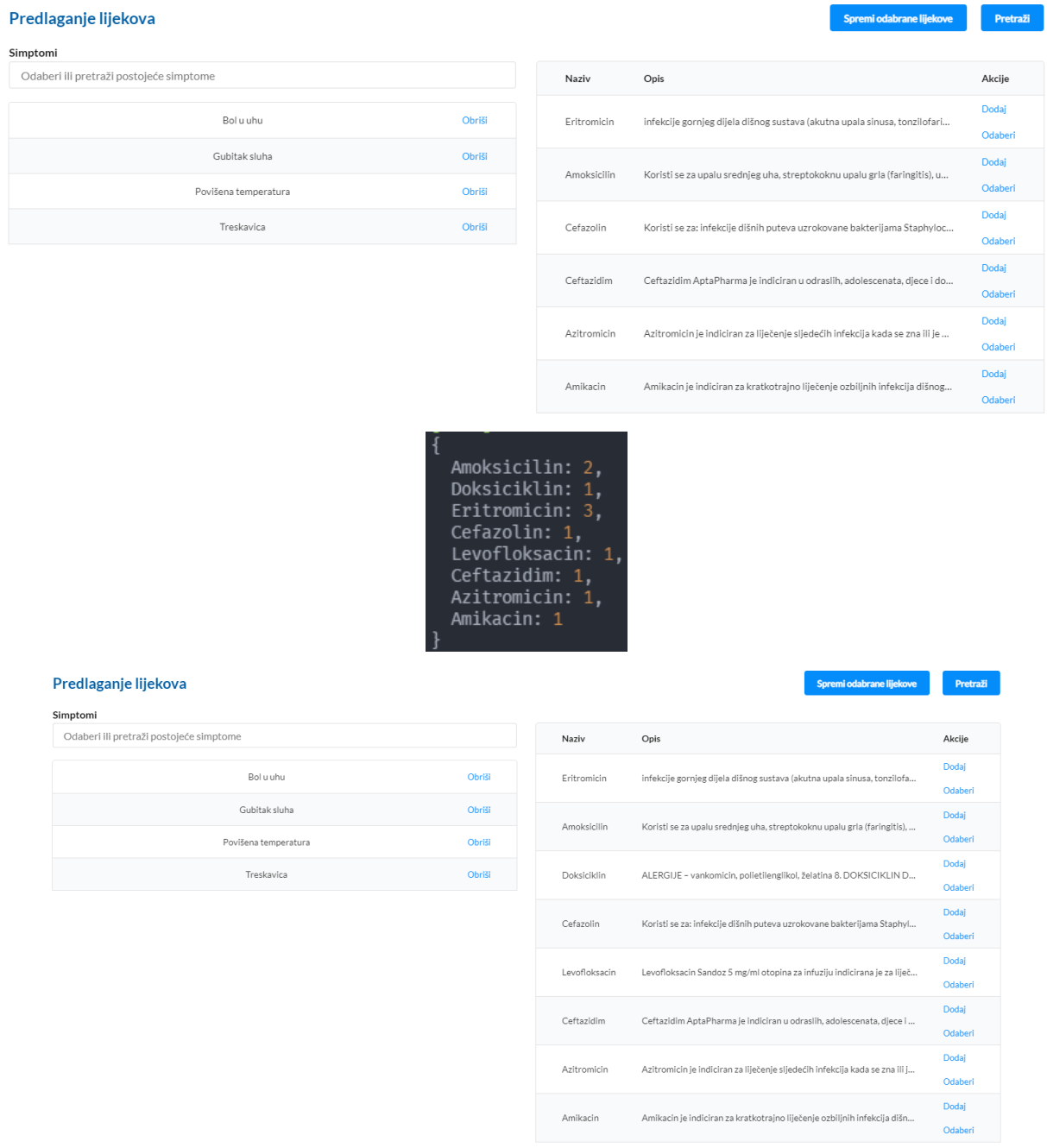
Uz to, pritiskom na tipku „Novi pacijent“ se brišu svi unosi iz polja za unos te se brišu sve greške kako bi se mogao dodati novi pacijent. Dodavanjem novog pacijenta korisniku se prikazuje nova stavka u izborniku za prijedlog lijekova. Na tom dijelu stranice korisnik pretražuje lijekove za pacijenta prema određenim parametrima. Parametri koji se uzimaju u obzir su doji li pacijent ili je trudan, alergije pacijenta i simptomi pacijenta. Na slici 5.4 se može vidjeti prikaz predloženih lijekova za pacijenta koji su filtrirani prema podacima pacijenta. U ovom slučaju, lijekovi su filtrirani prema podatku doji li pacijent ili je trudan i može se vidjeti da je vraćeno 10 lijekova koji se smiju koristiti tijekom trudnoće ili dojenja od 15 lijekova koji su trenutno uneseni u bazu.



Sl. 5.4. *Prikaz detalja pacijenta i popisa lijekova za pacijenta*

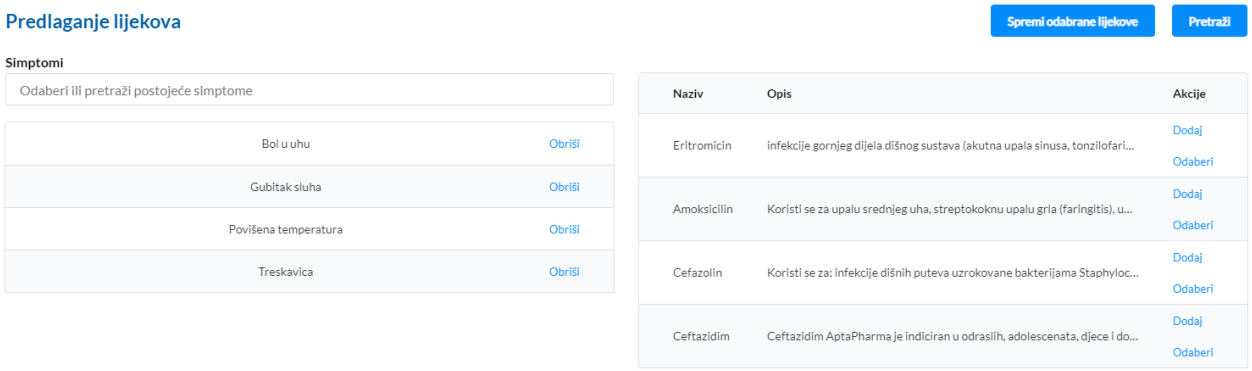
Kako bi se provjerila ispravnost predlaganja, može se uzeti za primjer da pacijent ima upalu srednjeg uha. U bazi lijekova postoje dva lijeka, Amoksicilin i Eritromicin, koji se koriste za upalu srednjeg uha i oba se smiju koristiti prilikom dojenja ili trudnoće. Zbog toga, oba ova lijeka se trebaju prikazati u listi predloženih lijekova za unesene simptome upale srednjeg uha.

Na slici 5.5 se mogu vidjeti neki od simptoma koji obilježavaju upalu srednjeg uha kao i predloženi lijekovi za unesene simptome. Iz predloženih lijekova se zaključuje kako filtriranje lijekova funkcionira za ovu bolest te se također na istoj slici može vidjeti koliko simptoma se podudara za pojedini lijek koji je predložen. Zbog odabrane stavke da pacijent doji ili je trudan, u listi predloženih lijekova ne prikazuju se Levofloksacin i Doksiciklin jer se oni ne smiju koristiti tijekom trudnoće ili dojenja. Prema tome, ako se promjeni vrijednost dojenja ili trudnoće za pacijenta, onda se za iste simptome dobiju još dva dodatna lijeka kao što je prikazano na slici 5.5 jer se mogu koristiti lijekovi i koji se ne smiju koristiti u trudnoći i tijekom dojenja.



Sl. 5.4. *Predlaganje lijekova za pacijenta koji doji ili je trudan i za pacijenta koji nije trudan i ne doji*

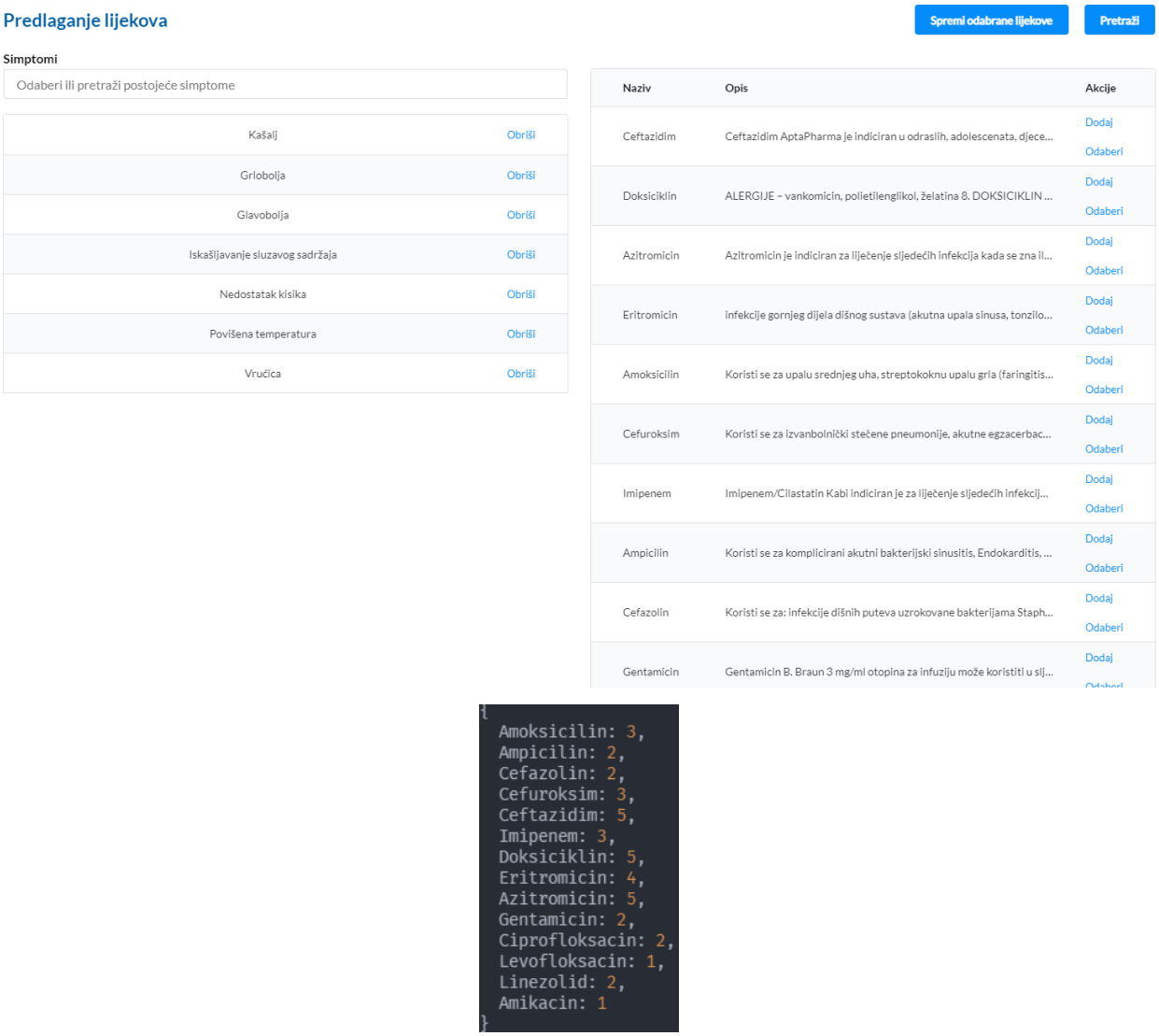
Kako bi se vidjelo cjelokupno filtriranje koje uključuje simptome, alergije i dojenje i trudnoću, potrebno je pacijentu ponovno postaviti vrijednost da doji ili je trudan te je potrebno postaviti alergije prije početka filtriranja. Pacijentu se postavljaju alergije na kukuruzni škrob, što bio trebalo isključiti Azitromicit, i natrijev citrat koji isključuje Amikacin iz prijedloga lijekova. S tako postavljenim podacima korisnik zatim za iste simptome kao na prethodnoj slici dobiva prijedlog lijekova kao na slici 5.6 gdje se može primijetiti kako na listi više ne postoje Azitromicin i Amikacin zbog odabranih alergija.



Sl. 5.5. *Predlaganje lijekova za pacijenta s odabranim alergijama, simptomima i dojenjem ili trudnoćom*

Još jedan primjer za provjeru ispravnost prijedloga lijekova s obzirom na parametre pacijenta je za bolest bronhitis. U bazi podataka uneseno je četiri različita lijeka koja služe za liječenje običnog bronhitisa i kroničnog bronhitisa. Prema slici 5.6, može se vidjeti dio liste predloženih lijekova u kojoj je predloženo 14 različitih lijekova za predane parametre. S obzirom na simptome i njihov broj, toliko lijekova je predloženo jer se barem jedan od simptoma nalazi u svakom od predloženih lijekova. Uneseni lijekovi koji se koriste za liječenje ili bronhitisa ili kroničnog bronhitisa su Cefuroksim, Doksiciklin, Eritromicin i Azitromicin.

U listi lijekova se nalaze i još mnogo drugih lijekova iz prethodno spomenutog razloga, no redoslijed lijekova ovisi o broju podudaranja predanih simptoma. U ovom slučaju Ceftazidim je prvi na listi jer se najviše simptoma podudara s predanim simptomima iako se ne koristi za liječenje bronhitisa. Razlog tome je što su simptomi identični za teže bolesti, ali su intenzivniji. Zbog toga Ceftazidim, koji se koristi za jače bolesti respiratornog sustava, u sebi sadržava i navedene simptome te odgovara načinu filtriranja. Nakon spomenutog lijeka može se vidjeti prijedlog i spomenutih lijekova koji se koriste za liječenje bronhitisa i kroničnog bronhitisa kao i drugi lijekovi koji se koriste za druge bolesti sa sličnim simptomima kao što je vidljivo na slici 5.6.



Sl. 5.6. *Predlaganje lijekova za pacijenta s odabranim simptomima za bolest bronhitis i broj podudaranja simptoma u lijekovima*

Ovim ručnim testiranjem potvrdila se ispravnost filtriranja lijekova na dijelu stranice za predlaganje lijekova. Takav način provjere se može provesti za razne druge bolesti i tipove pacijenata te će predloženi lijekovi uvijek ovisiti o točnosti i konzistentnosti unesenih podataka unutar lijekova. Pod time se misli da se koristi isključivo jedan naziv simptoma za određeni simptom. Primjer toga bio bi da se ne koristi uz grlobolju naziv „Bol u grlu“ kao simptom jer će se predlagati onda isključivo lijekovi koji u sebi sadrže simptom „Bol u grlu“ kada je taj simptom odabran, dok će lijekovi s grloboljom kao simptomom ostati izostavljeni iako se misli na isti simptom. Uz to, svaki od ovih lijekova se koriste za veliki broj bolesti, i svaka od tih bolesti ima svoje simptome te se u ovim primjerima lijekova možda ne nalaze svi simptomi za sve bolesti što također može utjecati na točnost predloženih lijekova i time se dodatno potvrđuje ovisnost o točnosti podataka.

U nastavku stranice su većinom prikazi liste koje podrazumijevaju korisnike, prošle pacijente te lijekove. Uz njih se nalaze i razni detaljni prikazi za pojedine podatke iz navedenih lista. Takve komponente i njihove funkcionalisti se najbolje provjeravaju pisanjem testova koji omogućuju provjeru postojećih funkcionalnosti kao i novih kada se uvedu promjene unutar komponenti. Prema tome, uvedeno je i automatizirano testiranje pisanjem testova za pojedine komponente gdje se komponente i funkcionalnosti unutar njih zasebno testiraju. Kao što je ranije spomenuto u idejnom riješenu aplikacije, za testiranje koristili su se Jest, Enzyme i React Testing Library alati za testiranje te Visual Studio Code za pisanje i pokretanje testova.

