Razvoj i testiranje web aplikacije sa sustavom stvaranja preporuka za potporu liječenju alergijskih bolesti

Staković, Ivan

Master's thesis / Diplomski rad

2023

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:200:961981

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-07-14**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Diplomski studij

RAZVOJ I TESTIRANJE WEB APLIKACIJE SA SUSTAVOM STVARANJA PREPORUKA ZA POTPORU LIJEČENJU ALERGIJSKIH BOLESTI

Diplomski rad

Ivan Staković

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 25.08.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Ivan Staković
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1244R, 08.10.2021.
OIB studenta:	51056747040
Mentor:	prof. dr. sc. Goran Martinović
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	prof. dr. sc. Goran Martinović
Član Povjerenstva 2:	izv. prof. dr. sc. Ivica Lukić
Naslov diplomskog rada:	Razvoj i testiranje web aplikacije sa sustavom stvaranja preporuka za potporu liječenju alergijskih bolesti
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U teorijskom dijelu diplomskog rada treba opisati probleme i izazove pri liječenju alergijskih bolesti s naglaskom na peludne alergije, alergije na hranu i na inhalatorne alergene. Na temelju analize postojećih sličnih rješenja i korištenih postupaka stvaranja preporuka, potrebno je definirati funkcionalne i nefunkcionalne zahtjeve na web sustav, predložiti model i arhitekturu web sustava temeljenu na predlošku MVC na strani korisnika i na strani poslužitelja, te prikladan višekriterijski postupak stvaranja preporuka i testiranja sustava. Web aplikacija treba liječniku i/ili pacijentu omogućiti stvaranje profila pacijenta, unos i pohranu simptoma i pokazatelja bolesti, postupak dijagnosticiranja bolesti, preporuke tijeka liječenja i prepisivanja terapije na temelju indikacija, kontraindikacija, nuspojava i uspješnosti liječenja. Koristeći programski jezik Java, razvojne okvire Spring i Bootstrap, web aplikaciju s bazom podataka treba programski ostvariti, obaviti API testiranje, te ispitivanje i analizu programskog rješenja za različite alergije i slučajeve korištenja. Tema rezervirana za studenta: Ivan Staković
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 od strane mentora:	25.08.2023.

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

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 08.09.2023.

Ime i prezime studenta:	Ivan Staković
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1244R, 08.10.2021.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Razvoj i testiranje web aplikacije sa sustavom stvaranja preporuka za potporu liječenju alergijskih bolesti**

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. IZAZOVI OBUHVAĆENIH ALERGIJA I POSTOJEĆA RJEŠENJA ... 2

2.1. Općenito o alergijama ... 2

2.2. Peludne alergije ... 2

2.2.1. Karakteristike peludnih alergija ... 3

2.2.2. Dijagnoza i terapijski pristupi ... 3

2.2.3. Tipovi peludnih alergija ... 4

2.3. Alergije na hranu ... 6

2.3.1. Uzroci i simptomi alergija na hranu ... 6

2.3.2. Dijagnostički postupci i terapijske strategije ... 6

2.4. Inhalatorne alergije ... 7

2.4.1. Identifikacija i karakterizacija inhalatornih alergena ... 7

2.4.2. Metode dijagnosticiranja i liječenja inhalatornih alergija ... 9

2.5. Prikaz stanja u području ... 9

2.6. Postojeća slična rješenja ... 10

3. ZAHTJEVI, MODEL I GRAĐA WEB APLIKACIJE SA SUSTAVOM PREPORUKA ... 15

3.1. Funkcionalni zahtjevi ... 15

3.2. Nefunkcionalni zahtjevi ... 17

3.3. Korišteni postupci za stvaranje preporuka ... 18

3.3.1. Primjena višekriterijskog postupka ... 18

3.3.2. Definiranje kriterija za preporuke ... 18

3.4. Građa programskog rješenja ... 20

3.4.1. Predložak MVC (engl. *Model View Controller*) ... 20

3.4.2. Arhitektura sustava na strani korisnika ... 22

3.4.3. Arhitektura sustava na strani poslužitelja ... 22

3.5. Osnove API testiranja ... 26

4. PROGRAMSKO RJEŠENJE WEB APLIKACIJE ... 28

4.1. Korišteni programski jezici i alati ... 28

4.1.1. Korišteni programski jezici ... 28

4.1.2. Korišteni razvojni alati ... 28

4.1.3. Integracija s bazom podataka ... 30

4.2. Programsko ostvarenje web aplikacije ... 32

4.2.1. Naslovna stranica ... 32
4.2.2. Mogućnosti prijavljenog pacijenta ... 38
4.2.3. Mogućnosti prijavljenog liječnika ... 58
4.2.4. Mogućnosti administrator korisnika ... 70
4.2.5. Postojeće uloge unutar aplikacije ... 73
4.2.6. API testiranje ... 75
5. PRIKAZ NAČINA RADA, ISPITIVANJE I TESTIRANJE WEB APLIKACIJE ... 80
5.1. Način rada web aplikacije ... 80
5.1.1. Pacijent kao korisnik ... 80
5.1.2. Liječnik kao korisnik ... 84
5.1.3. Admin kao korisnik ... 86
5.2. Ispitivanje web aplikacije ... 89
5.2.1. Primjer korištenja 1 ... 90
5.2.2. Primjer korištenja 2 ... 91
5.2.3. Primjer korištenja 3 ... 92
5.3. Prikaz rješenja API testiranjem ... 94
6. ZAKLJUČAK ... 97
7. LITERATURA ... 98
SAŽETAK ... 101
ABSTRACT ... 102
ŽIVOTOPIS ... 103
PRILOZI ... 104

1. UVOD

Alergijske bolesti predstavljaju složen problem s ozbiljnim posljedicama za zdravstvene sustave i pojedince diljem svijeta. Alergijske reakcije, poput peludnih alergija, alergija na hranu i inhalacijskih alergija, postaju češće i znatno utječu na fizičko i emocionalno stanje ljudi. Na primjer, peludne alergije uzrokuju sezonske simptome poput kihanja, curenja nosa te svrbeža očiju i grla, što ometa i produktivnost. Alergije na hranu mogu izazvati različite reakcije, uključujući osip, oticanje usne šupljine, gastrointestinalne smetnje i u težim slučajevima anafilaktički šok. Inhalacijske alergije, kao što su reakcije na prašinu i grinje, dovode do problema s dišnim sustavom poput kašlja, otežanog disanja i čestih infekcija dišnih puteva. Unatoč napretku medicinskih istraživanja, dijagnosticiranje i liječenje alergijskih bolesti i dalje su izazovni. Identifikacija specifičnih alergena koji uzrokuju simptome zahtijeva pažljivo praćenje i analizu. Stoga je ključno pružiti prilagođene pristupe upravljanju alergijama, budući da reakcije variraju među pojedincima. Prepoznavanje simptoma alergija i pravodobno pružanje preporuka za liječenje od vitalnog su značaja kako bi se postigli najbolji mogući rezultati za svakog pacijenta. Sveukupno, suočavanje s izazovima alergijskih bolesti zahtijeva sveobuhvatan pristup koji uzima u obzir različite simptome i individualne potrebe.

Glavni cilj ovog diplomskog rada jest temeljito istražiti različite aspekte peludnih alergija, alergija na hranu i inhalatornih alergija. Također, razvit će se model preporuka za dijagnosticiranje alergija na temelju simptoma. Uz to, rad se bavi analizom primjene odgovarajućih lijekova za već dijagnosticirane alergije. Kroz temeljiti pregled relevantne literature, analizu dostupnih podataka i primjenu suvremenih metoda, cilj je povećati razumijevanje alergijskih bolesti i pojednostaviti proces dijagnosticiranja i liječenja. Ovakav pristup ima mogućnost poboljšati kvalitetu života pacijenata te umanjiti utjecaj alergijskih bolesti na njihovo opće zdravstveno stanje.

U drugom poglavlju bit će prikazani izazovi povezani s alergijama koje će biti obuhvaćene u radu te postojeća slična rješenja. Treće poglavlje će detaljno prikazati zahtjeve, model i strukturu web aplikacije, uključujući objašnjenje korištene arhitekture te funkcionalnih i nefunkcionalnih zahtjeva na web sustav. Četvrto poglavlje daje pregled korištenih programske jezike i alata, zajedno s opisom programskog rješenja web aplikacije. Peto poglavlje prikazuje način rada i ispitivanje web aplikacije, uključujući testiranje aplikacije za različite slučajeve upotrebe i ispitivanje ispravnosti za različite parametre korisničke interakcije.

2. IZAZOVI OBUHVAĆENIH ALERGIJA I POSTOJEĆA RJEŠENJA

U ovom poglavlju će se razmotriti izazovi vezani uz različite obuhvaćene alergije, istražujući njihove uzroke, simptome i utjecaj na život pacijenata. Također, analizirat će se postojeća medicinska rješenja i terapije koje se primjenjuju kako bi se ublažili simptomi alergija i poboljšala kvaliteta života obuhvaćenih osoba.

2.1. Općenito o alergijama

Alergija se razvija kao rezultat složene interakcije između genetskih faktora i vanjskih utjecaja. Predstavlja pretjeranu reakciju imunološkog sustava na tvari iz okoline koje obično nisu štetne, bilo da ih udahnemo, dodirnemo kožom, unesemo injekcijom ili konzumiramo. Kada organizam ponovno dođe u kontakt s tim tvarima, imunološki sustav reagira preosjetljivo, što rezultira interakcijom između alergena - stranih bjelančevina i protutijela - vlastitih bjelančevina, pri čemu se nepotrebno i prekomjerno proizvode protutijela. Po uzoru na [1], alergijske reakcije koje su potaknute tim protutijelima pripadaju skupini imunoglobulina E (eng. *Immunoglobulin E* (IgE)). Imunoglobulini se vežu na posebne receptore na mastocitima, koji sadrže tvar poznatu kao histamin. Kada se IgE veže na mastocite, dolazi do oslobađanja histamina, što rezultira proširenjem krvnih žila, povećanom cirkulacijom, kontrakcijom glatkih mišića i pojavom simptoma alergijske reakcije. Alergije se često razvijaju tijekom djetinjstva, ali mogu se pojaviti u bilo kojem životnom razdoblju. Procjenjuje se da u Hrvatskoj 7-10% stanovništva pati od peludne alergije, dok 3-5% boluje od astme.

Terapija je vrijedan resurs u preventivnom liječenju pacijenata s alergijama koje mogu biti opasne po život ili kod kojih se alergijska bolest ne odaziva na druge lijekove. Iako je u početku skupa, terapija može spriječiti alergijski odgovor tijekom mnogih godina i može spriječiti razvoj dodatnih alergijskih stanja. Stoga, šira primjena terapija potencijalno bi rezultirala značajnim dugoročnim uštedama za zdravstvene usluge [3].

2.2. Peludne alergije

Tijekom svakog proljeća, ljeta i jeseni, pa čak i zimi u određenim državama, biljke otpuštaju sitna peludna zrnca kako bi se oplodile s drugim biljkama iste vrste. Većina peludi koja izaziva alergijske reakcije potječe od drveća, trava i korova. Ove biljke proizvode malena, suha i lagana peludna zrnca koja se prenose zrakom. Ta zrnca mogu dospjeti u oči, nos i pluća, izazivajući simptome alergije kod osoba koje pate od peludne alergije.

2.2.1. Karakteristike peludnih alergija

Cvjetne biljke koje se oprašuju insektima, poput ruža i određenih vrsta stabala kao što su trešnje i kruške, obično ne uzrokuju peludnu alergiju. Postoje tri glavne vrste peludne alergije: alergija na pelud drveća, alergija na pelud trava i alergija na pelud korova [2].

2.2.2. Dijagnoza i terapijski pristupi

Prema [2], prilikom prvog koraka u dijagnosticiranju peludne alergije, liječnik će intervjuirati pacijenta o njihovim simptomima. Pitanja će se odnositi na mjesto i vrijeme javljanja simptoma, trajanje simptoma te obiteljsku anamnezu alergijskih bolesti. Nakon toga slijedi fizički pregled, a mogu se provesti i kožni i/ili krvni testovi radi potvrde alergije.

Ti testovi otkrivaju sklonost alergiji na određene peludne alergene, ali ne pružaju konačan dokaz o postojanju alergije. Stoga, ako postoji sumnja, može biti potreban provokacijski test. Za taj test se na sluznicu nosa pacijenta pod medicinskim nadzorom nanose male količine sumnjivog peludnog alergena. Pacijent se zatim promatra radi reakcija kao što su oticanje nosa, kihanje ili suzenje očiju. Ako pacijent ima respiratorne simptome, provodi se test plućne funkcije. Liječnik će mjeriti količinu zraka koju pacijent udiše i izdiše te brzinu izdisaja kako bi se utvrdile moguće promjene na dišnim putevima na temelju rezultata testa. Tablica 2.1 prikazuje popis korištenih simptoma peludnih alergija.

Tablica 2.1. *Prikaz simptoma peludnih alergija* [3]

Simptomi
Začepljen nos
Curenje nosa
Crvenilo i svrbež očiju
Zaplakane oči
Kihanje
Glavobolja
Svrbež nosa
Bol u licu
Bol u uhu
Začepljen nos
Upala sluznice i sinusa
Astma
Suhi kašalj
Otežano disanje
Stezanje u prsima
Osip
Umor
Frustracija
Poremećaji sna

Važno je isključiti alergije na grinje, životinjsku dlaku i plijesan, jer one također mogu uzrokovati simptome slične peludnoj groznici. Također, rinitis uzrokovan bakterijama ili virusima može se na prvi pogled zamijeniti s peludnom alergijom. Međutim, prehlada obično traje samo oko deset dana, uz zelenu i viskoznu sluznicu nosa, dok peludna alergija uzrokuje vodenaste simptome i svrbež očiju. Groznica i otečeni limfni čvorovi u grlu su također znakovi prehlade.