Prikaz jednog testa se može vidjeti na slici 5.7 gdje je definiran test i njegovi početni podaci bez kojih ne može komponente normalno funkcionirati unutar lažne lokalne baze stanja definirane *mockStore* funkcijom. Opis testa je vidljiv unutar *describe* funkcije kao tekstualni parametar, a unutar parametra funkcije se nalaze definirani:

- Testovi unutar *it* funkcije koja prima nazivi i funkciju koju će izvoditi tijekom teksta
- Varijabla unutar koje će biti spremljena komponenta koja se testira
- Sve funkcije koje su potrebne za funkcioniranje cijele komponente i njezinih funkcionalnosti.
- Dodatno definirane vrijednosti po potrebi

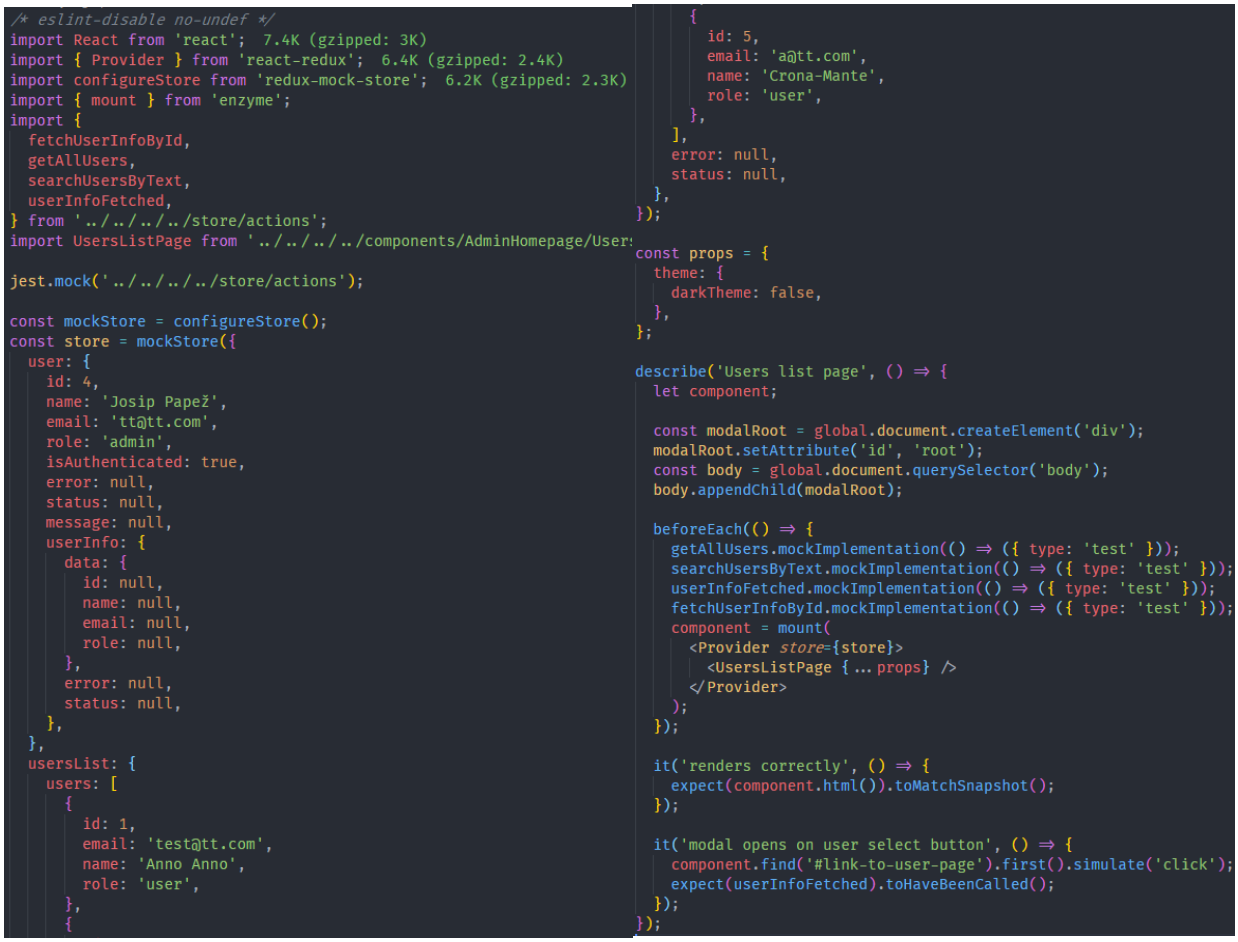
Iz spomenutog se može zaključiti da je na slici 5.7 definiran test s nazivom „Users list page“ za komponentu koja predstavlja cijelu stranicu s listom korisnika. Ona uključuje prikaz liste, pretraživanje liste te detaljan prikaz korisnika.

Unutar *store* varijable su spremljeni podaci koji su potrebni za pristup svim dijelovima na stranici s listom korisnika. Podaci koji su potrebni za ispravan rad komponente su podaci o korisniku koji pristupa stranici i popis svih korisnika koji će se nalaziti u listi. Dodatni podaci koji se predaju komponenti iz neke druge komponente, a mogu i ne moraju utjecati na rad komponente, se nalaze unutar *props* varijable.

Unutar prikazane *describe* funkcije se nalaze, kao što je spomenuto, testovi, postavljene sve varijable koje su potrebne i definirane funkcije koje će oponašati one funkcije koje će biti izvedene pri testiranju raznih dijelova stranice.

Primjer tome bi bio pritisak na „Odaberi“ tipku u listi korisnika koja otvara detaljan prikaz tog korisnika. Prilikom pritiska i pojavljivanja detaljnog prikaza izvršava se *fetchUserInfoById* zahtjev koji dohvaća detalje o korisniku kojem se želi pristupiti. Ta funkcija ne treba izvoditi cijeli poziv prema poslužitelju već treba biti pozvana kako bi se znalo da funkcionira komponenta. Zbog toga

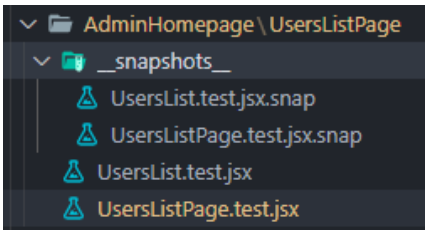
se zamjenjuje pravi poziv s pozivom koji vraća objekt s varijablom *type* i vrijednosti „test“ pozivom *mockImplementation* funkcije.



```
/* eslint-disable no-undef */
import React from 'react';  7.4K (gzipped: 3K)
import { Provider } from 'react-redux';  6.4K (gzipped: 2.4K)
import configureStore from 'redux-mock-store';  6.2K (gzipped: 2.3K)
import { mount } from 'enzyme';
import {
  fetchUserInfoById,
  getAllUsers,
  searchUsersByText,
  userInfoFetched,
} from '../../../../store/actions';
import UsersListPage from '../../../../components/AdminHomepage/Users

jest.mock('../../../../store/actions');

const mockStore = configureStore();
const store = mockStore({
  user: {
    id: 4,
    name: 'Josip Papež',
    email: 'tt@tt.com',
    role: 'admin',
    isAuthenticated: true,
    error: null,
    status: null,
    message: null,
    userInfo: {
      data: {
        id: null,
        name: null,
        email: null,
        role: null,
      },
      error: null,
      status: null,
    },
  },
  usersList: {
    users: [
      {
        id: 1,
        email: 'test@tt.com',
        name: 'Anno Anno',
        role: 'user',
      },
      {
        id: 5,
        email: 'a@tt.com',
        name: 'Crona-Mante',
        role: 'user',
      },
    ],
    error: null,
    status: null,
  },
});

const props = {
  theme: {
    darkTheme: false,
  },
};

describe('Users list page', () => {
  let component;

  const modalRoot = global.document.createElement('div');
  modalRoot.setAttribute('id', 'root');
  const body = global.document.querySelector('body');
  body.appendChild(modalRoot);

  beforeEach(() => {
    getAllUsers.mockImplementation(() => ({ type: 'test' }));
    searchUsersByText.mockImplementation(() => ({ type: 'test' }));
    userInfoFetched.mockImplementation(() => ({ type: 'test' }));
    fetchUserInfoById.mockImplementation(() => ({ type: 'test' }));
    component = mount(
      <Provider store={store}>
        <UsersListPage {...props} />
      </Provider>
    );
  });

  it('renders correctly', () => {
    expect(component.html()).toMatchSnapshot();
  });

  it('modal opens on user select button', () => {
    component.find('#link-to-user-page').first().simulate('click');
    expect(userInfoFetched).toHaveBeenCalled();
  });
});
```

Sl. 5.7. *Prikaz testne datoteke za stranicu s listom korisnika*

Funkcija *mount*, iz Enzyme alata za testiranje, postavlja i prikazuje dokumentu postavljenu komponentu te pruža omot za testiranje oko komponente koji se može koristiti u testovima za provjeru funkcionalnosti i korisničkog sučelja. Prvi test koji je definiran je „renders correctly“ test koji provjerava prikazuje li se komponenta ispravno tako da snimi snimak testa komponente korisničkog sučelja te ga uspoređuje s referentom snimkom koja je pohranjena uz test te se takav način testiranja naziva Snapshot testiranje [28]. U slučaju da se snimke ne podudaraju, test neće proći jer se dogodila neočekivana ili je potrebno referentnu snimku ažurirati na novu verziju komponente korisničkog sučelja. Na slici 5.8 se može vidjeti prikaz datoteke testa te prikaz direktorija „snapshots“ koji se nalazi uz test te sadržava snimke testova.



Sl. 5.8. *Prikaz datoteke testa i snimke testa*

Snimka testa se provjerava pozivom *expect* funkcije koja u sebi prima vrijednost, a u ovom slučaju je to vrijednost cijelog HTML-a komponente koja se testira. Zatim, postavljanjem funkcije podudaranja prema kojem će se pregledavati vrijednost unutar *expect* varijable, izvršava se testiranje. Kako bi se testiralo podudaranja snimke testa, poziva se funkcija podudaranja *toMatchSnapshot* koja provjerava podudaranje snimki. Osim spomenute funkcije podudaranja Jest u sebi sadrži i mnoge druge funkcije podudaranja poput uspoređivanja vrijednosti varijabli, poziva li se određena funkcija, koliko puta je pozvana određena funkcija, što vraća funkcija, provjere je li definirana varijabla ili funkcija itd.

U drugom testu koji se izvršava provjerava se poziva li se funkcija koja postavlja vrijednost odabranog korisnika u lokalnu bazu stanja. Izvršava se tako da se na komponenti pronađe tipka s identifikacijskim tekstom „link-to-user-page“ te se poziva simulacija pritiska na tipku. Pritiskom na tipku trebala bi se pozvati spomenuta funkcija koja je definirana unutar *expect* funkcije.

Ovakve vrste testova se koriste i u drugim testnim datotekama te se, ovisno o tome što se nalazi na stranici, provjeravaju različite funkcionalnosti i vrijednosti. Primjer tome bi bilo provjeravanje mijenja li se vrijednost unosa prilikom pritiska tipke na tipkovnici za unos e-pošte na stranici za prijavu i registraciju. Na slici 5.9 se može vidjeti primjena takve provjere gdje se prvo simulira promjena na unosu imena, e-pošte i lozinke te se zatim provjeravaju podudaraju li se unesene vrijednosti s očekivanima. Uz to, može se provjeravati cijeli proces prijave, od pritiska na tipku za prijavu pa sve do završetka prijave.

```
it('name, email and password content changes on input', () => {
  component
    .find('.login__button-container__register-btn')
    .simulate('click');

  component.find('.content-container__name').simulate('change', {
    target: { value: 'test' },
  });
  component.find('.content-container__email').simulate('change', {
    target: { value: 'test' },
  });
  component.find('.content-container__password').simulate('change', {
    target: { value: 'test' },
  });
  component.find('.password-icon').simulate('click');
  expect(component.find('.content-container__name').props().value).toBe(
    'test'
  );
  expect(component.find('.content-container__email').props().value).toBe(
    'test'
  );
  expect(component.find('.content-container__password').props().value).toBe(
    'test'
  );
});
it('it should dispatch login user when login form button is clicked', () => {
  loginUser.mockImplementation((email, password) => ({
    type: 'TEST',
    payload: { email, password },
  }));
  loginError.mockImplementation(error => ({
    type: 'TEST',
    payload: error,
  }));
  component.find('.login__button-container__login-btn').simulate('click');
  component.find('.content-container__email').simulate('change', {
    target: { value: 'test' },
  });
  component.find('.content-container__password').simulate('change', {
    target: { value: 'test' },
  });
  component.find('form').simulate('submit');
  expect(loginUser).toBeCalled();
  component.unmount();
});
```

Sl. 5.8. *Test provjere unosa za ime, e-poštu i lozinku te provjere procesa prijave*

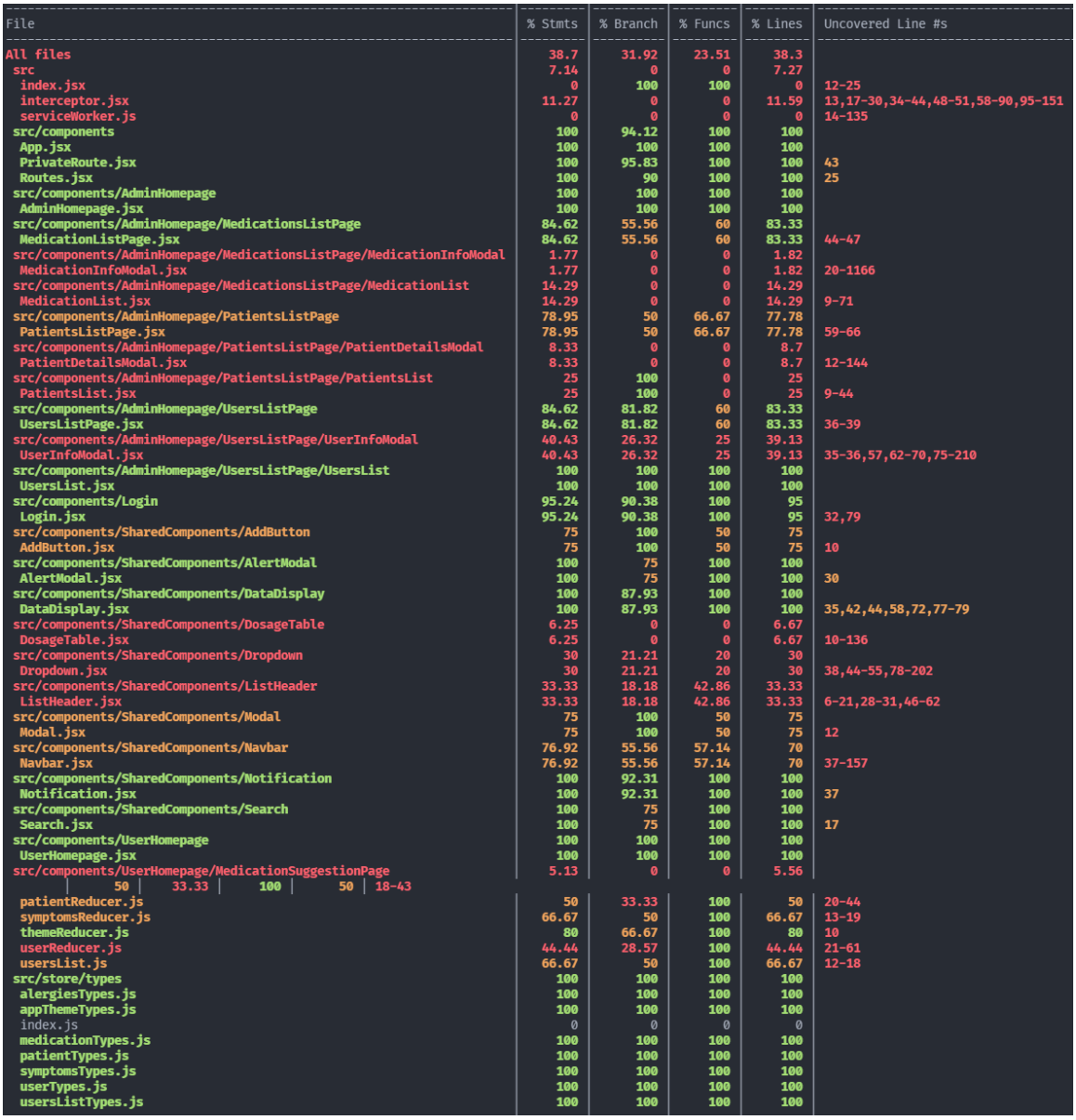
Nakon pisanja testova i definiranja svega što je potrebno za uspješno izvršavanje testova, unutar Visual Studio Code terminala pozivom skripte „npm run test“ pokreće se izvršavanje testova. U terminalu se dobivaju povratne informacije o napisanim testovima. Na slici 5.9 se može vidjeti izvršavanje testova, informacija je li prošao test ili ne te koliko je bilo potrebno za izvršenje nekog testa u slučaju da traje neko duže vrijeme.

```
PS C:\Users\Josip\Desktop\illness-recommend-fe> npm run test

> illness-recommend-fe@0.1.0 test C:\Users\Josip\Desktop\illness-recommend-fe
> set CI=true&&react-scripts test --updateSnapshot --reporters=default --coverage

 PASS  src/tests/components/SharedComponents/AlertModal.test.jsx
 PASS  src/tests/components/SharedComponents/DataDisplay.test.jsx
 PASS  src/tests/components/SharedComponents/Notification.test.jsx
 PASS  src/tests/components/SharedComponents/Search.test.jsx
 PASS  src/tests/components/SharedComponents/Navbar.test.jsx
 PASS  src/tests/components/AdminHomepage/UsersListPage/UsersList.test.jsx
 PASS  src/tests/components/AdminHomepage/UsersListPage/UsersListPage.test.jsx
 PASS  src/tests/components/App.test.jsx
 PASS  src/tests/components/PrivateRoute.test.jsx
 PASS  src/tests/components/Login/Login.test.jsx
 PASS  src/tests/components/Routes.test.jsx (5.09 s)

Test Suites: 11 passed, 11 total
Tests:       33 passed, 33 total
Snapshots:   21 passed, 21 total
Time:        8.463 s
```

Sl. 5.9. *Izvršavanje testova i statistika izvršavanja*

Uz informacije o prolaznosti testa, unutar terminala zbog zastavice „--coverage“ ispisuje se u tablici i pokrivenost koda testovima. U tablici se prikazuju sve datoteke koje se mogu testirati, postotak pokrivenosti tvrdnji, grana, funkcija i linija u testovima te linije koje nisu testirane.