2.2.3. Tipovi peludnih alergija

Prvi tip peludne alergije je alergija na pelud drveća. Drveće ima značajnu ulogu u uzroku peludnih alergija. To je djelomično zbog njihove široke rasprostranjenosti i gustoće u ljudskom okruženju, što vjerojatno čini većinu biljaka koje se uzgajaju zbog proizvodnje peludi diljem svijeta. Kroz povijest su drveće smatrana prekrasnim, a u nekim slučajevima čak i svetim. Ona imaju važnu ulogu u ljudskoj kulturi, pružajući ne samo hranu, već i materijal za gradnju skloništa, gorivo i alate. Izrazito urbanizirani gradovi i siromašna, ekološki degradirana ruralna naselja ili urbane sirotinjske četvrti u razvijenom svijetu smatraju se nezdravima djelomično zbog nedostatka drveća [4].

Kada je riječ o alergenom drveću, sezona peludi može započeti rano u proljeće s cvjetanjem drveća poput breza, čak prije nego što počne sezona peludi trava i korova. Međutim, neka drveća cvjetaju u jesen ili čak dva puta godišnje, što može stvoriti nejasnoće u vezi s izlaganjem peludi. Tipični simptomi alergije na pelud drveća uključuju rinitis, astmu i konjunktivitis. Slično kao i kod alergije na pelud trava i korova, često se toleriraju nelagoda i ne provode odgovarajuće dijagnostičke i terapijske strategije. Prema [4], umjesto toga, neki se odlučuju za izostanak liječenja ili čak primjenjuju neprikladnu dugotrajnu terapiju. No, zbog utjecaja peludi drveća na astmu iznimno je važno da se alergija na pelud drveća klinički pravilno dijagnosticira i tretira.
Iduća alergija koja pripada peludnim alergijama je alergija na pelud trava. Ovisno o klimi i geografskom području, peludi trava predstavljaju glavne izvore alergena u zraku tijekom proljeća i ljeta. One se javljaju na svim kontinentima i obuhvaćaju velik dio vegetacije na Zemlji. Simptome astme tijekom toplih godišnjih doba u umjerenim klimatskim područjima doživljava barem polovica osoba koje su osjetljive na pelud travki. Pozitivni rezultati kožnih uboda i prisutnost specifičnih IgE protutijela u krvnim testovima na peludne pripravke ukazuju na alergijsku osjetljivost, ali su klinički relevantni samo ako se javljaju odgovarajući simptomi. Imuno terapija peludnih alergija najučinkovitija je kada se primjenjuje tijekom najmanje tri godine putem potkožnih injekcija ili putem kapi ili tableta koje sadrže pripravke jedne vrste trave ili mješavine trava [5].

Posljednje su alergije na pelud korova. Iako postoji ogroman broj vrsta korova, njihova uloga u peludnoj alergiji je ograničena, ali iznimno važna. Većina korova nije oprašivana vjetrom te stoga ne proizvodi male, lagane peludne čestice koje se lako prenose zrakom i uzrokuju alergijske reakcije. Dodatno, korovi su često žrtve ljudskih aktivnosti usmjerenih na kontrolu okoliša. Budući da nemaju značajan utjecaj na ljudsku kulturu i materijalno blagostanje, tradicionalno se nastoji suzbiti njihov rast i širenje, posebno u poljoprivredi. Njihova upotreba kao ukrasnih biljaka ili hrane pruža samo djelomičan balans. Korovi se najčešće nalaze na neplodnom tlu i pašnjacima te igraju važnu ulogu u prirodnom procesu ponovnog pošumljavanja kao tzv. sukcesijske biljke. No, mnoge vrste korova se suočavaju s poteškoćama u suprotstavljanju ljudskoj civilizaciji. Unatoč tome, nekoliko korova ima ključnu ulogu u sezonskom sindromu alergijskog rinitisa, konjunktivitisa, kihanja, začepljenja nosa i astme. Iako ovi čimbenici mogu izgledati zastrašujuće, svaki slučaj alergije na pelud korova zahtijeva pažljiv i sustavan pristup. Iako alergijski rinitis rijetko predstavlja ozbiljnu prijetnju životu, ne bi ga trebalo zanemariti niti tretirati samo simptomatskim lijekovima prije nego što se identificiraju i izbjegnu alergeni u pitanju. Alergija na pelud korova ima izrazito iscrpljujuće učinke, s velikim brojem izgubljenih dana u školi i na poslu te s visokim društvenim i ekonomskim troškovima. Stoga je važno pridavati odgovarajuću pažnju alergiji na pelud korova zbog česte povezanosti s alergijskom astmom i križnim reaktivnostima. Sezona alergija za pelud korova obično se javlja kasnije od sezone peludi trave i drveća, uglavnom u sredini ljeta do kasne jeseni. [6]. Korištene peludne alergije i simptomi unutar rada prikazane su tablicom 2.2.

Tablica 2.2. *Prikaz korištenih peludnih alergija i simptoma*

Naziv alergije	**Popis simptoma**
Peludna alergija	Svrbež nosa - očiju - ušiju i usta, curenje iz nosa, začepljen nos, crvene i suzne oči, oticanje oko očiju, umor
Alergija na travu	Curenje nosa, začepljen nos, kihanje, svrbež nosa i očiju, iritacija grla, glavobolja
Alergija na korov	Začepljen nos, curenje iz nosa, kihanje, svrbež nosa, naknadno otjecanje sluzi (osjećaj da sluz teče niz stražnji dio grla).
Alergija na drveće	Kihanje, svrbež nosa - očiju - ušiju i usta, crvene i suzne oči, oticanje oko očiju, glavobolja
Sezonske alergije	svrbež, curenje iz nosa, začepljen nos, privremeni gubitak mirisa, glavobolja, bol u licu

2.3. Alergije na hranu

Za dijagnozu alergije na hranu potrebno je detaljno prikupiti povijest alergije i provesti testove poput kožnog testa, određivanja specifične molekularne alergologije (poznate i kao dijagnostika s komponentama) kako bi se otkrila osjetljivost IgE-om te razlikovala od drugih stanja povezanih s hranom, kao što su intolerancija na laktozu ili reakcije uzrokovane farmakološkim čimbenicima poput histamina u rajčicama ili Amina u siru. Za potvrdu dijagnoze alergije na hranu može biti potreban test izazova hrane koji se provodi u medicinskoj instituciji [7].

2.3.1. Uzroci i simptomi alergija na hranu

Simptomi alergije na hranu mogu se manifestirati u različitim dijelovima tijela istovremeno. Česti simptomi alergije na hranu uključuju [8]:

- Osjećaj vrtoglavice ili slabosti
- Svrbež kože ili pojava osipa
- Oticanje usana, lica i očiju
- Kašljanje, zviždanje u prsima, otežano disanje, glasan ili promukao glas
- Kihanje ili svrbež, curenje ili začepljenje nosa
- Osjećaj mučnine ili povraćanje
- Bolovi u trbuhu
- Proljev

2.3.2. Dijagnostički postupci i terapijske strategije

Ukoliko pacijent ima alergiju na hranu, neće moći konzumirati namirnice na koje je alergičan, uključujući one koje sadrže sastojke na koje je alergičan. Propisat će se lijekovi koji će pomoći u kontroliranju simptoma ili koje će se koristiti u hitnim situacijama. Ovi lijekovi mogu uključivati:

- Antihistaminike za ublažavanje blagih alergijskih reakcija.
- Hitne lijekove poput adrenalinskih injekcija, kao što je EpiPen, za teže alergijske reakcije.

Specijalist će izraditi plan upravljanja alergijom koji će vam objasniti kako pravilno upravljati svojom alergijom. Djeca koja imaju alergiju na kikiriki mogu proći imuno terapiju kako bi smanjila osjetljivost na kikiriki, ali i dalje bi trebala izbjegavati konzumaciju kikirikija [8]. Korištene alergije hrane i njeni simptomi unutar rada prikazani su tablicom 2.3.

Tablica 2.3. *Prikaz korištenih alergija hrane te njihovi simptomi*

Naziv alergije	Popis simptoma
Alergija na kikiriki	povraćanje, grčevi u želucu, probavne smetnje, proljev, disanje sa čudnim zvukom iz pluća, nedostatak zraka, ponavljajući kašalj, stegnutost u grlu, slab puls, blijeda ili plavkasta boja kože, vrtoglavica, zbunjenost
Alergija na orašaste plodove	otežano disanje, disanje sa čudnim zvukom iz pluća, oteklina jezika, oteklina grla, poteškoće u govoru, zbunjenost, blijedost i slabost
Alergija na mlijeko	urtikarija, disanje sa čudnim zvukom iz pluća, povraćanje, proljev, nadutost, curenje nosa, ekcem
Alergija na jaja	urtikarija, osip, začepljenje nosa, bol u trbuhu, mučnina, povraćanje, simptomi astme, oteklina usana, oteklina jezika, oteklina lica
Alergija na soju	urtikarija, svrbež, bol u trbuhu, proljev, povraćanje, oteklina, začepljenje nosa, disanje sa čudnim zvukom iz pluća, poteškoće s disanjem
Alergija na pšenicu	oticanje, svrbež, urtikarija, ekcem, bol u trbuhu, proljev, mučnina, povraćanje, poteškoće s disanjem
Alergija na ribu	urtikarija, oticanje, svrbež, bol u trbuhu, mučnina, povraćanje, proljev, začepljenje nosa, poteškoće s disanjem
Alergija na školjke	oticanje, svrbež, urtikarija, ekcem, bol u trbuhu, mučnina, povraćanje, proljev, začepljenje nosa, poteškoće s disanjem
Alergija na sezam	urtikarija, svrbež, crvenilo, bol u trbuhu, povraćanje, proljev, začepljenje nosa, poteškoće s disanjem
Alergija na sulfate (konzervans u hrani i piću)	urtikarija, crvenilo, svrbež, bol u trbuhu, proljev, poteškoće s disanjem, disanje sa čudnim zvukom iz pluća, simptomi astme

2.4. Inhalatorne alergije

Inhalacijske alergije predstavljaju tvari prisutne u zraku koje se udišu. Ova kategorija uključuje alergene koji mogu uzrokovati reakcije tijekom svih godišnjih doba i one koji su sezonski. Manifestacije inhalacijskih alergija obično obuhvaćaju simptome poput curenja nosa.

2.4.1. Identifikacija i karakterizacija inhalatornih alergena

Prema uzoru na [27], alergijske reakcije koje se javljaju udisanjem mogu biti uzrokovane različitim tvarima koje se prenose zrakom, kako unutar prostora tako i na otvorenom. Sezonske alergije su jedan od najčešćih oblika inhalacijskih alergija. Osim toga, i onečišćenje zraka može izazvati simptome alergije. Tipični alergeni na otvorenom uključuju:

- Pelud (trava, drveće, korov)
- Spore plijesni
- Gljivice

- Dim
- Smog

Mnogi inhalacijski alergeni su povezani s unutarnjim prostorom, kao što su:

- Perut kućnih ljubimaca (npr. psi, mačke, konji, zamorci)
- Grinje
- Plijesan
- Mirisi
- Kućni kemikalije
- Dim svijeća

Inhalacijske alergije mogu biti izazvane i hlapljivim organskim spojevima (VOC), koji se oslobađaju kao plinovi. Takve kemikalije mogu se naći u različitim kućanskim proizvodima, uključujući:

- Sredstva za čišćenje
- Sprejevi protiv insekata
- Proizvodi za njegu automobila
- Gorivo
- Kemijsko čišćenje odjeće
- Tepisi

Uobičajeni simptomi uključuju:

- Curenje nosa
- Začepljenost nosa
- Svrbež u očima, nosu, ustima i grlu
- Pritisak u sinusima
- Glavobolja
- Gubitak osjeta mirisa
- Osip
- Kihanje
- Kašljanje
- Pritisak ili osjećaj punine u ušima
- Otečene, crvene, suzne oči

2.4.2. Metode dijagnosticiranja i liječenja inhalatornih alergija

Prema [9], postoje različiti tretmani dostupni za inhalacijske alergije, čak i ako ne znate točno što ih uzrokuje. Smanjenje simptoma može se postići uz pomoć lijekova i prirodnih kućnih lijekova. Uobičajeni lijekovi za alergije uključuju sljedeće:

- Antihistaminici: Ovi lijekovi blokiraju djelovanje histamina, tvari koja se oslobađa iz imunološkog sustava i uzrokuje simptome alergije.
- Kortikosteroidi: U slučaju ozbiljnijih simptoma alergije, može biti potrebno primjenjivanje steroidnih protuupalnih lijekova.
- Alergenske injekcije: Male doze alergena postupno se uvode u tijelo kako bi se smanjila osjetljivost. Ovo može biti posebno korisno kod inhalacijskih alergija na prašinu, pelud i perut kućnih ljubimaca

Korištene inhalatorne alergije i njeni simptomi prikazani su tablicom 2.4.