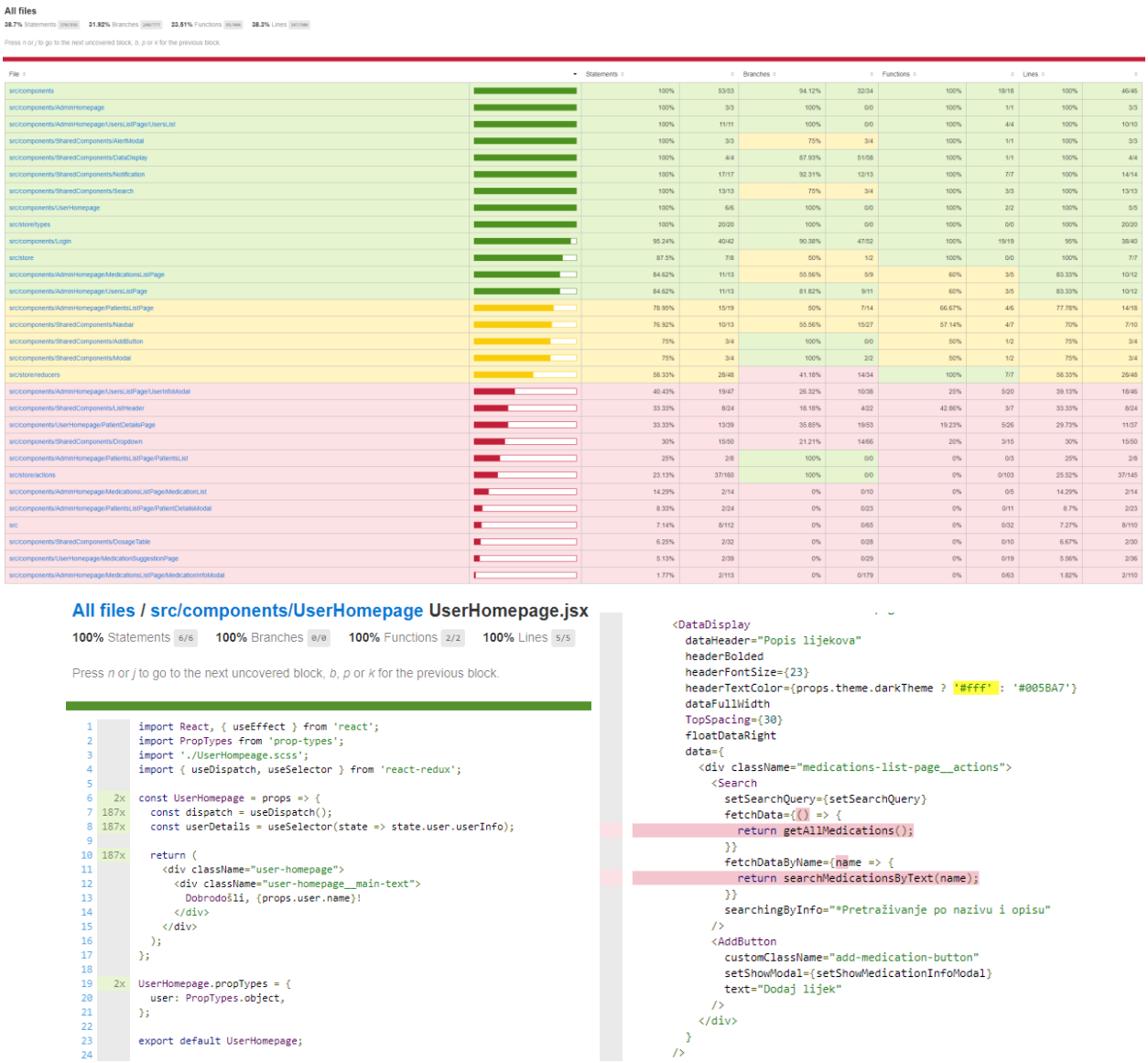
Primjer takve tablice je dan na slici 5.10. gdje se različitim bojama označavaju pojedine vrijednosti u tablici s obzirom na pokrivenost testovima tog dijela koda.



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	38.7	31.92	23.51	38.3	
src	7.14	0	0	7.27	
index.jsx	0	100	100	0	12-25
interceptor.jsx	11.27	0	0	11.59	13,17-30,34-44,48-51,58-90,95-151
serviceWorker.js	0	0	0	0	14-135
src/components	100	94.12	100	100	
App.jsx	100	100	100	100	
PrivateRoute.jsx	100	95.83	100	100	43
Routes.jsx	100	90	100	100	25
src/components/AdminHomepage	100	100	100	100	
AdminHomepage.jsx	100	100	100	100	
src/components/AdminHomepage/MedicationsListPage	84.62	55.56	60	83.33	
MedicationListPage.jsx	84.62	55.56	60	83.33	44-47
src/components/AdminHomepage/MedicationsListPage/MedicationInfoModal	1.77	0	0	1.82	
MedicationInfoModal.jsx	1.77	0	0	1.82	20-1166
src/components/AdminHomepage/MedicationsListPage/MedicationList	14.29	0	0	14.29	
MedicationList.jsx	14.29	0	0	14.29	9-71
src/components/AdminHomepage/PatientsListPage	78.95	50	66.67	77.78	
PatientsListPage.jsx	78.95	50	66.67	77.78	59-66
src/components/AdminHomepage/PatientsListPage/PatientDetailsModal	8.33	0	0	8.7	
PatientDetailsModal.jsx	8.33	0	0	8.7	12-144
src/components/AdminHomepage/PatientsListPage/PatientsList	25	100	0	25	
PatientsList.jsx	25	100	0	25	9-44
src/components/AdminHomepage/UsersListPage	84.62	81.82	60	83.33	
UsersListPage.jsx	84.62	81.82	60	83.33	36-39
src/components/AdminHomepage/UsersListPage/UserInfoModal	40.43	26.32	25	39.13	
UserInfoModal.jsx	40.43	26.32	25	39.13	35-36,57,62-70,75-210
src/components/AdminHomepage/UsersListPage/UsersList	100	100	100	100	
UsersList.jsx	100	100	100	100	
src/components/Login	95.24	90.38	100	95	
Login.jsx	95.24	90.38	100	95	32,79
src/components/SharedComponents/AddButton	75	100	50	75	
AddButton.jsx	75	100	50	75	10
src/components/SharedComponents/AlertModal	100	75	100	100	
AlertModal.jsx	100	75	100	100	30
src/components/SharedComponents/DataDisplay	100	87.93	100	100	
DataDisplay.jsx	100	87.93	100	100	35,42,44,58,72,77-79
src/components/SharedComponents/DosageTable	6.25	0	0	6.67	
DosageTable.jsx	6.25	0	0	6.67	10-136
src/components/SharedComponents/Dropdown	30	21.21	20	30	
Dropdown.jsx	30	21.21	20	30	38,44-55,78-202
src/components/SharedComponents/ListHeader	33.33	18.18	42.86	33.33	
ListHeader.jsx	33.33	18.18	42.86	33.33	6-21,28-31,46-62
src/components/SharedComponents/Modal	75	100	50	75	
Modal.jsx	75	100	50	75	12
src/components/SharedComponents/Navbar	76.92	55.56	57.14	70	
Navbar.jsx	76.92	55.56	57.14	70	37-157
src/components/SharedComponents/Notification	100	92.31	100	100	
Notification.jsx	100	92.31	100	100	37
src/components/SharedComponents/Search	100	75	100	100	
Search.jsx	100	75	100	100	17
src/components/UserHomepage	100	100	100	100	
UserHomepage.jsx	100	100	100	100	
src/components/UserHomepage/MedicationSuggestionPage	5.13	0	0	5.56	
50 | 33.33 | 100 | 50 | 18-43					
patientReducer.js	50	33.33	100	50	20-44
symptomsReducer.js	66.67	50	100	66.67	13-19
themeReducer.js	80	66.67	100	80	10
userReducer.js	44.44	28.57	100	44.44	21-61
usersList.js	66.67	50	100	66.67	12-18
src/store/types	100	100	100	100	
alergiesTypes.js	100	100	100	100	
appThemeTypes.js	100	100	100	100	
index.js	0	0	0	0	
medicationTypes.js	100	100	100	100	
patientTypes.js	100	100	100	100	
symptomsTypes.js	100	100	100	100	
userTypes.js	100	100	100	100	
usersListTypes.js	100	100	100	100	

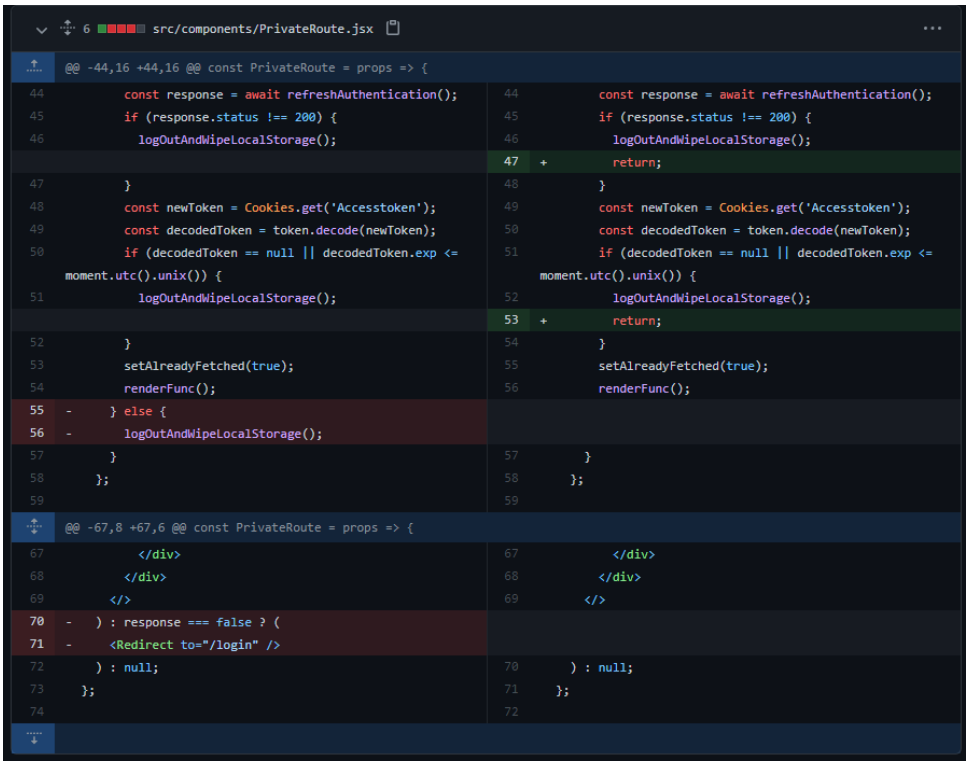
Sl. 5.10. *Pokrivenost koda testovima*

Uz takav prikaz u terminalu, korišten je i alat istanbul koji služi za bolji prikaz pokrivenosti koda testovima te se pomoću njega mogu i pregledavati zasebne datoteke te pokrivenosti unutar njih u obliku HTML izvještaja. Na slici 5.11 se nalazi primjer prikaza iste tablice, ali u prikazu u pregledniku na lokalnom poslužitelju koji je pokrenut preko proširenja u Visual Studio Code-u. Pritiskom na naziv datoteka otvara se prikaz datoteke gdje su označene linije koje su testirane i koliko puta su pozivane, linije koje nisu testirane te grane koje nisu pokrivene. Ovaj alat omogućuje lakše praćenje pokrivenosti linija i testiranja u smislu što je potrebno testirati unutar svake datoteke.



Sl. 5.11. *Pokrivenost koda testovima unutar istanbul izvještaja u pregledniku*

Prema tome, na konkretnom primjeru *PrivateRoute* datoteke, ovaj alat je omogućio pronalazak ne korištenih dijelova koda kao i propust u prekidanju izvršenja funkcije kada je potrebno odjaviti korisnika i obrisati lokalnu memoriju. Nakon provođenja svih testova unutar datoteke i pokrivanja svih slučajeva koji se mogu dogoditi, preostale grane koje su označene crvenom bojom na slici 5.12 se mogu ukloniti jer se u niti jednom testu nisu izvršile, a tijekom testiranja se moglo primijetiti da se funkcija nastavi izvoditi iako se korisnika odjavi te je dodana *return* izjava kako bi se izvođenje prekinulo.



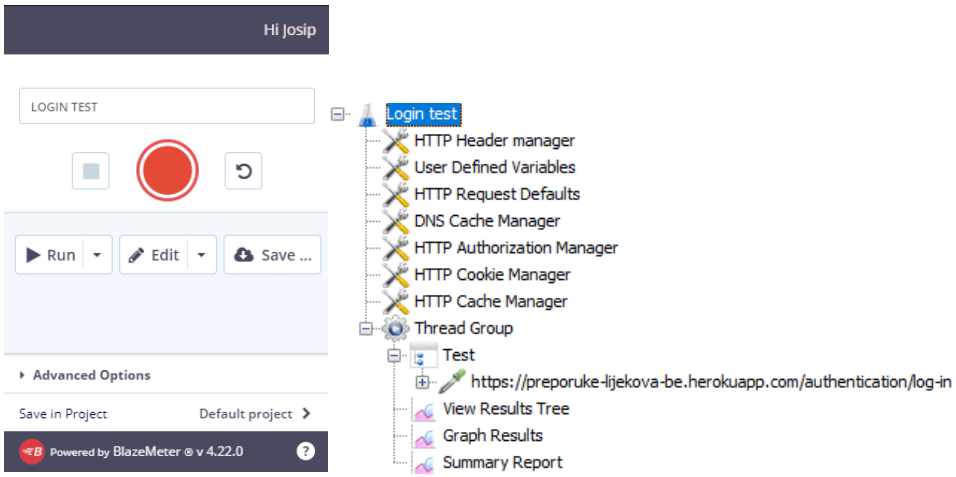
Sl. 5.12. *Izmjena PrivateRoute datoteke nakon testiranja*

Nakon završetka testiranja, na više mjesta u kodu su se prepravile manje greške te je maknut redundantan kod koji se nije koristio ili je bio suvišan. Time se dokazuje bit testiranja i pronalaska grešaka koje mogu utjecati na čistoću koda, izvršenje funkcionalnosti te pravilan prikaz korisničkog sučelja kako bi se korisniku pružilo najbolje moguće korisničko iskustvo. Isto tako, potrebno je shvatiti da je testiranje vrlo važno provoditi nakon izmjena dijelova koda kako bi osigurali rad već implementiranih funkcionalnosti kao i novo dodanih.

5.3. Stres test

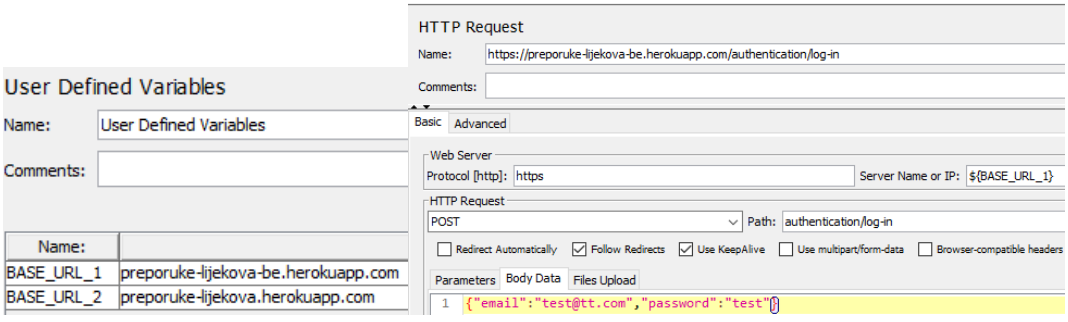
Kao što je ranije spomenuto, testiranje stresa se odvija tek nakon postavljanja web sustava na udaljenog poslužitelja. Klijentska i poslužiteljska strana je postavljena na Heroku poslužitelja jer pruža PostreSQL dodatak za poslužiteljsku stranu te je jednostavno povezati poslužiteljsku stranu s novom bazom podataka na udaljenom poslužitelju. Osim toga, sa svakom novom izmjenom i slanjem izmjena na GitHub repozitorij, klijentska i poslužiteljska strana se automatski ponovno postavlja kako bi web sustav uvijek uključivao najnovije promjene. Također, baza podataka je postavljena na udaljenog poslužitelja u Europi i ograničena je brojem povezivanja i redaka besplatnim planom pa se stres testiranjem može provjeriti najveći broj korisnika koji se pokušava spojiti istovremeno na web sustav. U besplatnom planu baza ima ograničenje na 20 istovremenih poziva, tj. zahtjeva prema bazi, što će utjecati na rezultate testa.

Na slici 5.13 se može vidjeti postavljena struktura stres testa u kojoj se nalazi naziv testa, unutar testa grupa niti koja će izvršavati pozive prema web sustavu te zahtjevi koje će izvršavati. Uz njih, potrebno je i vizualizirati podatke te se koristi graf, stablo i prikaz izvješća s rezultatima. Test se generirao korištenjem BlazeMeter alata koji se može preuzeti s Chrome web trgovine i koji omogućuje snimanje zahtjeva koji se trebaju izvršiti na stranici te svih potrebnih podataka koji se trebaju slati. BlazeMeter alat omogućuje spremanje snimljenog testa u obliku kojeg JMeter može čitati sa svim potrebnim elementima za izvršenje testa.