Tablica 2.4. *Prikaz korištenih inhalatornih alergija i njenih simptoma*

Naziv alergije	Popis simptoma
Alergija na prašinu	kihanje, curenje nosa, crvene ili vodene oči, začepljenost nosa, svrbež nosa - očiju - ušiju i usta, stvaranje sluzi u grlu, ponavljajući kašalj, osjećaj pritiska i boli u licu
Alergija na kućne ljubimce	kihanje, curenje nosa, začepljen nos, osjećaj pritiska i boli u licu, ponavljajući kašalj, urtikarija, osip
Alergija na plijesan	začepljen nos, curenje iz nosa, kihanje, nadražene oči, ponavljajući kašalj, piskanje pri disanju, svrbež grla
Alergija na kemijske tvari	Povećan puls, bol u prsima, znojenje, poteškoće pri disanju, umor, vrtoglavica

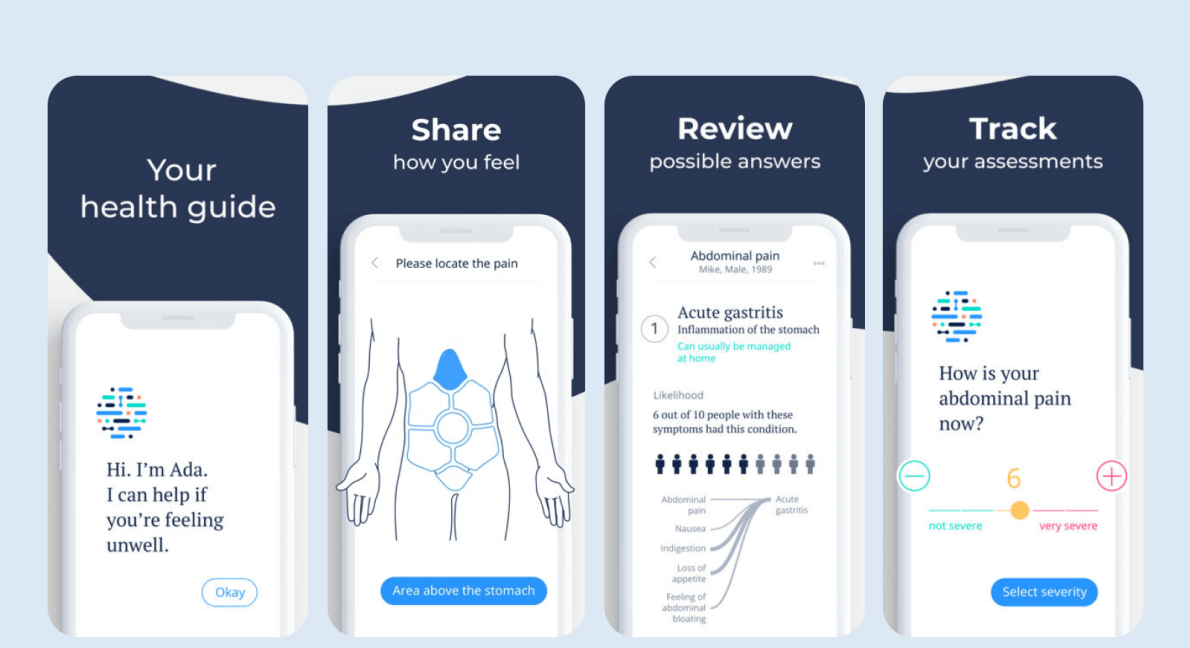
2.5. Prikaz stanja u području

U modernom svijetu, informacijsko-komunikacijska tehnologija (ICT), programsko inženjerstvo i metode učenja igraju ključnu ulogu u sučeljavanju s raznovrsnim medicinskim izazovima, uključujući probleme vezane uz alergije i lijekove. U ovom odjeljku, istražuju se neke od tih izazova i ističu kako drugi pristupaju sličnim pitanjima putem tehnoloških i znanstvenih rješenja. Sastavni izazovi u ovom kontekstu uključuju složenu prirodu medicinskih podataka, njihovu integraciju te potrebu za personalizacijom terapija. Osobito je važno osigurati sigurnost i zaštitu privatnosti osjetljivih medicinskih informacija. Rješenja koja su već u primjeni uključuju korištenje analize velikih skupova podataka kako bi se identificirali oblici i poveznice među alergijskim reakcijama i određenim lijekovima. Prema [25], prediktivna analitika koja se oslanja na napredne algoritme strojnog učenja, omogućuje predviđanje potencijalnih alergijskih reakcija za pojedine pacijente na temelju njihove

povijesti liječenja i genetskog profila. Nadalje, personalizirane preporuke za terapiju postaju sve više dostupne zahvaljujući aplikacijama koje koriste tehnike umjetne inteligencije. Takve aplikacije uzimaju u obzir individualne alergijske reakcije, osjetljivosti i povijest pacijenta kako bi pružile relevantne i prilagođene savjete o lijekovima. Osim toga, razvoj tehnologija omogućuje pacijentima da se konzultiraju s medicinskim stručnjacima putem internetskih platformi, omogućavajući brze konzultacije o alergijama i terapijama bez potrebe za putovanjem do ordinacije. Sve u svemu, kombinacija ICT-a, programskog inženjerstva i naprednih metoda učenja značajno unapređuje pristup rješavanju problema alergija i lijekova. Integracija ovih tehnologija omogućava personaliziranu medicinsku terapiju i doprinosi boljoj kvaliteti zdravstvene skrbi pacijenata. Iako su najmoderniji računalni alati napredni, imaju svoje ograničenosti u obradi velikih količina podataka. Prema uzoru na [26], uobičajeni analitički pristupi u medicini obuhvaćaju tradicionalnu statistiku, mrežnu analizu, prostornu analizu, grupiranje i strojno učenje. Svatko od ovih pristupa ima svoju sposobnost za rješavanje različitih vrsta pitanja, ali također donosi svoja vlastita ograničenja.

2.6. Postojeća slična rješenja

Jedno od sličnih rješenja koje odgovara tematici aplikacije je Ada. To je platforma za zdravlje koja koristi umjetnu inteligenciju kako bi korisnicima pružila personalizirane procjene zdravlja i informacije. S analizom zdravstvenih podataka i simptoma koje korisnik pruža, generira moguća objašnjenja za simptome te nudi relevantne zdravstvene savjete. Glavni je cilj pomaganje korisnicima da bolje razumiju svoje zdravstveno stanje i preporučuje odgovarajuću medicinsku skrb ako je potrebno. Slika 2.1 prikazuje izgled aplikacije Ada.

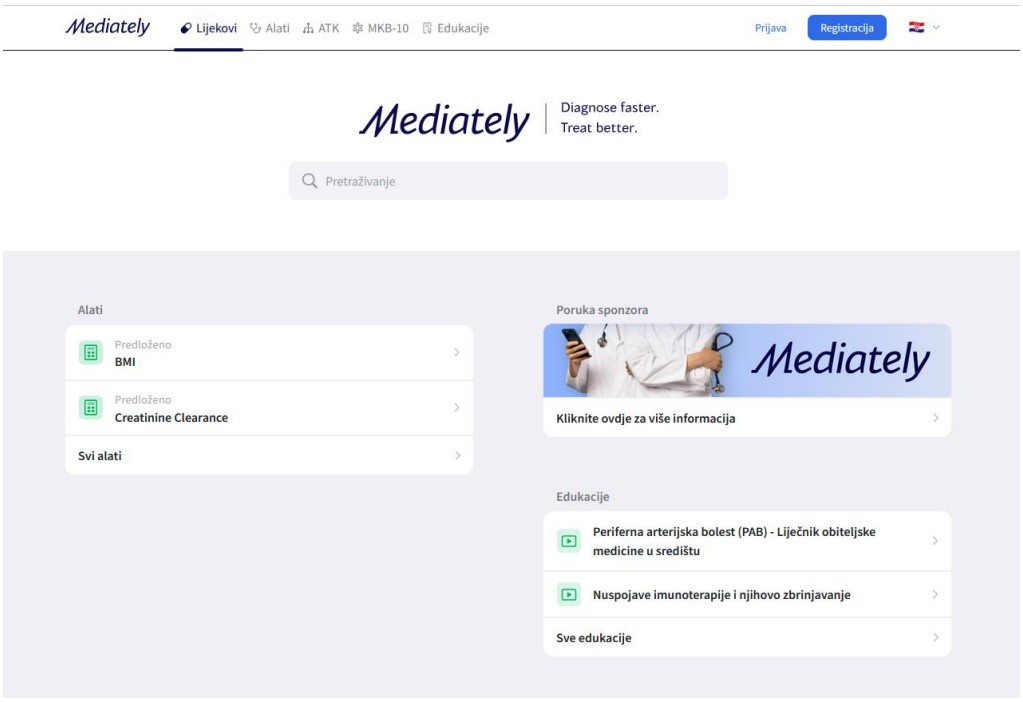


Slika 2.1. *Prikaz sadržaja aplikacije Ada* [10]

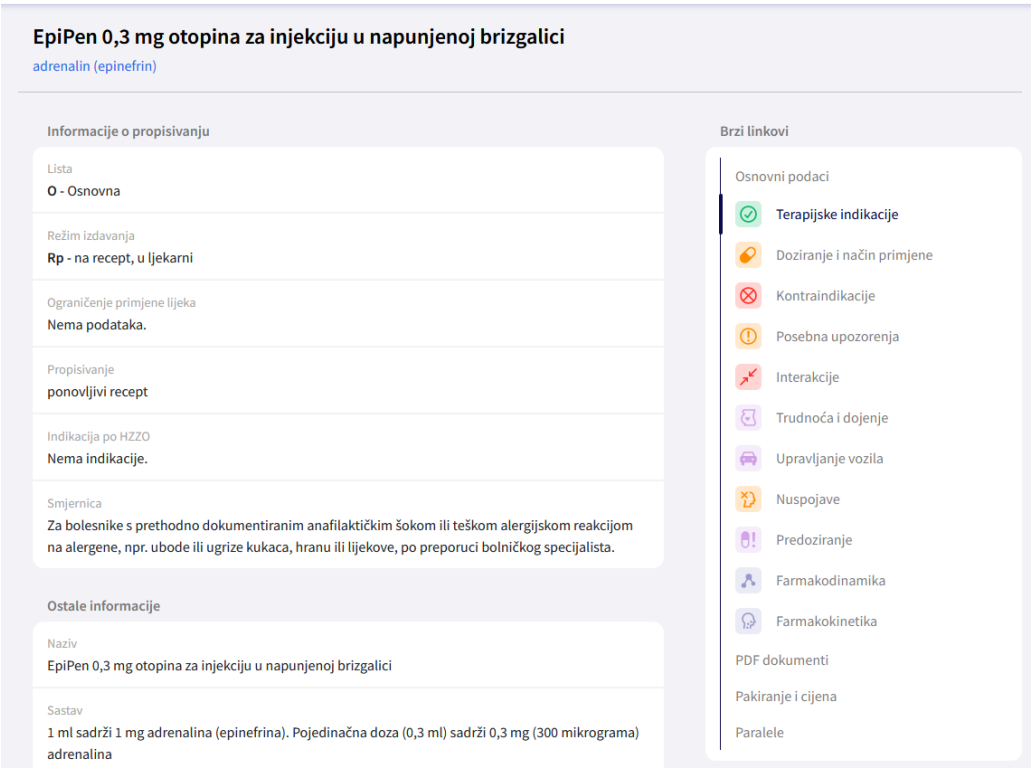
Ada također može na temelju unesenih simptoma, prepoznati alergiju i preporučiti određeni lijek koji bi se u toj situaciji trebao koristiti. Rezultat aplikacije nije/ne može biti oslonac jer svako konzumiranje nekog lijeka ili terapije zahtjeva odobrenje liječnika. Ada, kao i svi drugi programi su tu kako bi liječniku, a i pacijentu olakšali prepoznavanje i tijek liječenja. Ovo su neki od današnjih sustava koji se koriste za odabir lijekova:

- Mediately – baza lijekova.
- Medscape – sustav odabira lijeka na temelju simptoma.

Na slici 2.2 prikazana je početna web stranica Mediately baze lijekova. Unutar baze su sadržani svi lijekovi koji su odobreni/propisani od strane zdravstvenih institucija. Za svaki od traženih lijekova pruža se obilje informacija koje opisuju uvjete konzumacije i doziranje. Također, svakom lijeku je pridružena PDF dokumentacija koja je dostupna za preuzimanje, uz prikaz cijene lijeka, indikacija, kontraindikacija i interakcija s drugim tvarima te ostalog korisnog sadržaja. Na slici 2.3 prikazan je pronađeni lijek EpiPen i sva pripadajuća dokumentacija koja je dostupna.

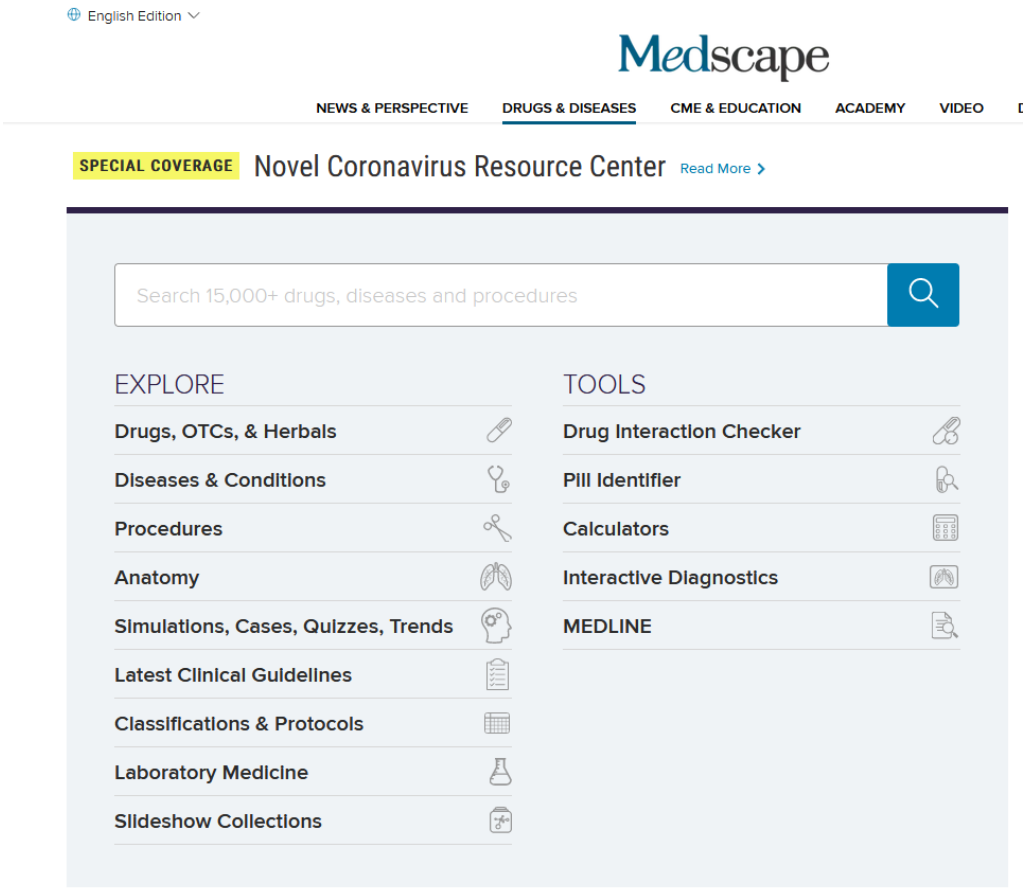


Slika 2.2. *Prikaz početne web stranice Mediately [11]*



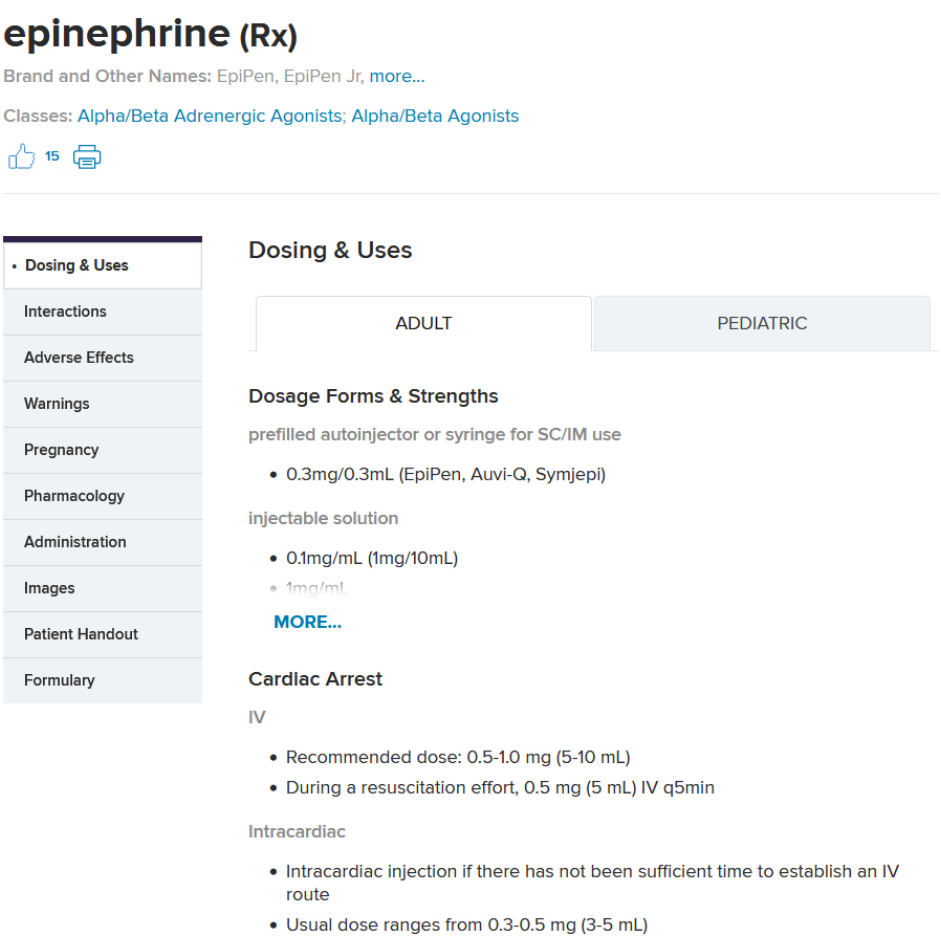
Slika 2.3. *Prikaz EpiPen dokumentacije lijeka [12]*

S druge strane, web aplikacija Medscape je glavno internetsko odredište za liječnike i zdravstvene stručnjake širom svijeta, pružajući najnovije medicinske vijesti i stručna stajališta, ključne informacije o lijekovima i bolestima koje su važne u skrbi pacijenata te relevantno stručno obrazovanje i CME (engl. *Continuous Medicine Education*). Slika 2.4 prikazuje izgled Medscape web aplikacije.



Slika 2.4. *Prikaz Medscape web aplikacije [13]*

Unutar aplikacije Medscape moguće je pretražiti/odabrati bolest te dobiti tijek liječenja za traženu bolest i sve postojeće lijekove koji se nude za tu bolest. U bazi lijekova Mediately omogućeno je pregledavanje pojedinih lijekova, njihovih detalja i promjena, interakcija s drugim lijekovima i tvarima, nuspojava te raznih drugih alata vezanih za lijekove, uključujući različite kalkulatore i slično. Međutim, nisu dostupni podaci o načinima liječenja određenih bolesti. Slika 2.5 prikazuje opis EpiEen lijeka u Medscape bazi lijekova.



Slika 2.5. *Prikaz EpiPen lijeka u Medscape bazi lijekova* [14]

3. ZAHTJEVI, MODEL I GRAĐA WEB APLIKACIJE SA SUSTAVOM PREPORUKA

Da bi se lakše shvatilo kako treba raditi web aplikacija, potrebno je definirati pravila na kojima će se sustav zasnivati, npr. način preporuke alergije te preporuka lijeka za preporučenu alergiju. Bitno je imati dobar model i strukturu, jer ako su ti parametri dobro definirani može se spriječiti implementaciju nekih nebitnih funkcionalnosti, te se usredotočiti na one više bitne.

3.1. Funkcionalni zahtjevi

Prema [15], funkcionalni zahtjevi su skup značajki ili funkcija proizvoda koje je potrebno implementirati kako bi korisnici mogli uspješno obavljati svoje zadatke. Njihova jasnoća i razumljivost su od velike važnosti kako za razvojni tim tako i za sve zainteresirane strane. Ovi zahtjevi detaljno opisuju kako se sustav treba ponašati u određenim situacijama.

Funkcionalni zahtjevi unutar web sustava koji su zadovoljeni su:

- Prijava i registracija pacijenta
- Spremanje prijavljenog pacijenta unutar kolačića
- Istekom trajanja kolačića odjava pacijenta
- Kreiranje korisnika s različitim ulogama (engl. *roles*)
- Posebno korisničko sučelje ovisno o dodijeljenim ulogama
- Kreiranje admin korisnik koji ima mogućnost brisanja i uređivanja korisnika
- Korisnik s ulogom admin kreira, uređuje i briše liječnika u web sustava
- Unos simptoma pacijenta
- Unos osobnih podataka pacijenta
- Preporuka alergije na temelju unesenih simptoma
- Kontaktiranje liječnika i pacijenta elektroničkom poštom
- Unos i praćenje tijeka liječenja pacijenta
- Grafički prikaz tijeka liječenja pacijenta
- Preporuka lijeka na temelju dijagnosticirane alergije, godina, težine i trudnoće
- Potvrda korištenoga lijeka od liječnika
- Praćenje tijeka uzimanja lijekova
- Dohvaćanje PDF dokumenta preporučenog lijeka
- Mogućnost preuzimanja tijeka liječenja i tijeka uzimanja lijekova pacijenta
- Kreiranje sigurnosne kopije baze svakih 3 sata

- Vraćanje baze pomoću .sh skripte

Ovo je web sustav u obliku web aplikacije namijenjen za praćenje zdravstvenog stanja pacijenata, te potiče bolju komunikaciju i suradnju između svih sudionika - liječnika i pacijenata. Glavna svrha ovog sustava jest omogućiti jednostavan i učinkovit način obavljanja zdravstvenih zadataka te upravljanje terapijama, što obuhvaća kako pacijente tako i medicinsko osoblje. U okviru aplikacije, korisnici imaju mogućnost registracije i prijave u sustav. Nakon prijave, podaci o korisniku se pohranjuju u kolačićima kako bi se olakšala identifikacija prilikom ponovne prijave. Važno je napomenuti da se sustav automatski odjavljuje korisnika nakon isteka trajanja kolačića kako bi se osigurala sigurnost i privatnost korisničkih podataka. Jedna od ključnih značajki sustava jest mogućnost stvaranja korisnika s različitim ulogama, poput administratora, pacijenta i liječnika. Svaka uloga pristupa posebnom korisničkom sučelju s funkcijama prilagođenima njihovoj ulozi. Uloga administratora omogućuje upravljanje korisničkim računima, uključujući brisanje i uređivanje, dok uloga liječnika može nadgledati razvoj pacijentovog stanja. Unos simptoma pacijenata i njihovih osobnih podataka omogućuje precizno praćenje njihovog zdravstvenog stanja. Sustav također nudi mogućnost preporuke alergija na temelju unesenih simptoma kako bi se identificirali potencijalni alergeni kod pacijenata. Komunikacija između liječnika i pacijenata omogućena je putem elektroničke pošte, što značajno olakšava razmjenu informacija i uputa. Za praćenje tijeka liječenja pacijenata, koristi se grafički prikaz i pojedinačne preporuke lijekova za svakog pacijenta. Sustav analizira dijagnosticirane alergije, godine, težinu i trudnoću pacijenata kako bi pružio najprikladnije preporuke za lijekove. Potvrda korištenja lijekova od strane liječnika je od ključne važnosti kako bi se osiguralo pravilno i sigurno liječenje pacijenata. Prilikom odobravanja lijeka, liječnik uzima u obzir i kontraindikacije koje pacijent ima navedene te na temelju njih donosi konačnu odluku. Osim toga, sustav omogućuje praćenje tijeka uzimanja lijekova kako bi se osiguralo pridržavanje propisane terapije od strane pacijenata. Na kraju, aplikacija nudi mogućnost preuzimanja PDF dokumenta s opisom preporučenog lijekovima, što omogućuje liječnicima lakše propisivanje lijeka zbog svih informacija koje su na jednome mjestu. Također korisnik i liječnik imaju mogućnost preuzimanja tijeka liječenja i tijeka uzimanja lijekova u svrhu lakšega pregleda Excel datoteke na svome računalu.

S ovim sveobuhvatnim sustavom, svi funkcionalni zahtjevi su uspješno zadovoljeni, pružajući učinkovit i pouzdan alat za praćenje zdravstvenog stanja pacijenata. Sustav poboljšava suradnju između liječnika i pacijenata, pružajući personalizirane preporuke lijekova i terapija za svakog korisnika. Dodatna funkcionalnost koja je implementirana je da se svakih 3h tijekom rada aplikacije,

kreira sigurnosna kopija baze podataka. Ako bi se sustav srušio, dovoljno je pokrenuti kreiranu skriptu koja vraća zadnje stanje baze.

3.2. Nefunkcionalni zahtjevi

Prema [15], u području inženjerstva sustava i inženjerstva zahtjeva, nefunkcionalni zahtjevi su zahtjevi koji postavljaju kriterije za procjenu rada sustava, umjesto definiranja konkretnih ponašanja. Za razliku od funkcionalnih zahtjeva koji opisuju specifične funkcije ili ponašanja sustava, ne-funkcionalni zahtjevi fokusiraju se na aspekte koji nisu vezani uz funkcionalnosti, već na performanse, pouzdanost, sigurnost i slično. Plan za implementaciju funkcionalnih zahtjeva obično je detaljno opisan u dizajnu sustava, dok se plan za implementaciju ne-funkcionalnih zahtjeva obično obrađuje u arhitekturi sustava, s obzirom na njihovu važnost za arhitektonski dizajn. Potrebni nefunkcionalni zahtjevi su:

- Upotrebljivost
- Sigurnost
- Pouzdanost
- Performanse
- Dostupnost
- Skalabilnost

Na temelju [22], upotrebljivost se odnosi na jednostavnost korištenja aplikacije i intuitivnost za korisnike. Fokus diplomskog rada je na pružanju korisničkog sučelja koje je pregledno, dobro organizirano. Cilj je olakšati korisnicima obavljanje zdravstvenih zadataka bez nepotrebnih poteškoća. Sigurnost predstavlja zaštitu osjetljivih korisničkih podataka te sprječavanje neovlaštenog pristupa sustavu. U okviru diplomskog rada, bit će implementirane odgovarajuće sigurnosne mjere kako bi se osiguralo da samo ovlašteni korisnici imaju pristup podacima i funkcijama. Pouzdanost se odnosi na stabilnost i dosljednost rada sustava. Posvetit će se posebna pažnja testiranju kako bi se osiguralo ispravno funkcioniranje sustava i izbjegle eventualne greške ili prekidi u radu. Performanse su ključne za brzinu i efikasnost sustava u odgovaranju na korisničke zahtjeve. Dostupnost je važna kako bi korisnici mogli pristupiti sustavu u svakom trenutku kada to zahtijevaju. Planirat će se strategije za minimalizirane vremena nedostupnosti sustava, a implementirat će se odgovarajući mehanizmi oporavka u slučaju problema (vraćanje baze pomoću kreirane sigurnosne kopije). Skalabilnost se odnosi na prilagodljivost sustava promjenjivim uvjetima i zahtjevima. Također je

bitno da aplikacija može rasti, tj. biti nadograđivana bez većih utjecaja na brzinu. Svakim novim dodavanjem, aplikacija bi trebala održati slične performanse. Svi ovi nefunkcionalni zahtjevi bit će temeljito razmotreni tijekom dizajniranja i implementacije sustava kako bi se osiguralo visokokvalitetno korisničko iskustvo, sigurnost, pouzdanost te zadovoljile performanse i dostupnost potrebne za učinkovito praćenje zdravstvenog stanja pacijenata.