Sl. 5.13. *BlazeMeter alat za snimanje testova i struktura stres testa prijave*

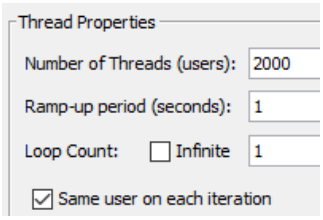
Unutar HTTP Request Defaults elementa definiraju se generalne postavke potrebne za spajanje na web sustav koje će koristiti ostali HTTP Request elementi. Također, iste postavke se mogu postaviti pojedinačno unutar HTTP Request elemenata. U ovom slučaju potrebno je postaviti protokol koji je *https* te IP adresu, tj. poveznicu na stranicu. Na slici 5.14 moguće je vidjeti postavke za HTTP Request element za spajanje na web sustav. U slučaju da je potrebno pristupiti nekoj drugoj stranici, unutar elementa HTTP Request elementa može se definirati zasebna IP adresa sa zasebnom rutom, dok u ovom slučaju se želi pristupiti „authentication/login“ ruti na poslužiteljskoj strani web sustava gdje se predaju parametri e-pošta i lozinka za prijavu.



Sl. 5.14. *Prikaz postavki unutar elementa za spajanje na web sustav*

Nakon postavljanja podataka vezanih za web sustav koji će biti testiran, unutar Thread Group elementa se postavljaju parametri vezani za broj korisnika koji će pristupati sustavu. Prilikom testiranja, postavljanjem broja korisnika na 2000, broj ponavljanja na 1 te vrijeme u sekundama koje je potrebno da se svi korisnici pokušaju spojiti na web sustav je 1.

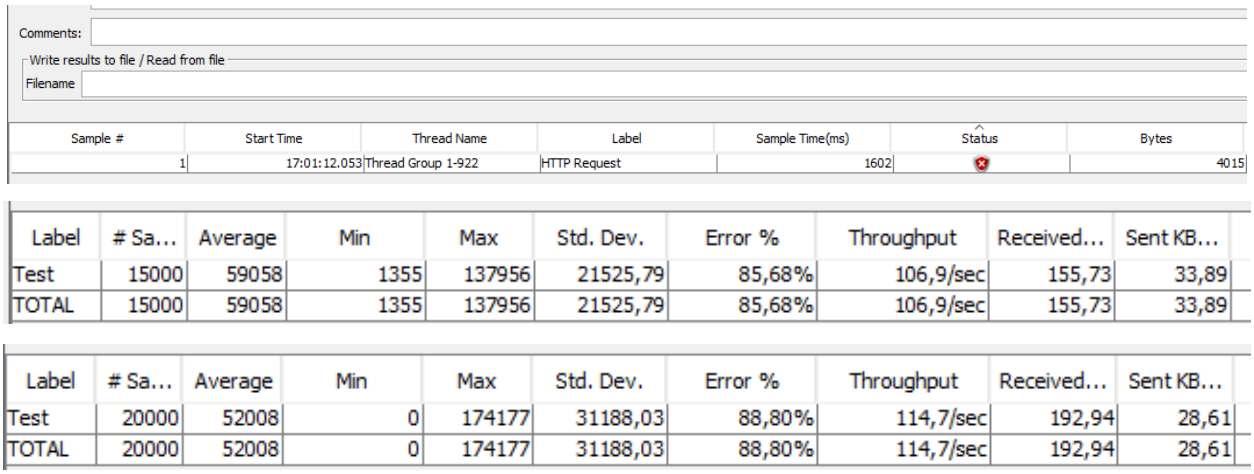
S tako postavljenim podacima se može izračunati da se 2000 korisnika po sekundi spaja na web sustav te se to ponavlja 1 put. Na slici 5.15 se mogu vidjeti navedene postavke te se za njih prilikom testiranja ne događaju greške.



Sl. 5.15. *Postavke za broj korisnika koji će pristupati web sustavu*

Podizanjem isključivo broja korisnika s 2000 na 2200 dogodila se jedna greška prilikom prijave. Također, podizanjem na 2500 korisnika dogodilo se 347 grešaka, za 3500 korisnika nastalo je 1429 grešaka, za 4500 korisnika 2402 greške itd.

Postavljanjem broja korisnika na 15000 dobiva se rezultat kao na slici 5.16 gdje je postotak pogrešaka narastao na 85,68%. Istim postupkom, promjenom na 20000 korisnika, greška je narasla na 88.80%


Comments:

Write results to file / Read from file

Filename

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes
1	17:01:12.053	Thread Group 1-922	HTTP Request	1602		4015

Label	# Sa...	Average	Min	Max	Std. Dev.	Error %	Throughput	Received...	Sent KB...
Test	15000	59058	1355	137956	21525,79	85,68%	106,9/sec	155,73	33,89
TOTAL	15000	59058	1355	137956	21525,79	85,68%	106,9/sec	155,73	33,89

Label	# Sa...	Average	Min	Max	Std. Dev.	Error %	Throughput	Received...	Sent KB...
Test	20000	52008	0	174177	31188,03	88,80%	114,7/sec	192,94	28,61
TOTAL	20000	52008	0	174177	31188,03	88,80%	114,7/sec	192,94	28,61

Sl. 5.16. *Prikaz greške u tablici i statistike nakon izvođenja testa*

Prema prikazanim podacima može se zaključiti da web sustav može izdržati do 2200 istovremenih korisnika prije pojavljivanja greške za prijavu u sustav.

Rezultati s većim broj korisnika također ovise i o brzini pokretanja niti te izvršenju zahtjeva pomoću JMeter-a. Svaki korisnik može pričekati na izvršenje zahtjeva te se ne može doći do 100% grešaka jer će se uvijek moći izvoditi 20 istovremenih zahtjeva sve dok je poslužiteljska strana aktivna. Prema tome, korisnici mogu doći na red te će se njihov zahtjev izvršiti.

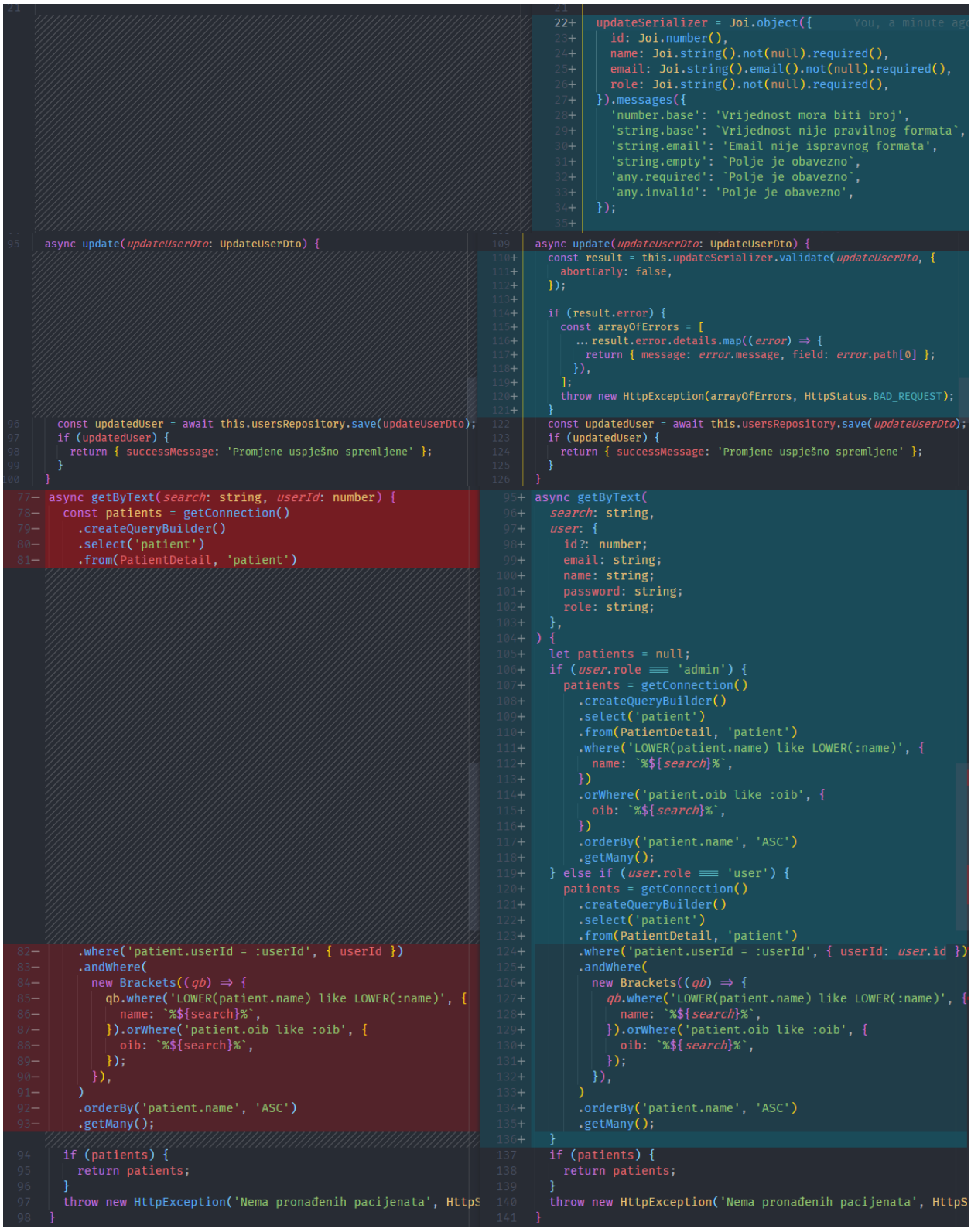
Osim toga, povećanjem broja obrade istovremenih zahtjeva promjenom plana za bazu ili prelaskom na nekog drugog poslužitelja bi trebalo smanjiti i broj grešaka za veći broj korisnika koji se istovremeno pokušava prijaviti ili izvršiti neki zahtjev prema poslužiteljskoj strani web sustava.

5.4. Testiranje API-ja

Testiranje API-ja se većinski odradilo pisanjem automatiziranih testova poput onih za klijentsku stranu. Testovi su pisani na identičan način jer se koristi Jest i na poslužiteljskoj strani kao i na klijentskoj strani, jedina razlika je što se uz njega koristi i SuperTest alat koji služi za testiranje poziva na krajnje točke i provjeru odziva kao i statusa zahtjeva.

Neke funkcionalnosti, poput dohvaćanja lijekova, pretraživanja korisnika, lijekova i pacijenata te uređivanje i spremanje novih podataka su testirani ručno na klijentskoj strani zbog lakše i brže provjere odziva i primljenih podataka s poslužiteljske strane. Ručnim testiranjem su popravljene manje greške kod spremanja podataka te provjere podataka prije spremanja kod korisnika, lijekova i pacijenata. Primjer ispravljene greške, tj. dodane provjere, je vidljiv na slici 5.17 gdje se prilikom ažuriranja podataka također provjeravaju predani podaci kako bi se izbjeglo spremanje korisnika s praznim imenima i adresama e-pošte.

Osim izmjena kod provjere podataka, dodane su i izmjene prilikom pretraživanja pacijenata po OIB-u i imenu i prezimenu. Pretraživanje funkcionira tako da se kod administratora dohvaćaju svi pacijenti koji se zatim filtriraju i vraćaju u slučaju da unutar OIB-a ili imena i prezimena sadržavaju unos prema kojem se pretražuje. Za razliku od njih, korisnicima, tj. liječnicima, se prvo dohvaćaju svi pacijenti koji su vezani za njih putem identifikacijskog broja što znači da su ih oni unijeli. Takvi pacijenti se također opet na isti način filtriraju kao kod administratora. Prema spomenutom, može se vidjeti izmjena na slici 5.17 gdje se dodaje grananje u slučaju da administrator i u slučaju da korisnik pretražuje pacijente. Bez te izmjene, korisnik bi mogao pretraživati pacijente, dok administrator ne jer administrator ne unosi pacijente te se ne mogu pronaći pacijenti za njegov identifikacijski broj.



Sl. 5.17. *Prikaz izmjena nakon ručnog testiranja ažuriranja korisnika i pretraživanja pacijenata*

Kao što je spomenuto kod ručnog testiranja klijentske strane, ručnim testiranjem su se mogle uhvatiti veće greške te greške koje u ovom slučaju nisu utjecale na funkcioniranje sustava, njegovih glavnih funkcionalnosti te stabilnosti.

Ostale funkcionalnosti i odzivi na zahtjeve su testirani pomoću napisanih testova. Testovi se pišu na identičan način kao što je spomenuto na testiranju klijentske strane kao što je vidljivo na slici 5.18. gdje se može vidjeti struktura testa. Razlika između pisanja testova na klijentskoj i poslužiteljskoj strani je način na koji se definiraju i oponašaju moduli u usporedbi s komponentama na klijentskoj strani. Kod oponašanja modula i funkcija iz modula važno je pripaziti na koji način se definiraju i da se nalazi u definiciji modula sve što je potrebno za radi. Najsličniji primjer tome je definiranje početnih stanja i varijabli te funkcija u testovima na klijentskoj strani.

```
describe('The AuthenticationController', () => {
  let app: INestApplication;
  let userData: User;
  let bcryptCompare: jest.Mock;
  beforeEach(async () => {
    userData = {
      ...mockedUser,
    };
    bcryptCompare = jest.fn().mockReturnValue(true);
    (bcrypt.compare as jest.Mock) = bcryptCompare;
    const usersRepository = {
      create: jest.fn().mockResolvedValue(userData),
      save: jest.fn().mockReturnValue(Promise.resolve())
      findOne: jest.fn().mockResolvedValue(userData),
      getByEmail: jest.fn().mockResolvedValue(userData),
    };

    const module = await Test.createTestingModule({
      controllers: [AuthenticationController],
      providers: [
        UsersService,
        AuthenticationService,
        JwtRefreshTokenStrategy,
        LocalStrategy,
        {
          provide: ConfigService,
          useValue: mockedConfigService,
        },
        {
          provide: JwtService,
          useValue: mockedJwtService,
        },
        {
          provide: getRepositoryToken(User),
          useValue: usersRepository,
        },
      ],
    }).compile();
    app = module.createNestApplication();
    await app.init();
  });
  describe('when registering', () => {
    describe('and using valid data', () => {
      it('should respond with the data of the user without the password', () => {
        const expectedData = {
          ...userData,
        };
        delete expectedData.password;
        return request(app.getHttpServer())
          .post('/authentication/register')
          .send({
            email: mockedUser.email,
            name: mockedUser.name,
            password: 'strongPassword',
          })
          .expect(201)
          .expect(expectedData);
      });
    });
    describe('and using invalid data', () => {
      it('should throw an error', () => {
        return request(app.getHttpServer())
          .post('/authentication/register')
          .send({
            name: mockedUser.name,
          })
          .expect(400);
      });
    });
  });
  describe('when login', () => {
    describe('and using valid data', () => {
      beforeEach(() => {
        bcryptCompare.mockReturnValue(true);
      });
      it('should respond with the data of the user without the password', () => {
        const expectedData = {
```

Sl. 5.18. *Primjer strukture testa kontrolera za ovjeru korisnika*

Pokretanje testova je isto kao i kod klijentske strane pozivom „npm run test“ komande unutar terminala koja započinje izvršavanje testova. Nakon izvršenja testova, unutar terminala se ispisuju uspješno i neuspješno izvršeni testovi te pokrivenost testovima, koja se može vidjeti na slici 5.19, sa svim stupcima kao i na klijentskoj strani. Osim terminala, izvješće se generira nakon svakog pokretanja testova i u HTML formatu korištenjem istanbul alata te se može pregledati pokretanjem lokalnog poslužitelja korištenjem *Live Server* proširenja unutar Visual Studio Code-a.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	58.03	15.25	33.07	55.36	
src	0	0	0	0	
main.ts	0	0	0	0	1-11
src/alergies	100	100	100	100	
alergies.controller.ts	100	100	100	100	
alergies.service.ts	100	100	100	100	
src/alergies/dto	0	100	100	0	
create-alergy.dto.ts	0	100	100	0	1
update-alergy.dto.ts	0	100	100	0	1-4
src/alergies/entities	80	100	0	85.71	
alergy.entity.ts	80	100	0	85.71	12
src/authentication	89.63	25	81.82	88.6	
Roles.Guard.ts	38.89	0	0	33.33	9-29
Roles.decorator.ts	100	100	100	100	
authentication.controller.ts	100	100	100	100	
authentication.service.ts	92.5	40	100	92.11	54-60
jwt-access-token.strategy.ts	100	100	100	100	
jwt-authentication.guard.ts	100	100	100	100	
jwt-refresh-token.strategy.ts	100	100	100	100	
jwt-refresh.guard.ts	100	100	100	100	
local.strategy.ts	100	100	100	100	
localAuthentication.guard.ts	100	100	100	100	
src/authentication/dto	50	100	100	50	
logIn.dto.ts	0	100	100	0	1-6
register.dto.ts	100	100	100	100	
src/authentication/tests	100	100	100	100	
user.mock.ts	100	100	100	100	
src/database	100	100	100	100	
postgresErrorCode.enum.ts	100	100	100	100	
src/medications	20.45	0	0	18.11	
medications.controller.ts	66.67	100	0	63.64	22,27,33,40,46,52,59,66
medications.service.ts	10.19	0	0	8.57	15-290
src/medications/dto	25	100	100	25	
create-medication.dto.ts	100	100	100	100	
update-medication.dto.ts	0	100	100	0	1-4
src/medications/entities	80	100	0	87.5	
medication.entity.ts	80	100	0	87.5	29,46
src/patients-details	59.26	27.27	35	57.14	
patient-details.controller.ts	54.84	0	0	51.72	24,29,36,43,49,55-59,65-71,81
patient-details.service.ts	62	33.33	58.33	60.42	80,105-144,173-188
src/patients-details/dto	100	100	100	100	
create-patient-detail.dto.ts	100	100	100	100	
update-patient-detail.dto.ts	100	100	100	100	
src/patients-details/entities	78.26	100	0	84.21	
patient-detail.entity.ts	78.26	100	0	84.21	19,25,51
src/symptom	100	100	100	100	
symptom.controller.ts	100	100	100	100	
symptom.service.ts	100	100	100	100	
src/symptom/dto	0	100	100	0	
create-symptom.dto.ts	0	100	100	0	1
update-symptom.dto.ts	0	100	100	0	1-4
src/symptom/entities	80	100	0	85.71	
symptom.entity.ts	80	100	0	85.71	12
src/users	30	16.67	23.53	28.79	
users.controller.ts	0	0	0	0	1-61
users.service.ts	44.68	18.75	36.36	42.22	37-75,97,110-144
src/users/dto	0	100	100	0	
createUser.dto.ts	0	100	100	0	1-7
updateUser.dto.ts	0	100	100	0	1-4
src/users/entities	100	100	100	100	
user.entity.ts	100	100	100	100	
src/utils/mocks	80	75	66.67	80	
config.service.ts	85.71	75	100	85.71	11
jwt.service.ts	100	100	100	100	
repositoryMockFactory.service.ts	60	100	50	60	10-11