3.3. Korišteni postupci za stvaranje preporuka

Svi navedeni zahtjevi dovode nas do biranja na koji način bi trebalo preložiti alergiju i prikladan lijek s obzirom na unesene podatke pacijenta.

3.3.1. Primjena višekriterijskog postupka

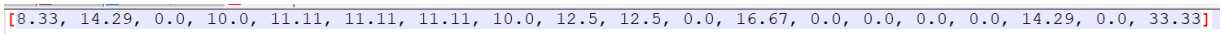
Po uzoru na [16], višekriterijsko filtriranje, poznato i kao višekriterijsko odabiranje ili višekriterijska analiza, je postupak u kojem se koristi više kriterija kako bi se donijela odluka o odabiru ili rangiranju skupa elemenata. Ova metoda omogućuje uzimanje u obzir više ciljeva ili preferencija pri odabiru najboljeg mogućeg rješenja. U ovom postupku, svaki element koji se razmatra ima svoje atribute ili karakteristike koje se procjenjuju. Ti atributi predstavljaju različite ciljeve, potrebe ili preferencije. Na primjer, u kontekstu stvaranja preporuka u zdravstvenom sustavu, mogu se koristiti različiti kriteriji poput učinkovitosti liječenja, sigurnosti lijeka, troškova, dostupnosti i nuspojava kako bi se ocijenili različiti lijekovi i terapije. Sam postupak višekriterijskog filtriranja uključuje identifikaciju kriterija, normalizaciju podataka, dodjelu težinskih vrijednosti kriterijima, računanje ukupne ocjene za svaki element i na kraju, izbor najboljih elemenata prema zadanim kriterijima. Ova tehnika može se primijeniti u različitim područjima, od poslovnih odluka, odabira proizvoda, stvaranja preporuka, analize podataka pa sve do drugih situacija u kojima je potrebno uzeti u obzir više faktora pri donošenju odluka.

3.3.2. Definiranje kriterija za preporuke

Filtriranje na temelju sadržaja u višekriterijskom filtriranju podrazumijeva proces odabira elemenata ili opcija na temelju njihovih karakteristika ili atributa. „Sadržaj“ se u ovom kontekstu odnosi na vrijednosti tih atributa koji opisuju svaki element ili opciju. U ovom postupku, koristimo različite kriterije kako bismo ocijenili svaki element ili opciju. Ti kriteriji mogu biti različite prirode, uključujući numeričke, kvalitativne ili druge vrste atributa. Nakon prikupljanja podataka o atributima za svaki element, provodimo ocjenjivanje kako bismo procijenili ukupnu vrijednost ili kvalitetu svake opcije. Višekriterijsko filtriranje omogućuje nam uzimanje u obzir više aspekata ili preferencija

prilikom donošenja odluka. Ovaj postupak omogućuje nam identifikaciju najboljih opcija koje najbolje odgovaraju našim potrebama i ciljevima, uzimajući u obzir višestruke čimbenike [24].

Prema [21], postoje dva tipa preporuka koji će se odrađivati. Jedan se odnosi na preporuku alergije, a drugi za preporuku lijeka. Kako bi preporučili alergiju koju pacijent ima, pacijent najprije mora odabrati simptome koje je primijetio da se pojavljuju. Ako simptomi nisu odabrani, pacijent nije u mogućnosti pokrenuti opciju pronalaska alergije. Tek nakon što pacijent ima neke simptome odabrane, može se ići u preporuku alergije koja uzima simptome te na temelju njih predlaže alergiju koja najviše odgovara odabranim simptomima. Svaka od alergija koja je pokrivena u radu, pokraj sebe ima definirane simptome koje ju karakteriziraju. Brojanjem pronađenih simptoma svake alergije, dobivamo listu koja unutar sebe sadrži postotke kolika je šansa da je alergija pogođena na temelju simptoma. Primjer liste prikazan je slikom 3.1.



Slika 3.1. *Prikaz liste postotaka točnosti prepoznate alergije*

Zatim se pronalazi indeks najvećeg broja, te sukladno tom indexu u Excel-u gdje su definirane alergije, na tom indeksu se uzima alergija. Za preporuku lijeka, prvotno je potrebno odraditi preporuku alergije. Nakon što je alergija preporučena, dolazi se do višekriterijske preporuke lijeka koja na temelju više parametara (alergija, dob, kilogrami i trudnoća) donosi rješenje, tj. pronađeni lijek. Tablicom 3.1 prikazani su koji parametri su korišteni kod višekriterijskog filtriranja.

Tablica 3.1. *Prikaz parametara bitnih kod višekriterijskog filtriranja lijeka i alergije*

Alergija	**Lijek**
Simptomi	dijagnosticirana alergija
	dob (god)
	težina (kg)
	trudnoća

Sukladno navedenim parametrima koji određuju koji lijek će se koristit, tu još pri samoj potvrdi lijeka, liječnik uzima u obzir i kontraindikacije koje je pacijent prvobitno naveo na svome profilu.

Pacijentova osobna situacija igra ključnu ulogu u odabiru odgovarajućeg lijeka, što je sasvim očekivano jer se lijekovi biraju prema njihovim specifičnim potrebama i parametrima. Neki od najvažnijih čimbenika koji se uzimaju u obzir su dob pacijenta, njegova tjelesna težina te trudnoća. Prema tome, dob, tjelesna težina, trudnoća su izrazito značajni parametri koje treba razmotriti kod svakog pacijenta. Na primjer, pacijentu koji je mlađi od preporučene dobi za određeni lijek, pacijentu koji ima manju tjelesnu težinu ili pacijentici koja je trudna, ne bi se trebali preporučiti lijekovi koji

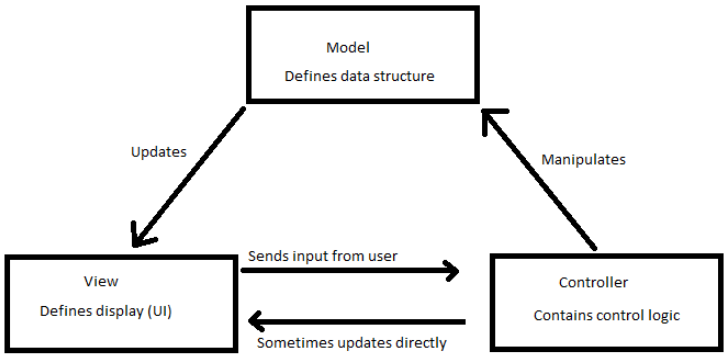
zahtijevaju određene kriterije koje pacijent ne ispunjava. Takvi lijekovi mogu imati nepoželjne nuspojave ili kontraindikacije koje bi mogli naštetiti pacijentu ili nerođenom djetetu. Stoga, individualiziran pristup i pažljivo razmatranje osobnih parametara pacijenta ključni su kako bi se osiguralo sigurno i učinkovito liječenje koje odgovara specifičnim potrebama svakog pojedinog pacijenta.

3.4. Građa programskog rješenja

Građa programskog rješenja se odnosi na strukturiranje i organiziranje samog softverskog sustava koji je razvijen s ciljem rješavanja specifičnog problema ili pružanja određene usluge. Ovaj dio obično se nalazi u tehničkim izvješćima ili dokumentaciji te detaljno opisuje arhitekturu, ključne komponente, dizajn i implementaciju cijelog sustava. To uključuje detaljno opisivanje različitih slojeva, modula, komponenti i njihovih uloga u cjelokupnom sustavu. Kroz građu programskog rješenja, daje se dublji uvid u unutarnju strukturu i organizaciju softverskog sustava, što pomaže u razumijevanju načina na koji sustav funkcionira kako bi pružio potrebne funkcionalnosti i performanse.

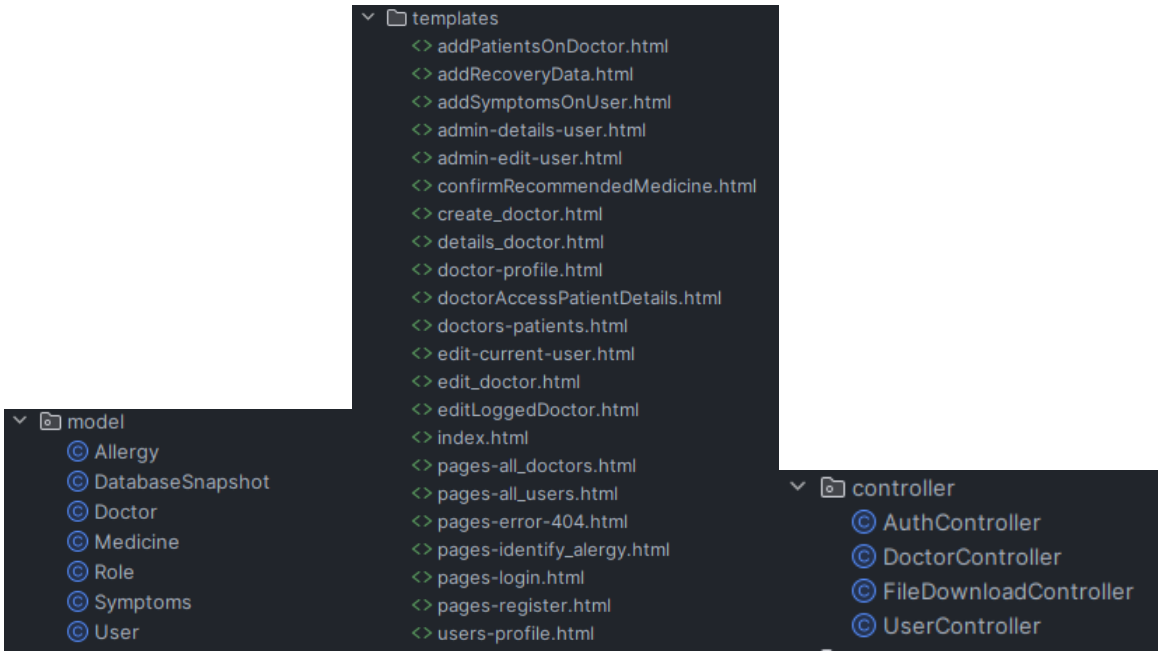
3.4.1. Predložak MVC (engl. *Model View Controller*)

MVC arhitektura je organizacijski obrazac koji se često koristi u razvoju softverskih aplikacija kako bi se jasno razdvojile različite odgovornosti i dijelovi aplikacije. Ova arhitektura pomaže u boljoj organizaciji programskog koda, olakšava održavanje i poboljšava skalabilnost aplikacije. Sastoji se od tri glavne komponente: modela, izgleda (engl. *View*) i kontrolera. Model predstavlja logičku strukturu i podatke aplikacije, uključujući poslovnu logiku, pristup bazi podataka i manipulaciju podacima. Izgled predstavlja korisničko sučelje koje korisnik vidi i s kojim se služi. Ono prikazuje podatke iz modela i reagira na korisničke akcije. Prema [17], kontroler djeluje kao posrednik između modela i izgleda. Prima korisničke akcije, obrađuje ih i ažurira model te osigurava da stranica prikaže ažurirane podatke. Osnovna svrha MVC arhitekture je omogućiti neovisnost između ovih komponenti, što olakšava promjene u jednom dijelu aplikacije bez utjecaja na ostale dijelove. To olakšava održavanje aplikacije i potiče princip "razdvajanja brige", gdje svaka komponenta ima jasno definiranu odgovornost. Ova organizacija čini kod čitljivijim, olakšava suradnju među timovima i pridonosi efikasnijem razvoju aplikacije. Slika 3.2 prikazuje način rada MVC arhitekture.



Slika 3.2. *Prikaz MVC Arhitekture* [19]

Ova arhitektura dijeli sustav na tri glavne komponente: Model koji sadrži logiku, izgled (engl. *View*) koji predstavlja korisnički sučelje, i kontroler koji upravlja promjenama na web stranici. U ranim danima interneta, MVC arhitektura se uglavnom primjenjivala na serverskoj strani, gdje bi klijenti putem formi ili linkova zahtijevali ažuriranja, a zatim bi dobivali osvježene prikaze koje bi se prikazivale u pregledniku. Međutim, u današnje vrijeme, sve više logike premješta se na klijentsku stranu zahvaljujući razvoju skladišta podataka na klijentskoj strani i tehnologiji XMLHttpRequest koja omogućava djelomično osvježavanje stranice po potrebi [18]. MVC arhitektura predloženog programskog rješenja prikazana je slikom 3.3.



Slika 3.3. *Prikaz MVC arhitekture unutar web aplikacije*

3.4.2. Arhitektura sustava na strani korisnika

Prema [17], arhitektura sustava na strani korisnika uključuje sve komponente i procese koji se odvijaju u korisničkom pregledniku. U kontekstu MVC arhitekture, to uključuje korisničko sučelje i dio aplikacijske logike koja se odvija na klijentskoj strani. Korisničko sučelje je odgovorno za prikazivanje informacija korisniku i interakciju s korisnikom. Thymeleaf, zajedno s HTML-om, omogućuje definiranje korisničkog sučelja i dinamičko umetanje podataka iz Modela (poslovne logike). Thymeleaf koristi sintaksu koja omogućuje integraciju Java objekata u HTML kod, što pojednostavljuje generiranje dinamičkog sadržaja na strani korisnika. JavaScript je skriptni jezik koji se izvršava u pregledniku korisnika i omogućuje stvaranje interaktivnih i dinamičkih elemenata na korisničkom sučelju. Može ga se koristiti za validaciju unosa korisnika, manipulaciju DOM-om (engl. *Document Object Model*), asinkronu komunikaciju s poslužiteljem (AJAX), animacije i još mnogo toga. Bootstrap služi za dizajn i responzivnost. Stranice se automatski prilagođavaju različitim veličinama ekrana, uključujući računala, tablete i mobilne uređaje.

3.4.3. Arhitektura sustava na strani poslužitelja

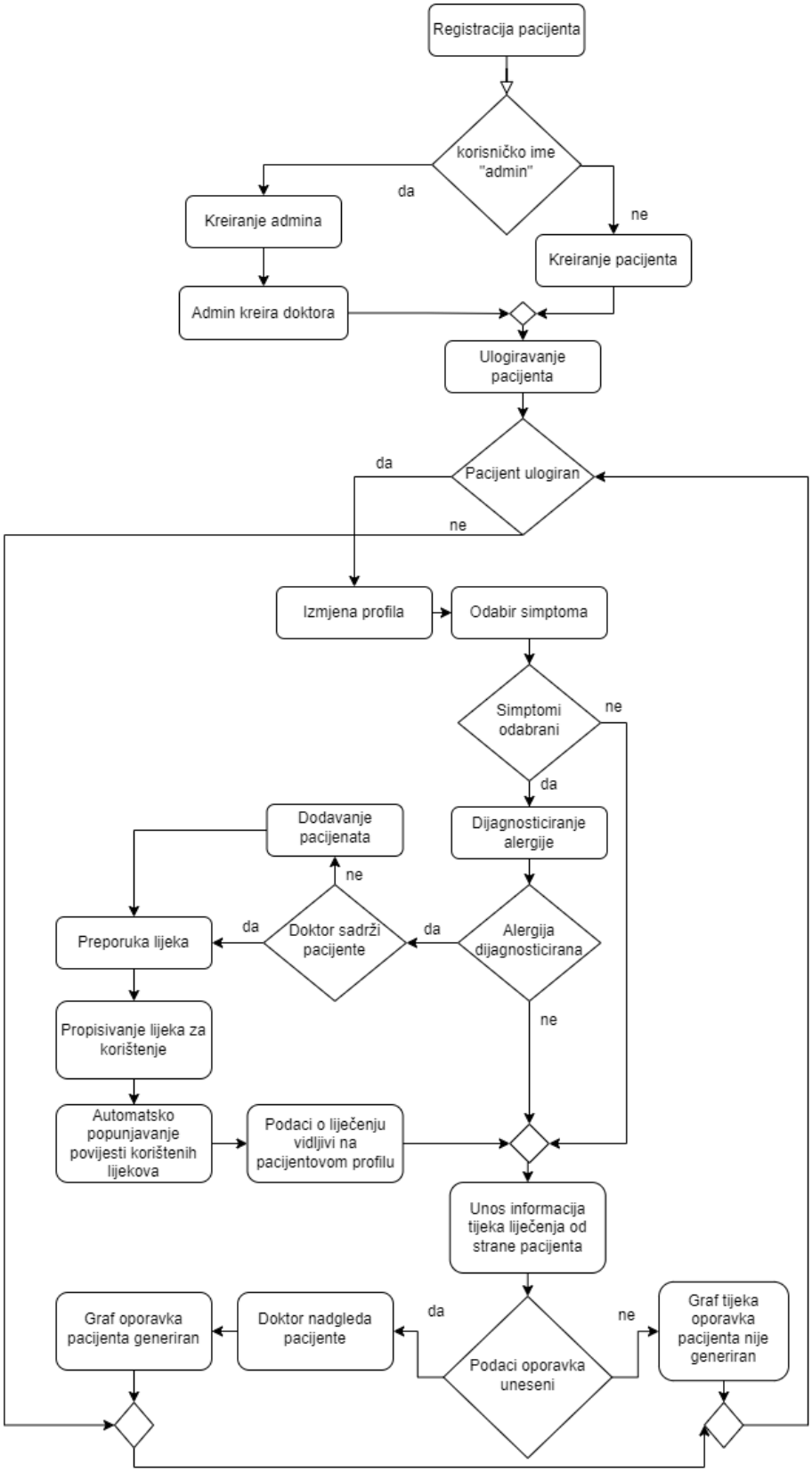
Arhitektura sustava na strani poslužitelja odnosi se na sve komponente i procese koji se odvijaju na poslužitelju, uključujući obradu poslovne logike (Model) i upravljanje zahtjevima klijenata (Kontroler). Model predstavlja poslovnu logiku aplikacije i pristup bazi podataka. U modelu su implementirate funkcionalnosti koje odgovaraju za poslovnu logiku, obradu podataka i pristupanje bazi podataka. To uključuje izvođenje poslovnih pravila, obradu korisničkih zahtjeva i pripremu podataka za prikaz na korisničkom sučelju. Kontroler je središnja točka za obradu dolaznih zahtjeva od klijenata (preglednici korisnika). Na temelju tih zahtjeva, kontroler tumači korisničke akcije i odgovarajuće reagira. On komunicira s modelom kako bi dohvatio ili ažurirao podatke potrebne za odgovor, a zatim rezultate predaje odgovarajućoj stranici za prikaz korisniku. Java se često koristi za implementaciju modela i kontrolera. U modelu, koristit ćete Java klase za definiranje objekata, metoda i funkcionalnosti koje predstavljaju vašu poslovnu logiku [18]. U kontroleru, Java će obrađivati dolazne zahtjeve od klijenata i odlučivati kako se ti zahtjevi trebaju obraditi i kojim putem preusmjeriti zahtjev. PostgreSQL uz pomoć jave, može komunicirati s bazom podataka kako bi dohvaćala, ažurirala i pohranjivala podatke koje koristi model aplikacije. Na slici 3.6 može se vidjeti prikaz dijagrama tijeka aplikacije a na slici 3.4 prikaz modela pacijenta te na slici 3.5 prikaz modela liječnika unutar aplikacije.

```java
@Entity
@Table(name = "users")
public class User {

    public User(String name, String email, String username, String password, String symptoms, String detectedAllergy,
            String recoveryProcess, String recommendedMedicine, String proscribedMedicine, String medicationUsage,
            String doctor, String age, String isPregnant, String weight, String contraindication) {
        this.name = name;
        this.email = email;
        this.username = username;
        this.password = password;
        this.symptoms = symptoms;
        this.detectedAllergy = detectedAllergy;
        this.recoveryProcess = recoveryProcess;
        this.recommendedMedicine = recommendedMedicine;
        this.proscribedMedicine = proscribedMedicine;
        this.medicationUsage = medicationUsage;
        this.doctor = doctor;
        this.age = age;
        this.isPregnant = isPregnant;
        this.weight = weight;
        this.contraindication = contraindication;
    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    3 usages
    @Column(nullable = false)
    private String name;

    3 usages
    @Column(nullable = false, unique = true)
    private String email;

    3 usages
    @Column(nullable = false, unique = true)
```

Slika 3.4. *Prikaz modela pacijenta*

```java
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "doctors")
public class Doctor {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;

    @Column(nullable = false)
    private String email;

    @Column(nullable = false)
    private String username;

    @Column(nullable = false)
    private String password;

    @Column
    private String patients;

    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinTable(name = "doctor_roles", joinColumns = {
            @JoinColumn(name = "DOCTOR_ID", referencedColumnName = "ID") }, inverseJoinColumns = {
                    @JoinColumn(name = "ROLE_ID", referencedColumnName = "ID") })
    private List<Role> roles = new ArrayList<>();

    public Doctor(String name, String email, String username, String password, String patients) {
        this.name = name;
        this.email = email;
        this.username = username;
        this.password = password;
        this.patients = patients;
    }

    public Long getId() { return id; }
```

Slika 3.5. *Prikaz modela liječnika*



Slika 3.6. *Prikaz dijagrama toka aplikacije*

3.5. Osnove API testiranja

Prema [28], API (engl. *Application Programming Interface*) je sučelje koje omogućuje različitim aplikacijama, sustavima ili servisima da komuniciraju i razmjenjuju podatke i funkcionalnosti. API su bitni za moderni razvoj softvera jer omogućuju integraciju i interoperabilnost između različitih komponenti i sustava. Općenito, API testiranje odnosi se na provjeru ispravnosti i pouzdanosti API-ja, provjeru njegovih funkcionalnosti, te obradu i prijenos podataka. Ovo je kritično jer, kada se API koristi u proizvodnom okruženju, bilo kakav problem u API-ju može uzrokovati probleme u cjelokupnom sustavu.

API se mogu testirati na različite načine, uključujući ručno testiranje, automatizirano testiranje i testiranje pomoću raznih alata. Neki od najčešćih pristupa API testiranju uključuju:

1. Ručno testiranje: Ovo je proces provođenja testiranja API-ja ručno, gdje se provjeravaju različiti zahtjevi, odgovori i povratne informacije. Testeri mogu koristiti alate poput Postmana kako bi ručno slali zahtjeve API-ju i analizirali odgovore. Ovaj pristup je koristan za testiranje manjeg broja scenarija ili kada API još nije dovoljno razvijen za automatizirano testiranje.
2. Automatizirano testiranje: Automatizirano testiranje API-ja koristi skripte i alate koji izvode testove umjesto ljudi. To može uključivati testiranje pojedinačnih zahtjeva, pozivanje API metoda s različitim ulaznim podacima i provjeru izlaznih podataka. Popularni alati za automatizaciju API testiranja uključuju Postman kolekcije, REST Assured, i JUnit s bibliotekama za testiranje API-ja.

API i Thymeleaf imaju različite funkcionalnosti i primjene u razvoju web aplikacija. U aplikacijama koje koriste Thymeleaf, API se obično koristi za komunikaciju s poslužiteljskom stranom i dobivanje podataka iz baze podataka ili drugih vanjskih izvora. API može pružiti podatke koje Thymeleaf koristi za generiranje dinamičkog sadržaja i prikaz na korisničkom sučelju.

Najbolji načini testiranja API-ja:

- Testiranje svih HTTP metoda (GET, POST, PUT, DELETE) koje API podržava.
- Provjera odgovora API-ja na različite ulazne parametre i uvjete.
- Testiranje autorizacije (ako je primjenjivo) kako bi se osiguralo da samo ovlašteni korisnici mogu pristupiti API resursima.
- Testiranje grešaka kako bi se osigurala ispravna i jasna reakcija API-ja na neočekivane situacije.

Važno je imati dobro pokrivene testove koji osiguravaju da API i Thymeleaf rade kako je očekivano i da se mogu uspješno integrirati u web aplikaciji. Automatizacija testiranja može uštedjeti vrijeme i osigurati dosljednost i pouzdanost testova. Sveukupno, dobro testiranje API-ja ključno je za izgradnju stabilnih i pouzdanih web aplikacija koje pružaju kvalitetno korisničko iskustvo. Unutar rada bit će odrađeno automatsko testiranje provedeno pomoću alata Cucumber.

4. PROGRAMSKO RJEŠENJE WEB APLIKACIJE

Programsko rješenje web aplikacije obuhvaća skup programskih komponenti, tehnologija i alata koji se koriste za stvaranje i djelovanje web aplikacija. Ove aplikacije izvode se preko web preglednika i omogućuju korisnicima pristup raznolikim funkcijama putem internetskog sučelja. Sustav se sastoji od korisničke strane koja je rađena u Thymeleaf-u, Bootstrapu i JavaScriptu dok je poslužiteljska strana kreirana pomoću modela, kontrolera, programskog jezika jave i baze podataka PostgreSQL.

4.1. Korišteni programski jezici i alati

Odabir jezika za izradu aplikacije je jednako bitan kao i odabir razvojnih alata za stvaranja i kreiranje aplikacije. Iz tog razloga je bitno navesti neke važne informacije alata i programskih jezika korištenih u izradi aplikacije.

4.1.1. Korišteni programski jezici

Uzorom na [20], Java je napredni programski jezik s objektno orijentiranim pristupom koji je razvijen 1995. godine od strane Sun Microsystems-a (sada Oracle Corporation). Izdvaja se po svojoj platformskoj neovisnosti, jednostavnosti i robusnosti. Koristi se za razvoj različitih aplikacija, uključujući web, igara i sustava za upravljanje bazama podataka . JavaScript je skriptni jezik klijentske strane koji omogućuje interaktivnost i dinamiku na web stranicama. To znači da se izvodi u web pregledniku korisnika i omogućuje manipulaciju HTML-om, CSS-om, obradu događaja i interakciju s korisnikom. Web stranice postaju dinamičnije i prilagođene korisniku zahvaljujući JavaScriptu. PostgreSQL je pouzdan sustav za upravljanje bazama podataka s mogućnostima transakcija, referencijalne cjelovitosti i podrškom za razne tipove podataka. Kao relacijska baza podataka, koristi SQL za upravljanje i manipulaciju podacima. PostgreSQL je čest izbor za web aplikacije koje traže sigurnu i učinkovitu bazu podataka. Ova kombinacija tehnologija često se koristi za stvaranje cjelovitih i skalabilnih web aplikacija.