All files

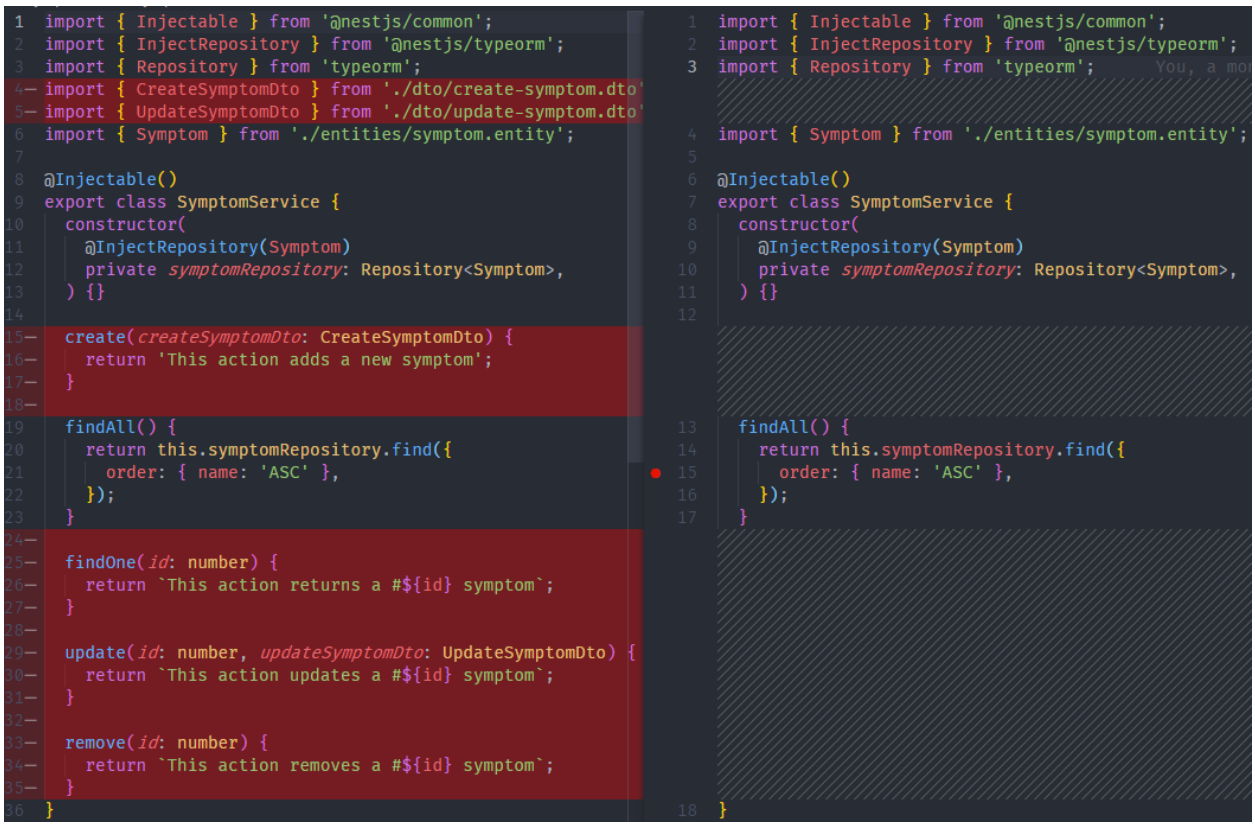
58.03% Statements 336/579 **15.25%** Branches 18/118 **33.07%** Functions 42/127 **55.36%** Lines 289/522

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File	Statements		Branches	
src/alergies	100%	18/18	100%	0/0
src/authentication/tests	100%	2/2	100%	0/0
src/database	100%	3/3	100%	2/2
src/patients-details/dto	100%	4/4	100%	0/0
src/symptom	100%	18/18	100%	0/0
src/users/entities	100%	9/9	100%	0/0
src/authentication	89.63%	121/135	25%	4/16
src/alergies/entities	80%	8/10	100%	0/0
src/medications/entities	80%	16/20	100%	0/0
src/symptom/entities	80%	8/10	100%	0/0
src/utils/mocks	80%	12/15	75%	3/4
src/patients-details/entities	78.26%	18/23	100%	0/0
src/patients-details	59.26%	48/81	27.27%	6/22
src/authentication/dto	50%	2/4	100%	0/0
src/users	30%	21/70	16.67%	3/18
src/medications/dto	25%	1/4	100%	0/0
src/medications	20.45%	27/132	0%	0/54
src	0%	0/8	0%	0/2
src/alergies/dto	0%	0/4	100%	0/0
src/symptom/dto	0%	0/4	100%	0/0

Sl. 5.19. *Prikaz pokrivenosti testova unutar terminala i korištenjem izvješća istanbul HTML*

Tako, tj. pregledom izvješća na lokalnom poslužitelju, se moglo bolje pratiti pisanje testova, testiranje pojedinih funkcionalnosti te pokrivenosti koda. Prema tome, testiranjem se maknula veća količina ne korištenog i redundantnog koda čime se povećala čitljivost i razumljivost koda, smanjila veličina određenih modula i na kraju povećala pokrivenost koda kao rezultat uklanjanja koda. Primjer takve izmjene se može vidjeti na slici 5.20 gdje se većina koda, koja je generirana prilikom kreiranja modula simptoma, maknuta zato što se nije koristila jer se simptomima pa i alergijama, pristupa na drugačiji način prilikom dohvaćanja, spremanja i brisanja što je spomenuto ranije kod postupka kreiranja lijeka.



Sl. 5.20. *Primjer uklanjanja ne korištenog koda iz usluge za simptome*

Dolaskom do testiranja krajnjih točaka primjećuje se i korištenje drugačijeg načina testiranja od onog na korisničkoj strani. Razlog tome je što se za testiranje krajnjih točaka koristi drugi alat za testiranje pod nazivom SuperTest koji je spomenut ranije. SuperTest alata služi za testiranje HTTP tvrdnji na jednostavan način tako da se *request* funkciji preda HTTP poslužitelj prema kojem se šalje jedan od tipova zahtjeva, u slučaju prema slici 5.21 zahtjeva *get*, unutar kojeg se definira ruta koja se poziva. Prema toj ruti tada se mogu slati podaci korištenjem *send* funkcije i definiranjem objekta unutar kojeg su podaci koji se šalju, razna zaglavlja poput autorizacije koje se šalje korištenjem *auth* funkcije, dok se ostala zaglavlja šalju korištenjem *set* funkcije i definiranjem

naziva te vrijednosti zaglavlja. Uz spomenute postavke, SuperTest pruža postavljanje i ostalih postavki koje su bitne za uspješno slanje zahtjeva prema krajnjoj točki.

```
describe('when registering', () => {
  describe('and using valid data', () => {
    it('should respond with the data of the
      const expectedData = {
        ...userData,
      };
      delete expectedData.password;
      return request(app.getHttpServer())
        .post('/authentication/register')
        .send({
          email: mockedUser.email,
          name: mockedUser.name,
          password: 'strongPassword',
        })
        .expect(201)
        .expect(expectedData);
    });
  });
```

```
describe('when fetching all symptoms', () => {
  describe('and using valid authentication', () => {
    it('should respond with the data', () => {
      const expectedData = [];
      return request(app.getHttpServer())
        .get('/symptom')
        .auth(
          'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJp
          { type: 'bearer' },
        )
        .expect(200)
        .expect(expectedData);
    });
  });
});
```

Sl. 5.21. *Primjer testa krajnje točke za registraciju i dohvaćanje svih simptoma*

Kako bi se provjerio odziv na zahtjev s krajnje točke, kao i kod testiranja s Jest alatom, koristi se *expect* funkcija koja prima parametre ili razne vrijednosti s kojima uspoređuje rezultat, ako se vrijednosti podudaraju, test koji je pokrenut će proći i prikazat će se u terminalu kao na slici 5.22, a HTML izvješće će se automatski ažurirati s novom pokrivenosti.

```
PASS src/alergies/tests/alergies.service.spec.ts
PASS src/alergies/tests/alergies.controller.spec.ts (5.966 s)
PASS src/symptom/tests/symptom.controller.spec.ts (5.993 s)
PASS src/symptom/tests/symptom.service.spec.ts
PASS src/patients-details/tests/patient-detail.service.spec.ts
PASS src/authentication/tests/authentication.controller.integration.spec.ts (6.176 s)

PASS src/patients-details/tests/patient-detail.service.spec.ts
  PatientDetailsService
    √ should be defined (36 ms)
    when creating new patient
      √ should return patient and success message (7 ms)
      √ should return errors when wrong data is sent (6 ms)
    when fetching all patient details
      √ should return an array (5 ms)
    when updating patient
      √ should return success message (5 ms)
      √ should return errors when wrong data is sent (8 ms)
```

Sl. 5.22. *Prikaz uspješnih testova i uspješnih pojedinačnih testova*

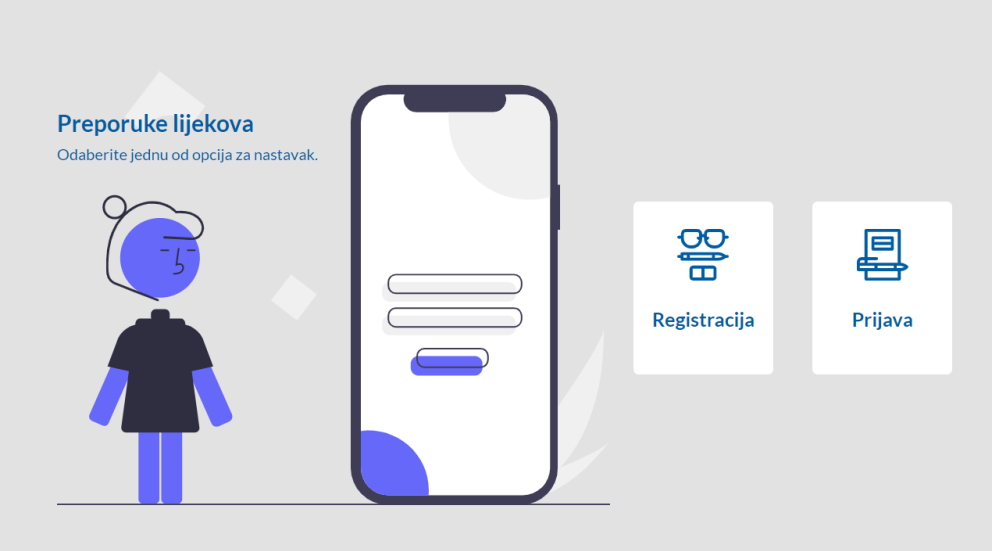
Iz testiranja poslužiteljske strane može se zaključiti da je pisanje testova, kao i ručno testiranje, dalo doprinos ispravljanju grešaka, poboljšanju funkcionalnosti, povećanju čitljivosti te smanjenju veličini koda čemu se teži prilikom razvoja svakog sustava.

6. KORIŠTENJE I ISPITIVANJE WEB SUSTAVA ZA STVARANJE PREPORUKA PRIKLADNIH LIJEKOVA

Nakon kreiranja i testiranja web sustava, potrebno je opisati kako koristiti web sustav i njegove funkcionalnosti. U ovom poglavlju govori se o korisničkim uputama, tj. načinu korištenja, web sustava, prikazu slučajeva s kojima se pokazuje točnost rada web sustava te analiza rezultata.

6.1. Opis načina korištenja

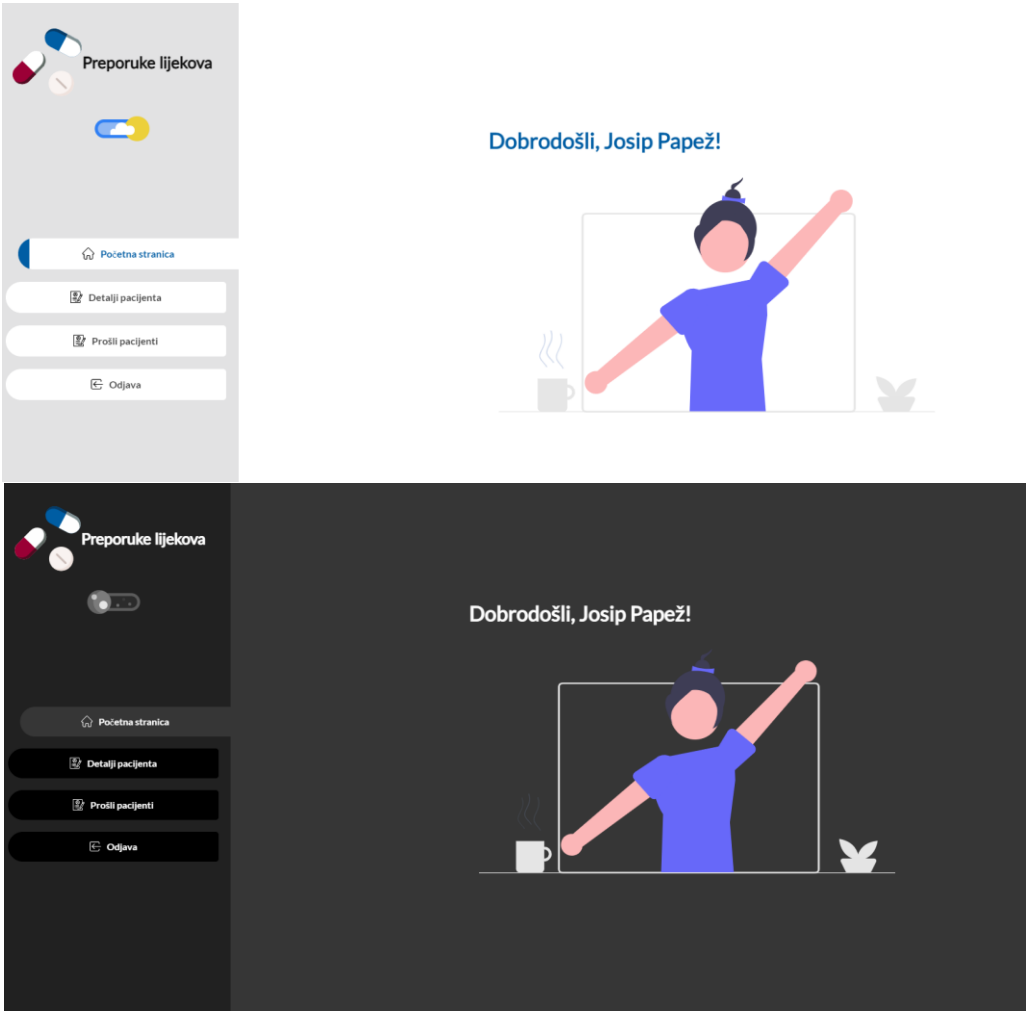
U prošlim poglavljima je detaljnije opisano na koji način funkcionira web sustav te što je potrebno da bi se odradio određeni zadatak. Zbog toga, potrebno je napisani kratak opis načina korištenja web sustava kako bi se nalazile sve potrebne informacije na jednom mjestu za nove korisnike. Novi i stari korisnici dolaskom na naslovnu stranicu web sustava se mogu registrirati i prijaviti u web sustav upisivanjem podataka. Pritiskom na tipku za potvrdu unosa na formama za registraciju i prijava korisnik se prijavljuje u web sustav te je prebačen na početnu stranicu web sustava kao što je prikazano na slici 6.2.



Sl. 6.1. *Prikaz naslovne stranice s opcijama za registraciju i prijavu*

6.1.1. Običan korisnik

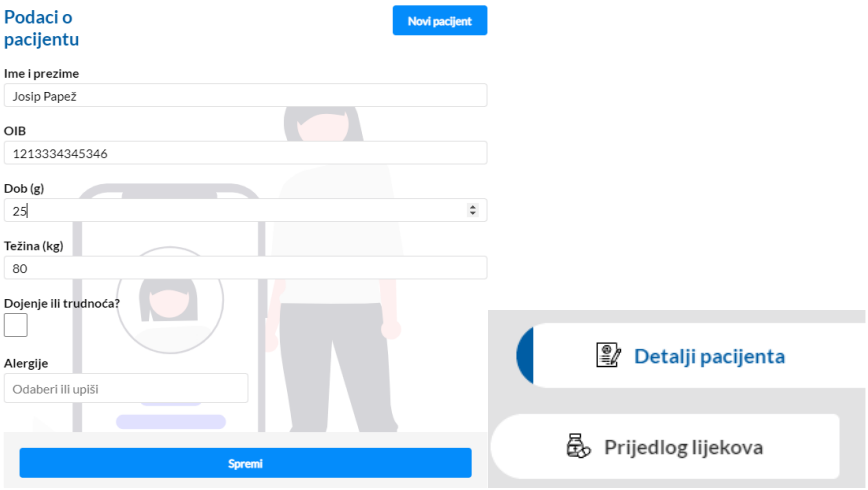
Običan korisnik na web sustavu ima pristup dijelovima stranice koje sadrže sve funkcionalnosti za uspješno predlaganje lijekova za pacijenta. Uz to, svi korisnici imaju i mogućnost prelaska na tamni način stranice pritiskom na tipku ispod logotipa stranice.