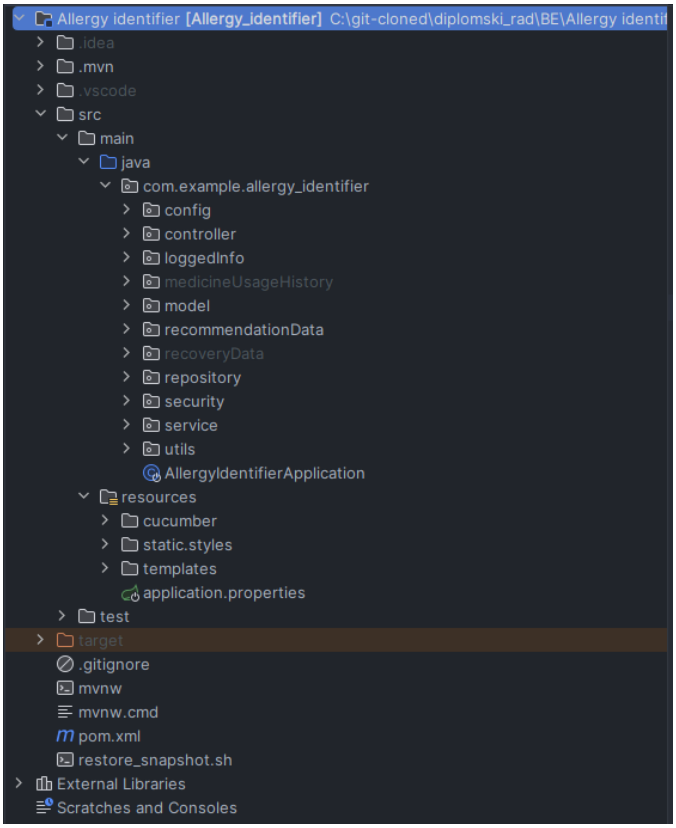
4.1.2. Korišteni razvojni alati

U razvoju web aplikacija koristio sam nekoliko korisnih alata koji su me podržavali tijekom cijelog procesa. Spring Web je dio Spring Frameworka koji se koristi za razvoj web aplikacija. To je moćan okvir koji pruža sve potrebne alate i mehanizme za izgradnju skalabilnih, sigurnih i visoko performantnih web aplikacija. Spring Web se temelji na MVC arhitekturi. InteliJ ima jako dobru integraciju sa Spring Web te je to razlog odabira razvojnog alata InteliJ. Ono je snažno integrirano razvojno okruženje koje pruža podršku za Java i druge jezike. Njegove napredne značajke, poput

refaktoriranja, automatskog dovršavanja i integracije s verzijama sustava, olakšavaju rad s MVC arhitekturom. Također, njegova ugrađena podrška za povezivanje s bazom podataka omogućuje jednostavno upravljanje podacima. Također je korišten i Visual Studio Code, popularan tekstualni uređivač koji pruža proširive mogućnosti za programiranje u raznim jezicima. Zahvaljujući jednostavnoj navigaciji i intuitivnom sučelju, ovaj alat također olakšava rad s MVC arhitekturom i ubrzava razvoj aplikacija.

Za testiranje API-ja koristio sam Postman. Njegove ugrađene funkcije omogućuju brzo provjeravanje zahtjeva i odgovora API-ja bez pisanja programskog koda. To mi je omogućilo brži razvoj web aplikacija temeljenih na MVC arhitekturi. Git je distribuirani sustav za upravljanje verzijama koji sam koristio za praćenje promjena u kodu tijekom vremena. Njegove ugrađene funkcije za suradnju među programerima, povrat na prethodne verzije i upravljanje verzijama projekta omogućavalo je strukturiran rad. SourceTree, kao grafički klijent za Git, pružio mi je vizualno upravljanje Git repozitorijima i olakšao pregled grana i promjena. DBeaver, besplatni alat za upravljanje bazama podataka, pružio je podršku za povezivanje s raznim bazama podataka i napredne funkcije za upravljanje podacima. To mi je omogućilo jednostavnu interakciju s bazom podataka i brži razvoj web aplikacija. PGAdmin sam koristio kako bih inicijalno mogao kreirati bazu podataka za aplikaciju, te nakon toga sav radi s bazom sam radio u DBeaveru.

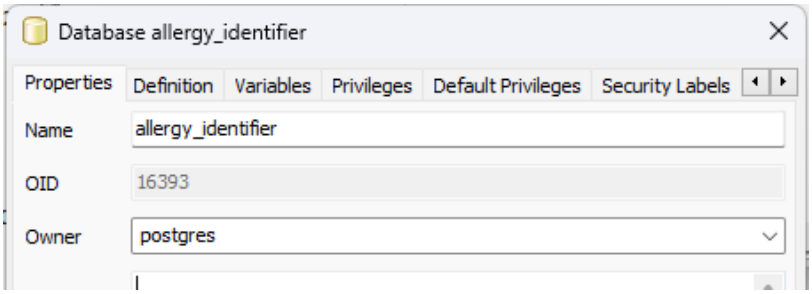
Uz navedene alate, korišteni su i neki dodaci instalirani unutar alata, primjera Prettier koji su omogućuje automatsko poravnanje programskog koda, te samim time lakši rad i razvijanje same aplikacije. Još jedan od instaliranih alata unutar InteliJ je ESLint alat koji služi za ukazivanje i otklanjanje grešaka unutar JavaScript jezika. Također jer korišten GitLab, mjesto na kojemu je spremljena cijela povijest razvijanja aplikacije. Poslužiteljska i klijentska strana nalaze se unutar istoga repozitorija jer je cijela aplikacija razvijana u Spring Web app okviru koji nudi zajedničku integraciju klijentske i poslužiteljske strane (nije potrebno odvajati u posebni repozitoriji, već kad se aplikacija pokreće, pale se i klijentska i poslužiteljska strana istovremeno). Prikaz strukture projekta prikazan je slikom 4.1.



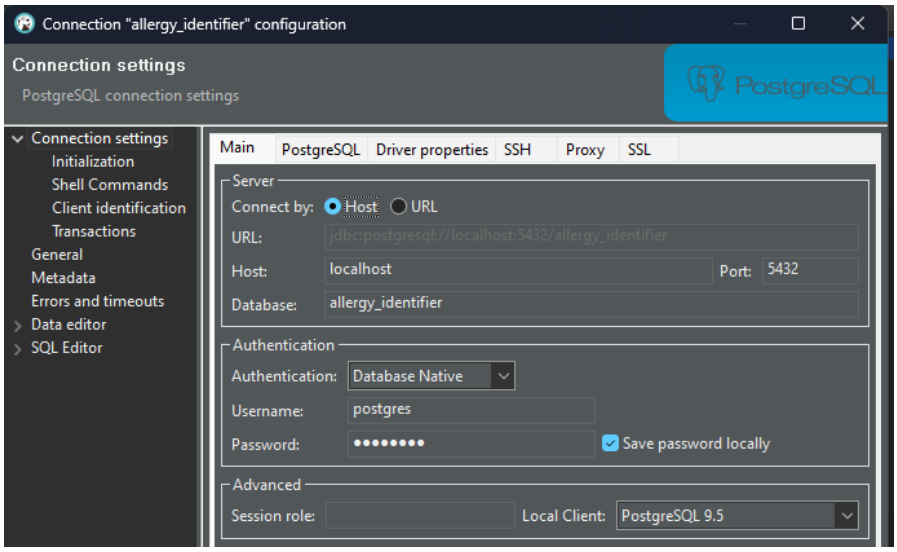
Slika 4.1. *Prikaz strukture repozitorija aplikacije*

4.1.3. Integracija s bazom podataka

U svrhu povezivanja vaše aplikacije s PostgreSQL bazom podataka, prvo je korišteno PG Admin sučelje kako bi se kreirala nova baza naziva allergy_identifier što je prikazano slikom 4.2. Nakon kreiranja baze, pomoću alata DBeaver je također uspostavljena konekcija zbog lakšeg pregleda baze. Detaljne postavke kreirane baze prikazano je na slici 4.3 Unutar tog prozora potrebno je definirati host, ono označava lokaciju poslužitelja baze podataka s kojim se DBeaver treba povezati. Također je potrebno navesti i korisničko ime (engl. *username*) i lozinku (engl. *password*) onoga koji ima ovlasti za pristup i upravljanje bazom podataka. Još ostaje definirati port na kojem poslužitelj sluša veze, koji je obično 5432.



Slika 4.2. *Prikaz kreiranja baze*



Slika 4.3. *Postavke kreirane baze podataka*

Nakon što je baza kreirana, postavke spajanja s bazom unešene su u application.properties datoteku unutar Spring aplikacije što je prikazano slikom 4.4. Kada aplikacija pokrene svoje procese, automatski prepoznaje definiranu allergy_identifier bazu i uspostavlja vezu s njom koristeći PostgreSQL driver. Ovaj driver omogućuje aplikaciji da učinkovito komunicira s bazom podataka, izvršava upite i manipulira podacima koji se odnose na alergije. Zahvaljujući integraciji s PostgreSQL bazom, aplikacija može sigurno i pouzdano upravljati podacima o alergijama. PostgreSQL je poznat po svojoj pouzdanosti i sigurnosti, što omogućuje očuvanje integriteta podataka. Ako se u budućnosti odluči promijeniti bazu podataka ili povezati aplikaciju s drugom bazom, dovoljno je jednostavno ažurirati postavke veze u application.properties datoteci. Aplikacija će tada koristiti nove postavke kako bi uspostavila vezu s odabranom bazom, dok će ostatak aplikacije ostati netaknut. Zadnja postavka na slici 4.4 govori aplikaciji da pri svakome pokretanju ne briše trenutno stanje baze, nego samo ažurira njeno stanje ako ima promjena definiranih unutar modela.

```
application.properties
1  server.port=8080
2  spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
3  spring.datasource.url=jdbc:postgresql://localhost:5432/allergy_identifier
4  spring.datasource.username=postgres
5  spring.datasource.password=postgres
6  spring.jpa.hibernate.ddl-auto=update
```

Slika 4.4. *Prikaz konekcija na bazu unutar aplikacije*

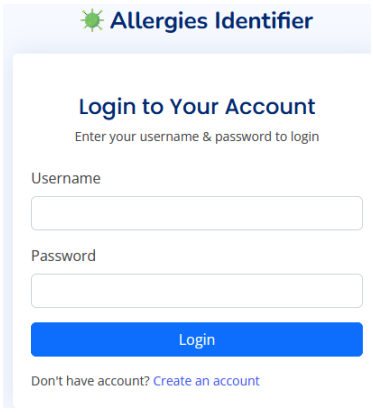
Integracija s bazom podataka čini aplikaciju moćnim alatom za upravljanje podacima vezanim uz alergije i identifikaciju alergena. Ovaj korak je od iznimne važnosti za izgradnju robusne i skalabilne aplikacije koja je spremna za buduće izazove i mogućnosti proširenja.

4.2. Programsko ostvarenje web aplikacije

Programsko ostvarenje web aplikacije podrazumijeva kreiranje web aplikacije kroz programski kod i integraciju različitih tehnologija kako bi omogućilo njezino funkcioniranje na internetu. To uključuje razvoj *backend* i *frontend* dijelova aplikacije i njihovo povezivanje kako bi zajednički pružali funkcionalnost korisnicima.

4.2.1. Naslovna stranica

Kada korisnik posjeti naslovnu stranicu, dočekuje ga izbor između dvije opcije: registracije ili prijave što je prikazano slikom 4.5. Kod za prikaz stranice i akcija gumba login prikazana je slikom 4.6. Odabirom teksta kreiraj račun, otvara se novi prozor koji omogućava registraciju korisnika. Kod koji omogućuje prikaz tog prozora prikazan je slikom 4.7, a kod koji pokreće akciju klikom na gumb registracije prikazan je slikom 4.8.



Slika 4.5. *Prikaz početne stranice aplikacije*

```
<form class="row g-3 needs-validation" method="post" role="form" th:action="@{/pages-login}">
  <div class="col-12">
    <label class="form-label">Username</label>
    <div class="input-group has-validation">
      <input type="text" name="username" class="form-control" id="username" value="" required />
      <div class="invalid-feedback">Please enter your username.</div>
    </div>
  </div>

  <div class="col-12">
    <label class="form-label">Password</label>
    <input type="password" name="password" class="form-control" id="password" value="" required />
    <div class="invalid-feedback">Please enter your password!</div>
  </div>

  <div class="col-12">
    <button class="btn btn-primary w-100" type="submit">Login</button>
  </div>

  <div th:if="${param.error}">
    <div class="alert alert-danger">Invalid username or password</div>
  </div>

  <div th:if="${param.logout}">
    <div class="alert alert-success">You have been logged out.</div>
  </div>

  <div class="col-12">
    <p class="small mb-0">
      Don't have an account? <a th:href="@{/pages-register}">Create an account</a>
    </p>
  </div>
</form>
```

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.csrf().disable()
            .authorizeHttpRequests((authorize) -> authorize.antMatchers("/pages-all_users")
                    .hasRole("ADMIN") AuthorizationManagerRequestMat...
                    .antMatchers("/pages-all_doctors").hasRole("ADMIN")
                    .antMatchers("/users-profile").hasRole("USER")
                    .antMatchers("/doctor-profile").hasRole("DOCTOR")
                    .antMatchers("/pages-identify_alergy").hasRole("USER")
                    .antMatchers("/users-profile").hasRole("USER")
                    .antMatchers("/index").permitAll()

            ).formLogin(
                    form -> form
                            .loginPage("/pages-login")
                            .loginProcessingUrl("/pages-login")
                            .defaultSuccessUrl("/index")
                            .permitAll())
            .logout(
                    logout -> logout
                            .logoutRequestMatcher(
                                    new AntPathRequestMatcher("/logout"))
                            .permitAll());
    return http.build();
}

DESKTOP-BO7DN5L\Ivan
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth
            .userDetailsService(userDetailsService)
            .passwordEncoder(passwordEncoder());
}
```

Slika 4.6. *Prikaz programskog koda za izgled i autorizaciju korisnika*

HTML programski kod sa slike 4.6 predstavlja HTML formu koja se prikazuje na stranici /pages-login. Forma sadrži dva polja za unos korisničkog imena i lozinke, te gumb za prijavu. Korisnicima se prikazuju naslov i opis forme za prijavu, a u slučaju pogrešne prijave ili odjave, prikazuju se odgovarajuće poruke o pogrešci. Drugi dio programskog koda sa slike 4.6 je Java konfiguracija za Spring Security. Ovaj dio konfigurira sigurnost aplikacije koristeći Spring Security okruženja. Anotacija @Configuration označava da je riječ o konfiguracijskoj klasi, dok @EnableWebSecurity omogućuje Spring Security za web aplikaciju. Metoda passwordEncoder() definira Bean koji koristi NoOpPasswordEncoder za preskakanje kodiranja lozinke korisnika. Glavni dio konfiguracije je metoda filterChain(HttpSecurity http), koja postavlja pravila za autorizaciju i autentifikaciju korisnika. Prvo, onemogućuje se CSRF zaštita kako bi testiranje aplikacije bilo lakše. Zatim se definiraju pravila za određene URL-ove, gdje su neki od njih dostupni svim korisnicima, dok drugi zahtijevaju određene uloge za pristup. Definira se putanja za prijavu i postupak prijave putem forme, te putanja za odjavu korisnika. U metodi configureGlobal(AuthenticationManagerBuilder auth), koristi se UserDetailsService kako bi se dohvatili podaci o korisniku za autentifikaciju. Kada korisnik pokuša pristupiti zaštićenim URL-ovima ili se prijaviti na aplikaciju, Java konfiguracija će obraditi zahtjev. Ako korisnik nije prijavljen ili nema odgovarajuće ovlasti, bit će preusmjeren na stranicu za prijavu. Kada korisnik unese ispravne podatke za prijavu, Spring Security će autentificirati korisnika i omogućiti mu pristup odgovarajućim URL-ovima u skladu s definiranim pravilima autorizacije. U slučaju neuspješne prijave, korisniku će biti prikazana poruka o pogrešci. Slikom 4.7 prikazan je HTML programski kod registracije pacijenta dok slika 4.8 prikazuje programski kod registracije korisnika.