Sl. 6.2. *Početna stranica u svijetlom i tamnom načinu nakon prijave*

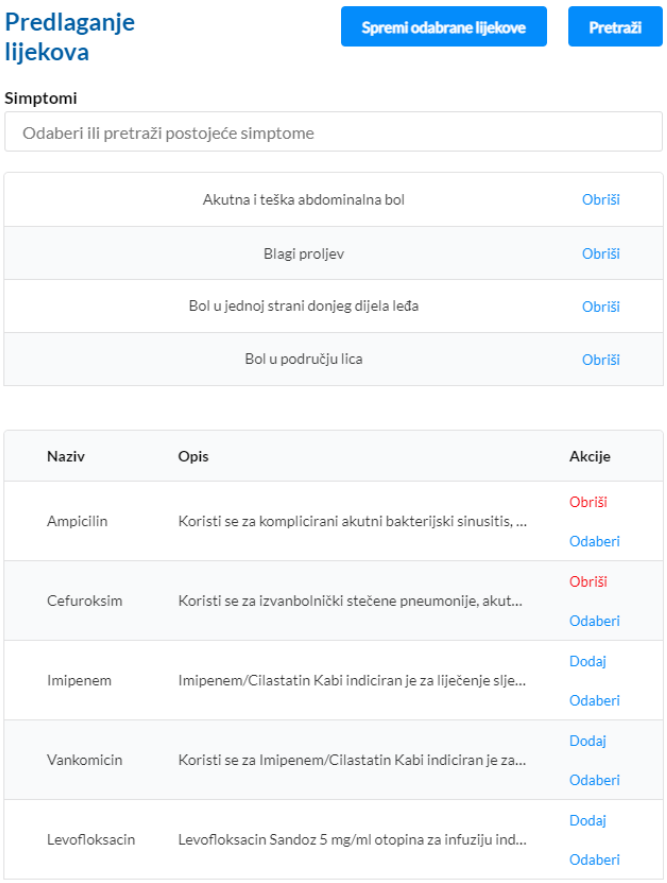
Kako bi se započeo proces predlaganja lijekova za pacijenta, potrebno je otići na stranicu s detaljima pacijenta gdje se upisuju podaci o pacijentu. Na tom dijelu stranice, prikazanog na slici 6.3, potrebno je unijeti podatke o pacijentu za kojeg se želi vršiti predlaganje lijekova. Podaci se naknadno mogu i uređivati za pacijenta promjenom podataka unutar unosa te pritiskom na tipku „Spremi“. Također, novi pacijent se dodaj pritiskom na tipku „Novi Pacijent“. U slučaju da korisnik pritisne na tipku za kreiranje novog pacijenta, stari pacijent će se moći pregledavati, ako je prethodno spremljen, u listi prošlih pacijenata koja će kasnije biti prikazan.

Nakon spremanja informacija o pacijentu, u navigacijskoj traci pojavljuje se nova stavka s nazivom „Predlaganje lijekova“ koja vodi korisnika na dio stranice za predlaganje lijekova. Na tom dijelu stranice korisnik odabire simptome za pacijenta iz padajućeg izbornika koji će biti korišteni za višekriterijsko filtriranje lijekova.



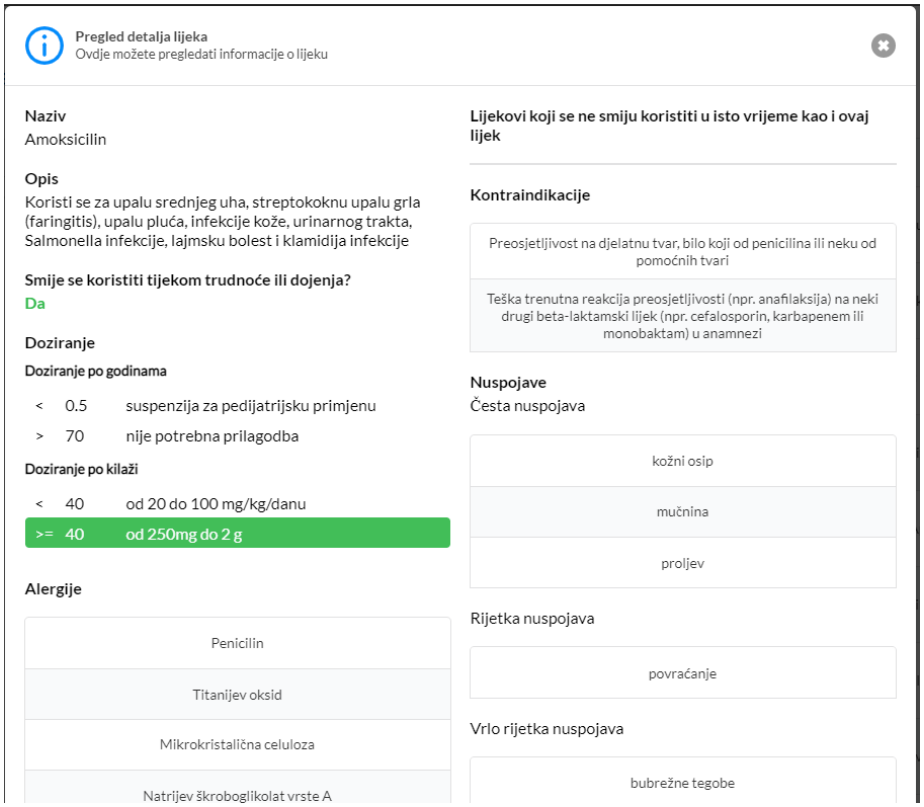
Sl. 6.3. *Primjer unosa detalja pacijenta i nove stavke u navigacijskoj traci*

Pritiskom na tipku „Pretraži“, korisniku se pojavljuje lista lijekova za predane simptome kao na slici 6.4. Uz simptome, prilikom filtriranja lijekova uzimaju se u obzir i podaci pacijenta unesenih na dijelu stranice s detaljima o pacijentu. Podaci koji se uzimaju u obzir su: doji li pacijent ili je trudan, odabrane alergije za pacijenta te simptomi odabrani za pacijenta.



Sl. 6.4. *Prikaz liste lijekova za odabrane simptome*

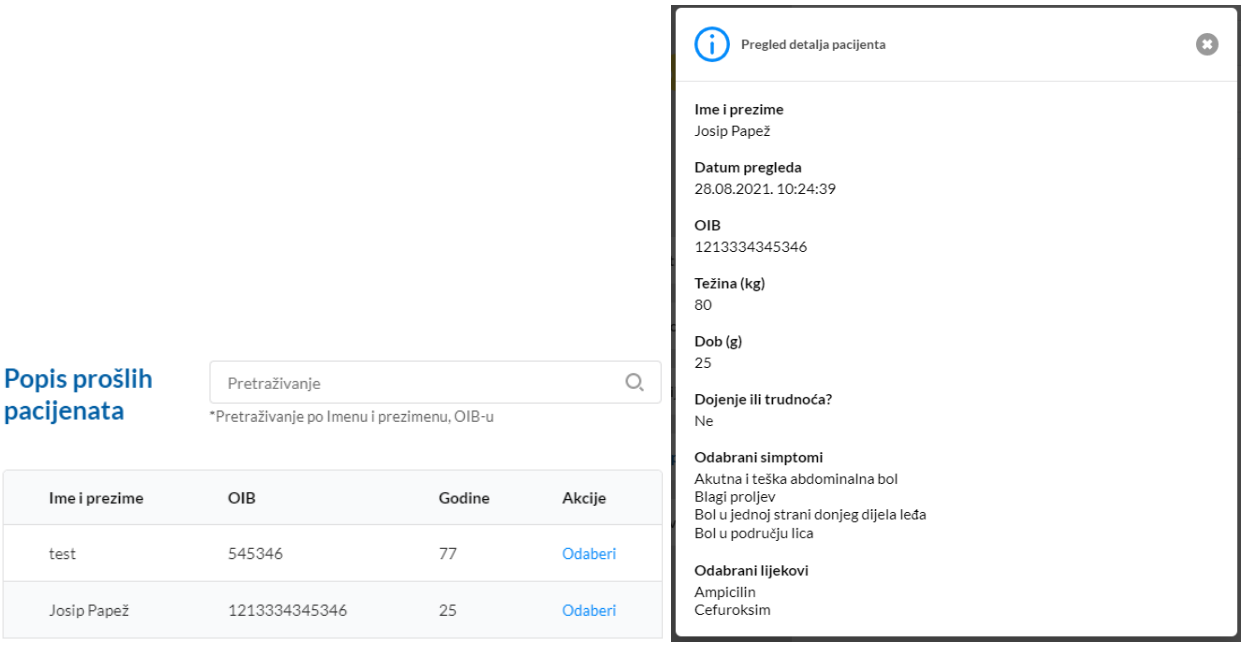
Na listi lijekova korisnik može odabrati pojedini lijek, pregledati ga i spremiti odabrane lijekove za pacijenta. Pritiskom na tipku „Odaberi“ pokraj lijeka kojeg se želi pregledati, otvara se detaljan prikaz za odabrani lijek. Unutar njega se mogu pregledavati detalji te simptomi za koje se koristi kao i nuspojave, kontraindikacije te lijekovi koji se ne smiju koristiti zajedno sa odabranim lijekom.



Sl. 6.5. *Detaljan prikaz lijeka*

Pritiskom na tipku „Spremi odabrane lijekove“, svi odabrani lijekovi se spremaju za pacijenta u bazu podataka te je moguće vidjeti spremljene podatke u listi prošlih pacijenata do koje se dolazi odabirom odgovarajuće stavke na navigacijskoj traci. Popis prošlih pacijenata sadrži listu pacijenata u kojoj se može pristupiti detaljnom pregledu svakog pacijenta te unosu za pretraživanje pacijenata po imenu i prezimenu te OIB-u. Odabirom pacijenta pritiskom na tipku „Odaberi“ otvara se detaljan prikaz unosa o tom pacijentu, kao što je prikazano na slici 6.6, gdje se može vidjeti postavljeni podaci pacijenta te odabrani simptomi i lijekovi za pacijenta.

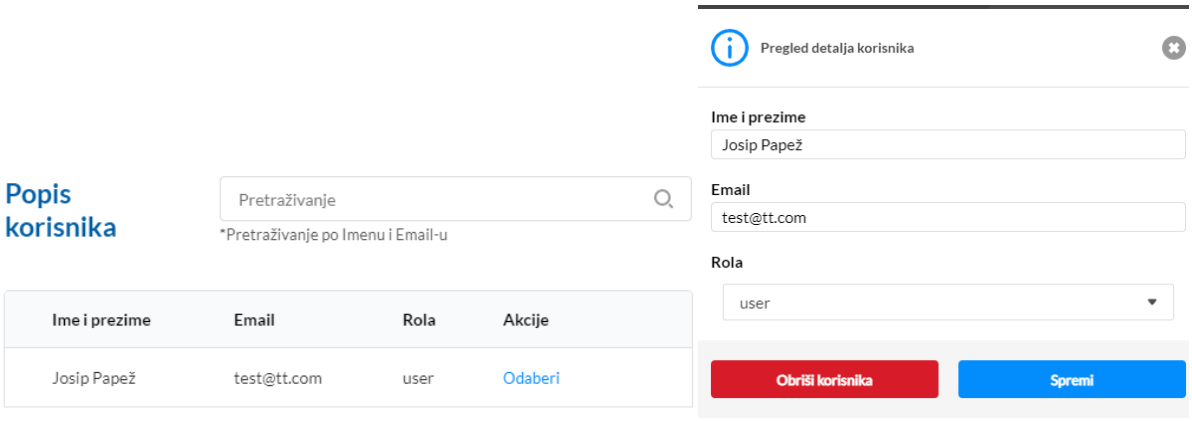
Zadnja stavka koju svaki korisnik ima na navigacijskoj traci je odjava. Pritiskom na nju korisnik se odjavljuje te se svi lokalno spremljeni podaci za stranicu, poput unesenih detalja o pacijentu ili odabranih simptoma, brišu i korisnika se prebacuje na naslovnu stranicu gdje se može ponovno prijaviti ili registrirati.



Sl. 6.6. *Popis prošlih pacijenata i detaljan prikaz odabranog pacijenta*

6.1.2. Administrator kao korisnik

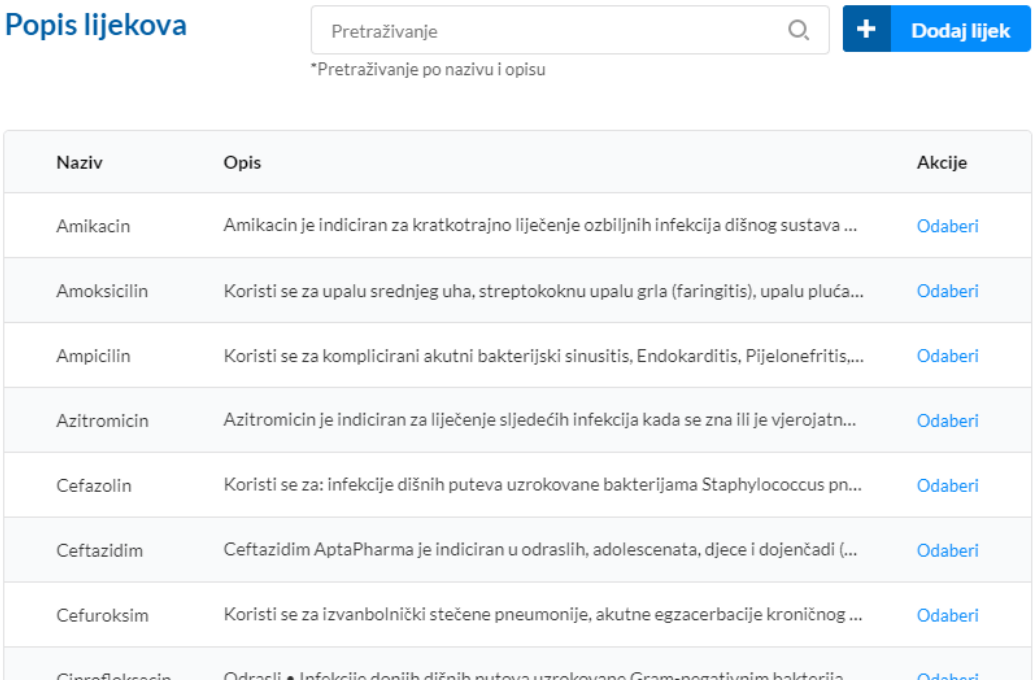
Administrator, za razliku od običnog korisnika, ima pristup dijelovima stranice za pregled liste korisnika, tj. liječnika, lijekova i podataka o pacijentima. Uz neke funkcionalnosti kojima kože pristupiti običan korisnika, administrator ima i dodatne pomoću kojih može uređivati sadržaj na stranici pa te ažurirati korisnike. Prema tome, prvi dio stranice kojem administrator može pristupiti je lista korisnika. Na njoj može pretraživati sve korisnike, po imenu i e-pošti, koji su se registrirali na stranicu, uređivati im podatke te brisati ih s liste. Uređivanje i brisanje se odvija tako da administrator odabere korisnika kojeg želi urediti pritiskom na tipku „Odaberi“ pokraj korisnike te u detaljnom prikazu odabire akcije za korisnika.



Sl. 6.7. *Popis svih korisnika i detaljan prikaz odabranog korisnika*

Pritiskom na tipku za brisanje korisnika, prikazuje se upozorenje o brisanju korisnika u slučaju da je ne namjerno pritisnuta tipka, isto tako upozorenje se prikazuje ako su izmijenjeni neki podatci i potom se izlazi iz prikaza prije nego što su podaci spremljeni pritiskom na tipku „Spremi“.

Drugi dio stranice kojem administrator ime pristup je lista lijekova. Kao i kod dosadašnjih lista, na ovoj se mogu pregledavati detalji o pojedinom lijeku, uređivati ih, dodavati novi lijekovi te pretraživati postojeći po nazivu i opisu. Pritiskom na tipku „Dodaj lijek“ otvara se prikaz za dodavanje novog lijeka gdje se unose sve potrebne informacije kako bi se lijek mogao ispravno prikazati i upotrijebiti prilikom predlaganja lijekova.

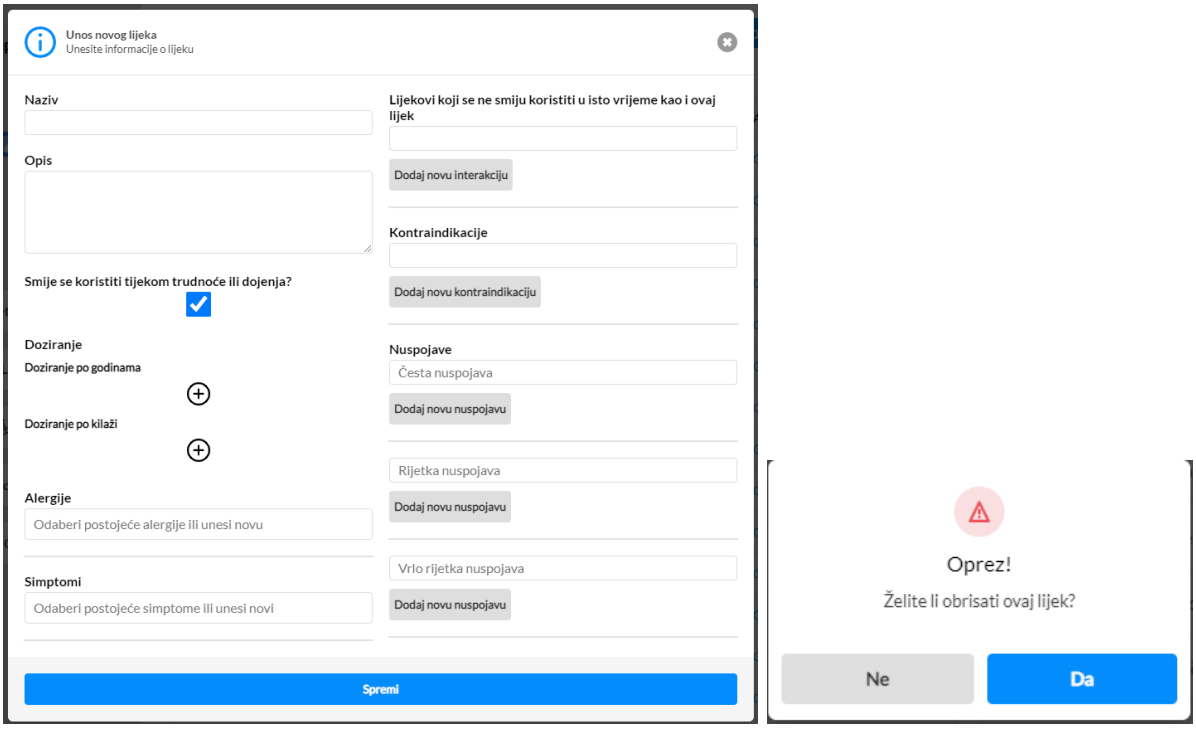


Sl. 6.8. *Prikaz dijela popisa svih lijekova u sustavu*

Alergije i simptomi kod lijeka, kao i kod detalja pacijenata, se odabiru iz padajućeg izbornika, no administrator može dodati i nove alergije i simptome tako da upiše naziv simptoma ili alergije u unos kod padajućeg izbornika za simptome ili alergije. Padajući izbornik funkcionira također kao unos za pretraživanje postojećih podataka i prema tome, ako uneseni podatak ne postoji u padajućem izborniku, prikazat će se tipka za dodavanje na listu s nazivom „Dodaj“. Time se novo dodani podaci prikazuju u listi alergija ili simptoma gdje se mogu i naknadno maknuti ako je to potrebno.

Popunjavanjem svih polja za lijek i pritiskom na tipku „Spremi“, lijek se dodaje u listu svih lijekova kojem mogu zatim pristupiti svi korisnici i administratori nakon dohvaćanja liste. U slučaju da je potrebno izmijeniti neke podatke na lijeku, potrebno je odabrati lijek pritiskom na

„Odaberi“ tipku pokraj lijeka i detaljan prikaz će se pojaviti. Detaljan prikaz je isti prikaz kao za dodavanje novog lijeka, ali s popunjenim podacima o odabranom lijeku koji se mogu uređivati. U slučaju da se izmjene neki podaci i da se želi izaći iz detaljnog prikaza prije spremanja podataka, pojavit će se upozorenje kao i kod uređivanja korisnika. Isto tako, u slučaju pritiska na „Obriši lijek“ pojavit će se identično upozorenje kao kod brisanja korisnika što je prikazano na slici 6.9.



Sl. 6.9. *Forma za unos novog lijeka i prikaz upozorenja pri brisanju*

Naposljetku, zadnji dio stranice kojem administrator može pristupiti je popisu svih prošlih pacijenata. Popis je identičan onome kod običnog korisnika, jedina razlika je što administrator ima mogućnost brisanja unosa o pacijentu nakon otvaranja detaljnog prikaza pacijenta. Tipka za brisanje prikazana je na dnu detaljnog prikaza o pacijentu i pritiskom na nju pojavljuje se upozorenje o brisanju kao kod brisanja lijeka ili korisnika, ali s drugačijom porukom.

Sl. 6.10. *Prikaz tipke za brisanje unosa i upozorenja pri brisanju*

6.2. Uvjeti ispitivanja

Kao što je spomenuto u prošlom poglavlju, potrebno je provjeriti ispravnost rada web sustava s naglaskom na stvaranje preporuka prikladnih lijekova s višekriterijskim filtriranjem. To se izvršava tako da se unesu realni podaci za pacijenta za kojeg se pretpostavi da ima određenu bolest, a da se za tu bolest koriste određeni lijekovi uneseni u bazu. Ovakav način ispitivanja je potreban kako bi se provjerila pokrivenost predlaganja lijekova, koji spadaju u skupinu antibiotika, za različite profile pacijenata s različitim bolestima. Također, uzimajući u obzir testiranje i način na koji je sustav napravljen, u teoriji bi se mogla proširiti baza lijekova na druge skupine lijekova bez da se utječe na glavne funkcionalnosti i bez potrebe za nekim većim izmjenama sustava. Najbitnija stvar o kojoj trenutno predlaganje lijekova ovisi je o količini, konzistentnosti i kvaliteti podataka, što govori da je najvažnije prikupiti dobre podatke prije proširivanja sustava na nove vrste lijekova.

U prošlom poglavlju je prilikom ručnog testiranja izvršeno ispitivanje za bolesti bronhitis i upala srednjeg uha čime se provjerio rad dijela stranice za predlaganje lijekova, korisničkog iskustva kao i poslužiteljske strane koja vrši višekriterijsko filtriranje lijekova. Uz spomenuta dva slučaja, ovdje će se izvršiti još koji slučaj kako bi se dodatno potvrdila točnog sustava za predlaganje lijekova s obzirom na detalje o pacijentu.

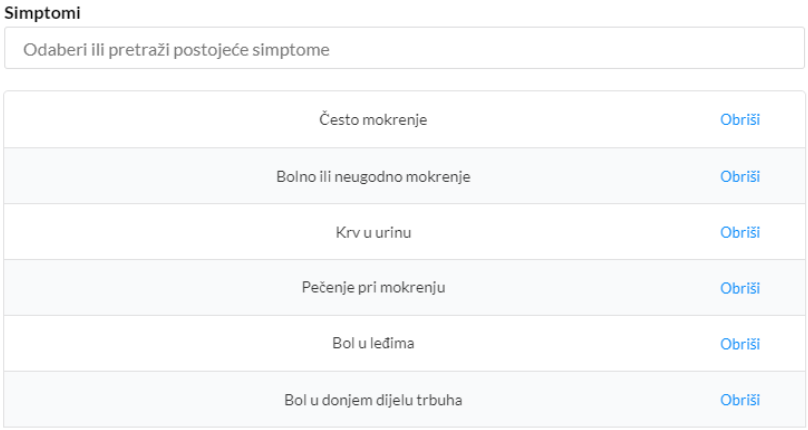
6.2.1. Primjer korištenja 1

Prvi primjer koji će se koristiti je bolest sinusitis. Tipični simptomi sinusitisa su začepljen nos, pojačano curenje nosa, osjećaj pritiska i bol u području zahvaćenog sinusa te može biti prisutna glavobolja kao i povišena temperatura, dok se simptomi kroničnog sinusitisa ne razlikuju značajno osim što duže traju [29]. Prema navedenim simptomima odabrani su najsličniji simptomi iz padajućeg izbornika te su prikazani na slici 6.11.

Sl. 6.11. *Odabrani simptomi za sinusitis*

6.2.1. Primjer korištenja 2

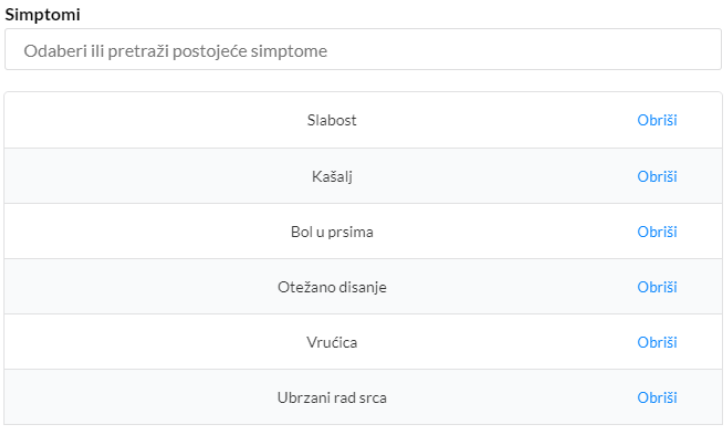
Drugi primjer za promatranje je infekcija donjeg urinarnog trakta, tj. cistitis. Simptomi za cistitis su učestalo mokrenje, tamniji ili krvavi urin, pečenje pri mokrenju, bol u leđima te bol u donjem dijelu trbuha, na mjestu mokraćnog mjehura [30]. Odabir prikladnih simptoma u web sustavu je vidljiv na slici 6.12.



Sl. 6.12. *Odabrani simptomi za cistitis*

6.2.1. Primjer korištenja 3

Naposljetku, zadnji primjer koji će se promatrati je primjer izvanbolničke pneumonije. Simptomi takve pneumonije su slabost, kašalj, otežano disanje, bol u prsima, ubrzano plitko disanje, vrućica i ubrzani rad srca [31]. Na slici 6.13 se mogu vidjeti odabrani simptomi za izvanbolničku pneumoniju pri čemu je odabrano otežano disanje bez simptoma za ubrzano plitko disanje.



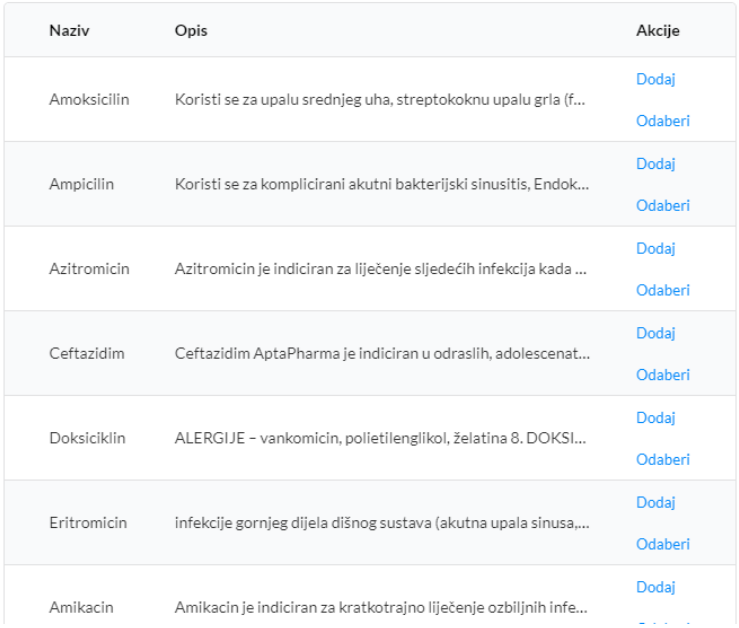
Sl. 6.13. *Odabrani simptomi za izvanbolničku pneumoniju*

Prilikom isprobavanja navedenih primjera za pacijenta je postavljeno da nema alergija te da pacijent nije trudan i ne doji. Tako postavljeni podaci mogu utjecati na predložene lijekove jer se uzimaju u obzir prilikom filtriranja lijekova

6.3. Rezultati ispitivanja s analizom

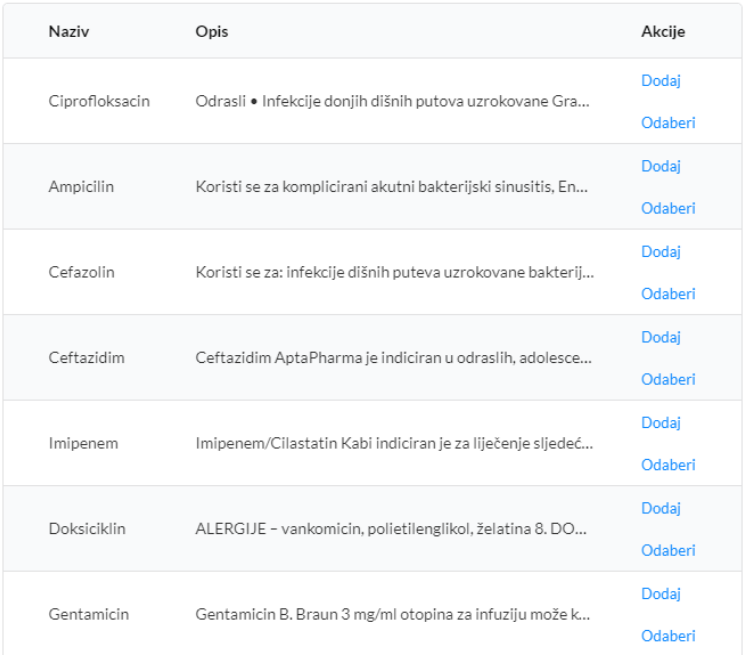
Rezultati ispitivanja navedenih bolesti, kao i svakog drugog ispitivanja, ovise o količini, točnosti te konzistentnosti unesenih podataka za pojedini lijek. Razlog tome je što se podaci uneseni za lijek prikazuju i korisniku poput simptoma i alergija koje ovise o lijeku te njihova ujednačenost direktno utječe na točnost predlaganja lijekova. Isto tako, gotovo svaki lijek se koristi za više različitih bolesti koje mogu imati slične ili iste simptome kao neke druge bolesti. Zbog toga, u većini rezultata prikazuju se lijekovi koji se najviše slažu s predanim podacima i lijekovi koji u sebi sadrže barem jedan simptom iz liste predanih simptoma po kojima se filtriraju lijekovi uz uvjet da se slažu s ostalim podacima. Iz tog razloga se prije prikazivanja predloženih lijekova korisniku, u pozadini izvršava sortiranje prema broju simptoma koji se podudaraju od većeg broja podudaranja prema nižem.

Prema tome, za sinusitis se dobiva lista predloženih lijekova prikazana na slici 6.14. Prema HALMED-u, Azitromicin, Ampicilin, Amoksicilin, Doksiciklin i Eritromicin se koriste za liječenje neke vrstu sinusitisa dok se Ceftazidim i Amikacin koriste za druge bolesti koje imaju iste simptome poput ove. Isto tako, može se primijetiti kako se neki spomenuti lijekovi koji se koriste za liječenje sinusitisa podudaraju manje nego lijekovi koji se ne koriste. Razlog tome je točnost unesenih podataka o kojoj se govorilo ranije te ne postojanje svih podataka vezanih za lijek. Primjer tome su simptomi u Eritromicinu gdje nisu navedeni svi mogući simptomi za sve bolesti za koje se lijek koristi.



Sl. 6.14. *Predloženi lijekovi za sinusitis*

Za razliku od sinusitisa, svi predloženi lijekovi u listi za bolest cistitis, prema HALMED-u, se koriste za njegovo liječenje. Uz to, primjećuje se da se određeni lijekovi ponovno pojavljuju i za ovu bolest što dodatno dočarava korištenje lijekova za različite bolesti i uz to, njihovo pojavljivanje na listama za razne simptome. Također, potrebno je naglasiti kako su na slici 6.14 prikazani lijekovi koji se podudaraju s vrijednostima postavljenim za pacijenta, a to znači da se svi lijekovi prikazani mogu koristiti tijekom dojenja i da niti jedan nije uklonjen s liste zbog alergija pacijenta.

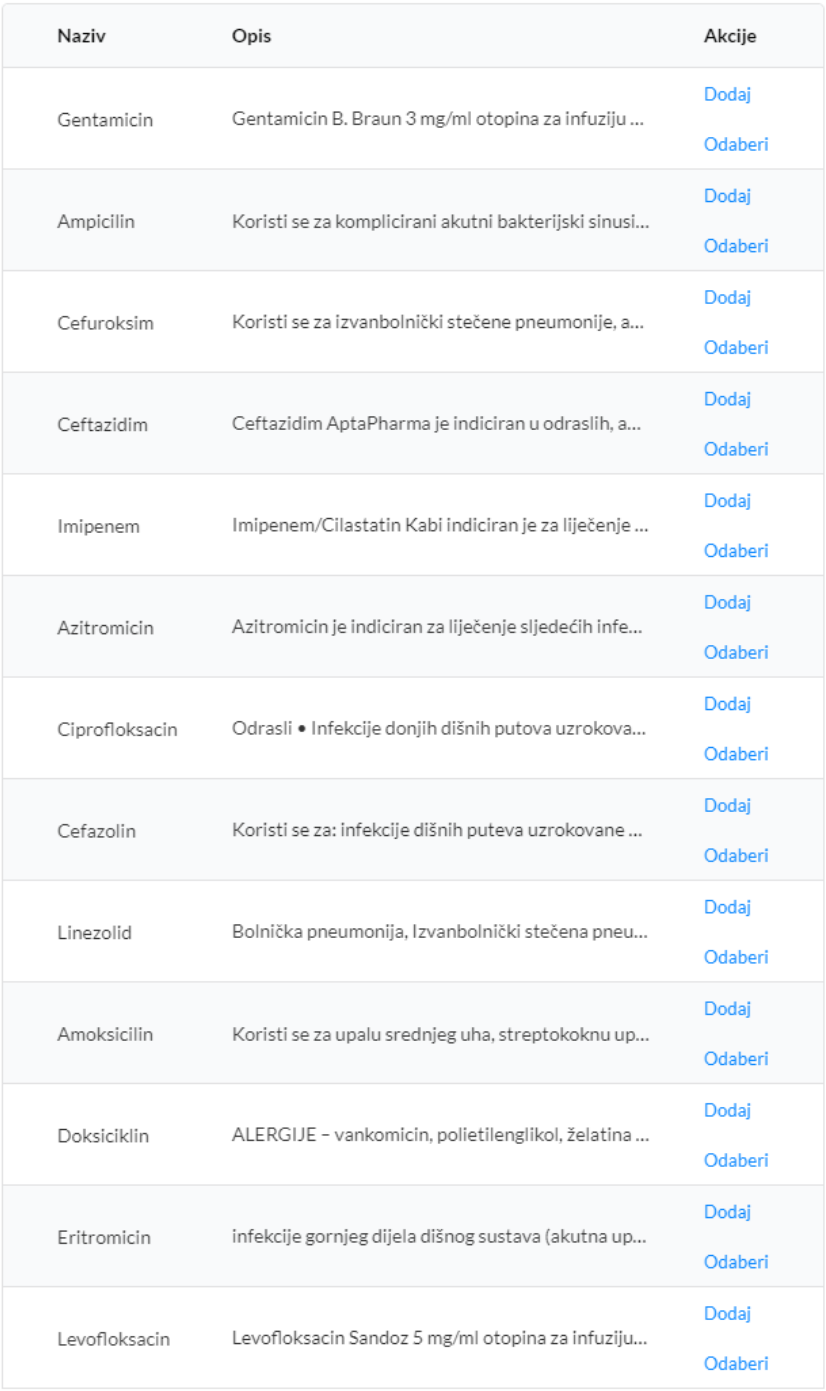


Naziv	Opis	Akcije
Ciprofloksacin	Odrasli • Infekcije donjih dišnih putova uzrokovane Gra...	Dodaj Odaberi
Ampicilin	Koristi se za komplicirani akutni bakterijski sinusitis, En...	Dodaj Odaberi
Cefazolin	Koristi se za: infekcije dišnih puteva uzrokovane bakterij...	Dodaj Odaberi
Ceftazidim	Ceftazidim AptaPharma je indiciran u odraslih, adolesce...	Dodaj Odaberi
Imipenem	Imipenem/Cilastatin Kabi indiciran je za liječenje sljedeć...	Dodaj Odaberi
Doksiciklin	ALERGIJE – vankomicin, polietilenglikol, želatina 8. DO...	Dodaj Odaberi
Gentamicin	Gentamicin B. Braun 3 mg/ml otopina za infuziju može k...	Dodaj Odaberi

Sl. 6.14. *Predloženi lijekovi za cistitis*

Naposljetku za bolest pneumonija, prema predanim simptomima predložena je duža lista lijekova na slici 6.15. U ovom slučaju, količina predloženih lijekova ovisi najviše o količini predanih simptoma, njihovoj raznolikosti te zastupljenosti unutar svakog lijeka. Primjer tome bi bio simptom kašalj koji se može povezati s velikim brojem bolesti i prema tome, može se nalaziti unutar gotovo svakog lijeka. Na isti se način vrućica i slabost mogu povezati s velikom količinom bolesti.

Prema svemu navedenom, u predloženoj listi lijekova može se nalaziti i velika količina lijekova koji se ne koriste za liječenje pneumonije. Iz prvih 10 predloženih lijekova, prema HALMED-u 8 lijekova se koristi u liječenju barem jedne pneumonije, dok se Ampicilin i Ciprofloksacin ne koriste za njeno liječenje.



Naziv	Opis	Akcije
Gentamicin	Gentamicin B. Braun 3 mg/ml otopina za infuziju ...	Dodaj Odaberi
Ampicilin	Koristi se za komplicirani akutni bakterijski sinusi...	Dodaj Odaberi
Cefuroksim	Koristi se za izvanbolnički stečene pneumonije, a...	Dodaj Odaberi
Ceftazidim	Ceftazidim AptaPharma je indiciran u odraslih, a...	Dodaj Odaberi
Imipenem	Imipenem/Cilastatin Kabi indiciran je za liječenje ...	Dodaj Odaberi
Azitromicin	Azitromicin je indiciran za liječenje sljedećih infe...	Dodaj Odaberi
Ciprofloksacin	Odrasli • Infekcije donjih dišnih putova uzrokova...	Dodaj Odaberi
Cefazolin	Koristi se za: infekcije dišnih puteva uzrokovane ...	Dodaj Odaberi
Linezolid	Bolnička pneumonija, Izvanbolnički stečena pneu...	Dodaj Odaberi
Amoksicilin	Koristi se za upalu srednjeg uha, streptokoknu up...	Dodaj Odaberi
Doksiciklin	ALERGIJE – vankomicin, polietilenglikol, želatina ...	Dodaj Odaberi
Eritromicin	infekcije gornjeg dijela dišnog sustava (akutna up...	Dodaj Odaberi
Levofloksacin	Levofloksacin Sandoz 5 mg/ml otopina za infuziju...	Dodaj Odaberi

Sl. 6.15. *Predloženi lijekovi za pneumoniju*

Zaključno s time, može se reći da će se s više unesenih simptoma podudarati više lijekova, no uz to će točnost sustava narasti jer će se provjeravati prema većem broju kriterija što i je cilj kako bi se dobili pouzdaniji i točniji podaci korištenjem višekriterijskog filtriranja. Isto tako, sustav se može uvijek unaprijediti s novim načinima filtriranja i novim podacima, no uvijek će se morati dio podataka provjeravati ručno, u ovom slučaju predlaganje lijekova, kako bi se osigurao prijedlog odgovarajućih lijekova za odgovarajuće bolesti.

7. ZAKLJUČAK

U ovom diplomskom radu prikazan je cijeli postupak izrade web sustava za stvaranje preporuka prikladnih lijekova i testiranje sustava s analizom rezultata. Najprije je objašnjen postupak izbora prikladnih lijekova s obzirom na podatke o pacijentu i lijeku i analiziran utjecaj tih podataka na kvalitetu preporuke lijekova. Nadalje, analizirani su postojeći sustavi za pretraživanje lijekova prema podacima o bolesti, lijeku, dijelu medicine i drugih pokazateljima. Predloženo je idejno rješenje web sustava za stvaranje preporuka prikladnih lijekova i navedeni potrebni funkcionalni i nefunkcionalni zahtjevi na sustav, arhitektura sustava, odabir načina filtriranja lijekova i potrebne vrste testiranja. Opisani su korišteni programski jezici, razvojna okolina i alati za provedbu testiranja i izradu web sustava te je opisano programsko rješenje na korisničkoj i poslužiteljskoj strani. Nakon toga, provedena su odvojena testiranja korisničkog i poslužiteljskog dijela sustava kao i cjelokupnog sustava postavljenog na udaljeni poslužitelj. Dane su upute za korištenje sustava i provedena ispitivanja za različite profile pacijenata i uvjete korištenja lijekova.

Ispitivanjem web sustava za preporuku prikladnih lijekova tijekom i nakon postupka izrade, potvrđeno je ostvarenje funkcionalnih i nefunkcionalnih zahtjeva na sustav. Rezultati ispitivanja tijekom razvoja potvrdili su ispravan rad web sustava s obzirom na predane simptome i postavljene parametre o pacijentu za bolesti bronhitis i upala srednjeg uha. Ispitivanja provedena nakon razvoja web sustava za bolesti sinusitis, cistitis i pneumoniju dodatno su potvrdila ispravnost rada sustava, ali i dodatno ukazale na potencijalne probleme koji se mogu dogoditi u sustavu. S obzirom na količinu simptoma za razne bolesti te njihovu sličnost, zaključeno je da je potrebno unositi što točnije simptome i simptome u jednakom obliku jer se simptomi provjeravaju prema nazivu. S obzirom na to, sustav je moguće dodatno unaprijediti tako da se dodatno olakša odabir ili omogući bolji prikaz simptoma, dodaju novi parametri prema kojima se filtriraju lijekovi ili promijeni način filtriranja lijekova s nekim drugim, prikladnijim načinom koji bi podržavao filtriranje velike količine podataka.

LITERATURA

[1] B. G. Katzung, Basics & Clinical Pharmacology, 14. ur., McGraw Hill Education / Medical, 2017. [25.5.2021.]

[2] E. Stević, Z. Jatić, E. Salihefedić, A. Kadić, Dobra praksa propisivanj i izdavanja lijekova, Sarajevo: Ministarstvo zdravstva Kantona Sarajevo, Institut za naučnoistraživački rad i razvoj Kliničkog centra Univerziteta u Sarajevu, 2011. [25.5.2021.]

[3] T. Ivković-Lazar, T. Kovač, L. Lepšanović, D. Pejin, K. Popović, Đ. Tabori, P. Tepavčević, S. Trifunović, M. Živanović, Praktikum fizičke dijagnostike sa osnovama interne propedevtike, Novi Sad, 2001. [25.5.2021.]

[4] G. D. Hammer, S. J. McPhee, Pathophysiology of Disease: An Introduction to Clinical Medicine, 8. ur., McGraw-Hill, 2018. [27.5.2021.]

[5] E. Stević, Z. Dizdarević, S. Trninić, Vodič za zdrav i dug život, Sarajevo, 2007. [27.5.2021.]

[6] P. Aspden, J. Wolcott, J. L. Bootman i L. R. Cronenwett, Preventing Medication Errors: Quality Chasm Series, National Academies Press, 2006. [30.5.2021.]

[7] R. B. Taylor, Essential Medical Facts Every Clinician Should Know: To Prevent Medical Errors, Pass Board Examinations and Provide Informed Patient Care, Springer-Verlag New York, 2011. [30.5.2021.]

[8] D. W. Bates, J. M. Teich, J. Lee, D. Seger, G. J. Kuperman, N. Ma'Luf, D. Boyle, L. Leape, The Impact of Computerized Physician Order Entry on Medication Error Prevention, *Journal of the American Medical Informatics Association*, svez. 6, br. 4, p. 313., srpanj/kolovoz 1999. [31.5.2021.]

[9] Baza lijekova Mediately, Google Play, Modra Jagoda, [Mrežno]. Dostupno na: https://play.google.com/store/apps/details?id=si.modrajagoda.bazalijekova. [31.5.2021.]

[10] Medscape, Google Play, WebMD LLC, [Mrežno]. Dostupno na: https://play.google.com/store/apps/details?id=com.medscape.android. [31.5.2021.]

[11] D. Bjørner, Software Engineering 3: Domains, Requirements, and Software Design, Springer, 2006. [5.6.2021.]

[12] L. Chung, B. A. Nixon, E. Yu, J. Mylopoulos, Non-Functional Requiremnets in Software Engineering, Springer US, 2000. [5.6.2021.]

[13] M. Erder, P. Pureur i E. Woods, Continuous Architecture in Practice: Software Architecture in the Age of Agility and DevOps, Addison-Wesley Professional, 2021. [5.6.2021.]

[14] N. H. Zardari, K. Ahmed, S. M. Shirazi i Z. B. Yusop, Weighting Methods and their Effects on Multi-Criteria Decision Making Model Outcomes in Water Resources Management, Springer International Publishing, 2015. [6.6.2021.]

[15] E. Triantaphyllou, Multi-Criteria Decision Making Methods: A Comparative Study, Springer US, 2000. [6.6.2021.]

[16] A. Deluka-Tibljaš, B. Karleuša, N. Dragičević, Pregled primjene metoda višekriterijske analize pri donošenju odluka o prometnoj infrastrukturi, *Građevinar,* svez. 65, br. 7, pp. 619-631, 2013. [6.6.2021.]

[17] X. Su, T. M. Khoshgoftaar, A Survey of Collaborative Filtering Techinques, *Advances in Artificial Intelligence*, svez. 2009, pp. 1-19, 2009. [8.6.2021.]

[18] L. Si, R. Jin, Flexible Mixture Model for Collaborative Filtering, *ICML,* pp. 704-711, 2003. [8.6.2021.]

[19] C. C. Aggarwal, Recommender System: The Textbook, Springer US, 2016. [8.6.2021.]

[20] W. Zhou, J. Wen, Q. Qu, J. Zeng, T. Cheng, Shilling Attack Detection for Recommender Systems Based on Credibility of Group Users and Rating Time Series, *PLoS ONE,* svez. 13, br. 5, pp. 1-17, 2018. [8.6.2021.]

[21] NestJS, nestjs.com, [Mrežno]. Dostupno na: https://nestjs.com/. [5.7.2021.]

[22] React, reactjs.org, [Mrežno]. Dostupno na: https://reactjs.org/. [5.7.2021.]

[23] JavaScript HTML DOM, w3schools.com, [Mrežno]. Dostupno na: https://www.w3schools.com/js/js_htmldom.asp. [5.7.2021.]

[24] Stress testing, en.wikipedia.org, [Mrežno]. Dostupno na: https://en.wikipedia.org/wiki/Stress_testing. [5.7.2021.]

[25] D. Westerveld, API Testing and Development with Postman, Birmingham-Mumbai: Packt Publishing, 2021. [7.7.2021.]

[26] Insomnia, insomnia.net, [Mrežno]. Dostupno na: https://insomnia.rest/. [7.7.2021.]

[27] DBeaver, dbeaver.io, [Mrežno]. Dostupno na: https://dbeaver.io/. [7.7.2021.]

[28] Jest, jestjs.io, [Mrežno]. Dostupno na: https://jestjs.io/. [7.7.2021.]

[29] PLIVAzdravlje, plivazdravlje.hr, [Mrežno]. Dostupno na: https://www.plivazdravlje.hr/bolest-clanak/bolest/73/Sinusitis.html. [10.8.2021.]

[30] PLIVAzdravlje, plivazdravlje.hr, [Mrežno]. Dostupno na: https://www.plivazdravlje.hr/aktualno/clanak/31237/Ponavljajuce-urinarne-infekcije-dijagnoza-lijecenje-i-savjeti.html. [10.8.2021.]

[31] MSD Priručnici, msd-prirucnici.placebo.hr, [Mrežno]. Dostupno na: http://www.msd-prirucnici.placebo.hr/msd-prirucnik/pulmologija/pneumonija/izvanbolnicke-pneumonije. [10.8.2021.]

SAŽETAK

U ovom diplomskom radu ostvaren je web sustav koji omogućuje preporuku prikladnih lijekova s obzirom na postojeće parametre pacijenta i lijekova. Opisani su pristupi u odabiru prikladnih lijekova na temelju podataka i tehnologija koje će se koristiti te problematike stvaranja preporuka prikladnih lijekova. Korištenjem višekriterijskog filtriranja omogućeno je stvaranje preporuka prema većem broju podataka vezanih za pacijenta. Poslužiteljska i klijentska strana zasnovane su na komponentama, odnosno modulima koji svaki zasebno predstavljaju neku od funkcionalnosti sustava. Poslužiteljska strana programski je ostvarena u razvojnom okviru NestJS, klijentska strana korištenjem biblioteke ReactJS, baza podataka ostvarena je u sustavu PostgreSQL, a web sustav je postavljen na udaljeni poslužitelj Heroku. Alati koji su korišteni u izradi su Visual Studio Code, Insomnia, DBeaver, JMeter, GitHub i drugi. Testiranje ostvarenog sustava obavljeno je na razini korisničkog sučelja, stres testom i testiranjem API-ja. Također, ispitivanje web sustava za više slučajeva korištenja pokazalo je da su ostvareni funkcionalni i nefunkcionalni zahtjevi na sustav.

Ključne riječi: odabir prikladnog lijeka, sustav stvaranja preporuka, testiranje programske podrške, višekriterijsko donošenje odluka, web sustav.

ABSTRACT

Development and Testing of Web System for Creating Recommendations of Suitable Medicines

In this master's thesis, a web system has been realized that enables the recommendation of suitable medications with regard to the existing parameters of the patient and medications. Approaches to selecting appropriate drugs based on the data and technologies to be used and the issues of making recommendations for appropriate drugs are described. The use of multi-criteria filtering has enabled the creation of recommendations according to a larger number of data related to the patient. The server and client side are based on components, i.e. modules, each of which represents one of the functionalities of the system. The server side is programmatically implemented in the NestJS development framework, the client side using the ReactJS library, the database is implemented in the PostgreSQL system, and the web system is set up on the remote Heroku server. The tools used in the development are Visual Studio Code, Insomnia, DBeaver, JMeter, GitHub and others. Testing of the realized system was performed at the user interface level, stress test and API testing. Also, testing of the web system for multiple use cases showed that functional and non-functional system requirements were met.

Keywords: selection of suitable drug, recommender systems, software testing, multicriteria decision making, web system.

ŽIVOTOPIS

Josip Papež rođen je 1996. godine u Osijeku. Pohađao je Osnovnu školu Jagode Truhelke u Osijeku. Srednju Elektrotehničku i prometnu školu završava 2015. godine te upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer Računarstvo. U akademskoj godini 2019./2020. upisuje diplomski studij, smjer Programsko inženjerstvo, na istom fakultetu. Tijekom 2019. godine počinje raditi kao *front-end developer*.

PRILOZI

PRILOG 1. Dokument diplomskog rada

PRILOG 2. Diplomski rad u PDF formatu

PRILOG 3. Programski kod web aplikacije za stvaranje preporuka prikladnih lijekova