```
<div class="card mb-3">
  <div class="card-body">
    <div class="pt-4 pb-2">
      <h5 class="card-title text-center pb-0 fs-4">
        Create an Account
      </h5>
      <p class="text-center small">
        Enter your personal details to create account
      </p>
    </div>

    <div th:if="${param.success}">
      <div class="alert alert-success">
        You have successfully registered!
      </div>
    </div>

    <form class="row g-3 needs-validation" method="post" role="form" th:action="@{/pages-register}" th:object="${user}">
      <div class="col-12">
        <label class="form-label">Your Name</label>
        <input type="text" name="name" class="form-control" id="name" th:field="*{name}" required />
        <p th:errors="*{name}" class="text-danger" th:if="${#fields.hasErrors('name')}"></p>
      </div>

      <div class="col-12">
        <label class="form-label">Your Email</label>
        <input type="email" name="email" class="form-control" id="email" th:field="*{email}" required />
        <p th:errors="*{email}" class="text-danger" th:if="${#fields.hasErrors('email')}"></p>
      </div>

      <div class="col-12">
        <label class="form-label">Username</label>
        <div class="input-group has-validation">
          <input type="text" name="username" class="form-control" id="username" th:field="*{username}" required />
          <p th:errors="*{username}" class="text-danger" th:if="${#fields.hasErrors('username')}"></p>
        </div>
      </div>
```

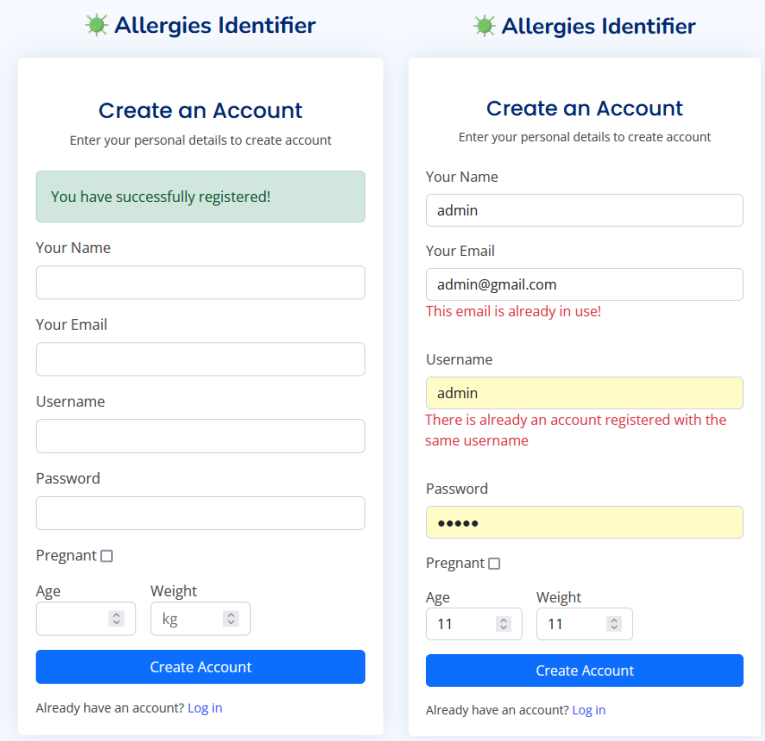
```
      <div class="col-12">
        <label class="form-label">Password</label>
        <input type="password" name="password" class="form-control" id="password" th:field="*{password}" required />
        <p th:errors="*{password}" class="text-danger" th:if="${#fields.hasErrors('password')}"></p>
      </div>

      <div class="col-12">
        <label for="isPregnant">Pregnant</label>
        <input type="checkbox" id="isPregnant" name="isPregnant" value="false" th:field="*{isPregnant}" />
      </div>

      <div class="col-4">
        <label for="age">Age</label>
        <input type="number" name="age" class="form-control" id="age" th:field="*{age}" required />
      </div>

      <div class="col-4">
        <label for="age">Weight</label>
        <input type="number" name="weight" class="form-control" id="weight" placeholder="kg" th:field="*{weight}" required />
      </div>

      <div class="col-12">
        <button class="btn btn-primary w-100" type="submit">
          Create Account
        </button>
      </div>
      <div class="col-12">
        <p class="small mb-0">
          Already have an account?
          <a th:href="@{/pages-login}">Log in</a>
        </p>
      </div>
    </form>
```

Slika 4.7. *Prikaz programskog koda za izgled stranice registracije*

```java
@PostMapping("/pages-register")
public String registration(@Valid @ModelAttribute("user") User user,
        BindingResult result,
        Model model) {
    User existingUser = userService.findUserByUsername(user.getUsername());
    User existingEmail = userService.findUserByEmail(user.getEmail());

    if (existingUser != null && existingUser.getUsername() != null && !existingUser.getUsername().isEmpty()) {
        result.rejectValue( field: "username",  errorCode: null,
                defaultMessage: "There is already an account registered with the same username");
    }

    if (existingEmail != null && existingEmail.getEmail() != null && !existingEmail.getEmail().isEmpty()) {
        result.rejectValue( field: "email",  errorCode: null,
                defaultMessage: "This email is already in use!");
    }

    if (result.hasErrors()) {
        model.addAttribute( attributeName: "user", user);
        return "/pages-register";
    }

    userService.saveUser(user);
    return "redirect:/pages-register?success";
}
```

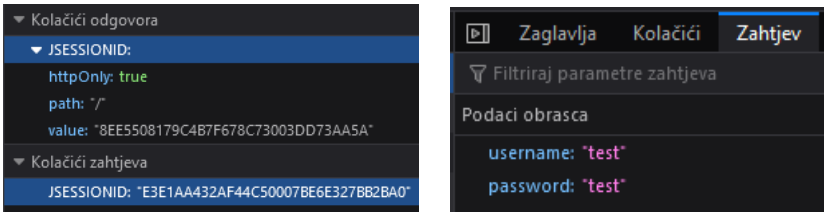
Slika 4.8. *Prikaz programskog koda za registraciju korisnika*

Programski kod sa slike 4.8 prikazuje metodu kontrolera u Spring Boot aplikaciji. Ta metoda obrađuje POST zahtjeve na putanji /pages-register te provjerava podatke koje korisnik unosi u formi za registraciju. Metoda prima tri parametra: korisnik (engl. *user*) koji sadrži podatke iz HTTP zahtjeva, rezultat (engl. *result*) koji sadrži rezultate provjere validacije i model za prijenos podataka na predložak za prikazivanje korisniku. Ako već postoji korisnik s istim korisničkim imenom ili e-mail adresom, korisnik će dobiti odgovarajuću poruku o grešci. Ako postoje greške u unosu podataka, korisnik će biti preusmjeren na istu stranicu za registraciju kako bi ispravio pogreške. U suprotnom, kada su uneseni podaci ispravni, novi korisnik će biti spremljen u bazu podataka. Nakon uspješne registracije, korisnik će biti preusmjeren na stranicu /pages-register?success s obavijesti o uspješnoj registraciji. Uspješna registracija kao i kriva registracija prikazani su slikom 4.9. Prilikom registracije, svaka registracija korisnika podrazumijeva da je to pacijent, te mu se dodjeljuje uloga USER, dok se uloga admin kreira samo u slučaju ako je korisničko ime ADMIN, a samo ADMIN ima mogućnost kreiranja liječnika što će kasnije biti objašnjeno.



Slika 4.9. *Prikaz uspješne i neuspješne registracije*

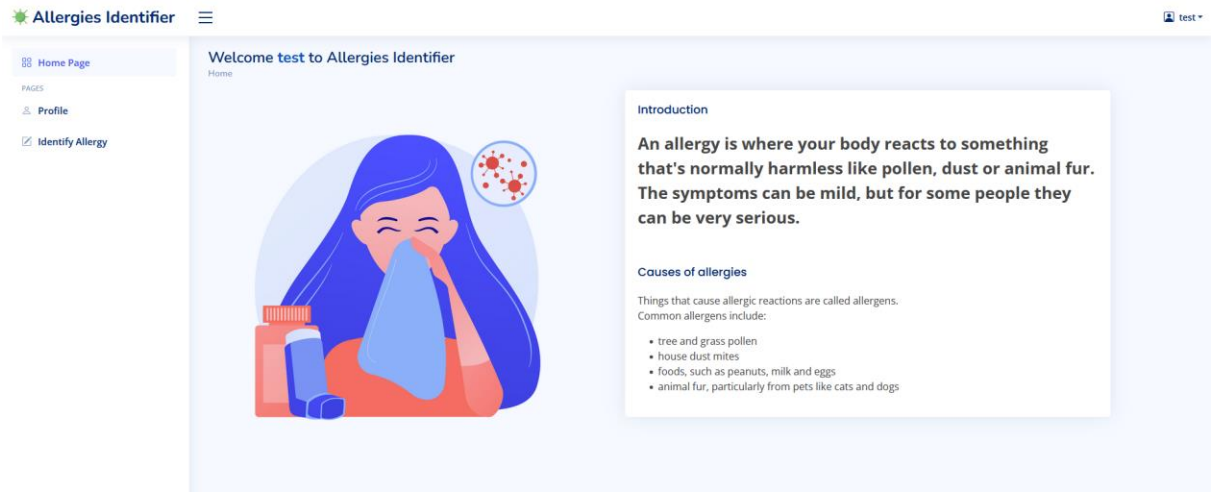
Nakon uspješne registracije, korisnik se vraća na početnu stranicu, gdje se prijavljuje sa svojim prijašnje unesenim podacima. Prilikom obrađivanja zahtjeva, ako su podaci točni korisnik je prijavljen i dobiva JSESSIONID token kojim se pamti stanje prijavljenog korisnika. JSESSIONID je poseban kolačić koji web aplikacije koriste kako bi pratile i održavale sesije za svakog pojedinog korisnika. Kada korisnik prvi put pristupi web stranici ili aplikaciji i prijavi se, server dodjeljuje jedinstveni identifikator sesije koji se pohranjuje u kolačiću JSESSIONID na korisnikovom uređaju, obično u web pregledniku. Svaki put kada korisnik šalje novi zahtjev prema istoj web aplikaciji, preglednik automatski šalje JSESSIONID kolačić u zaglavlju zahtjeva što je prikazano slikm 4.10. To omogućuje serveru da prepozna korisnika i održava kontinuiranu sesiju za tu osobu.



Slika 4.10. *Prikaz dodijeljenog tokena i zahtjev prijave korisnika s podacima*

4.2.2. Mogućnosti prijavljenog pacijenta

Uspješnom prijavom, korisnik dolazi na početnu stranicu aplikacije koja je za sve uloge unutar aplikacije ista, stranica dobrodošlice (4.11.a). Prikaz HTML programskog koda prikazan je slikom (4.11.b).



a)

```
<section class="section">
    <div class="row">

        <div class="col-lg-5">
            <img class="image" th:src="@{/styles/images/allergy - main.jpg}" alt="">
        </div>

        <div class="col-lg-6">
            <div class="card">
                <div class="card-body">
                    <h5 class="card-title">Introduction</h5>
                    <p style="..."><b>An allergy is where your body reacts to something that's normally
                        harmless like pollen, dust or animal fur. The symptoms can be mild, but for some people they
                        can be very serious.</b></p>
                </div>

                <div class="col-lg-12">

                    <div class="card-body">
                        <h5 class="card-title">Causes of allergies</h5>
                        <p>Things that cause allergic reactions are called allergens.

                            <br>Common allergens include:

                        <ul>
                            <li>tree and grass pollen</li>
                            <li>house dust mites</li>
                            <li>foods, such as peanuts, milk and eggs</li>
                            <li>animal fur, particularly from pets like cats and dogs</li>
                        </ul>
                        </p>
                    </div>
                </div>

            </div>
        </div>

    </div>
</section>
```

b)

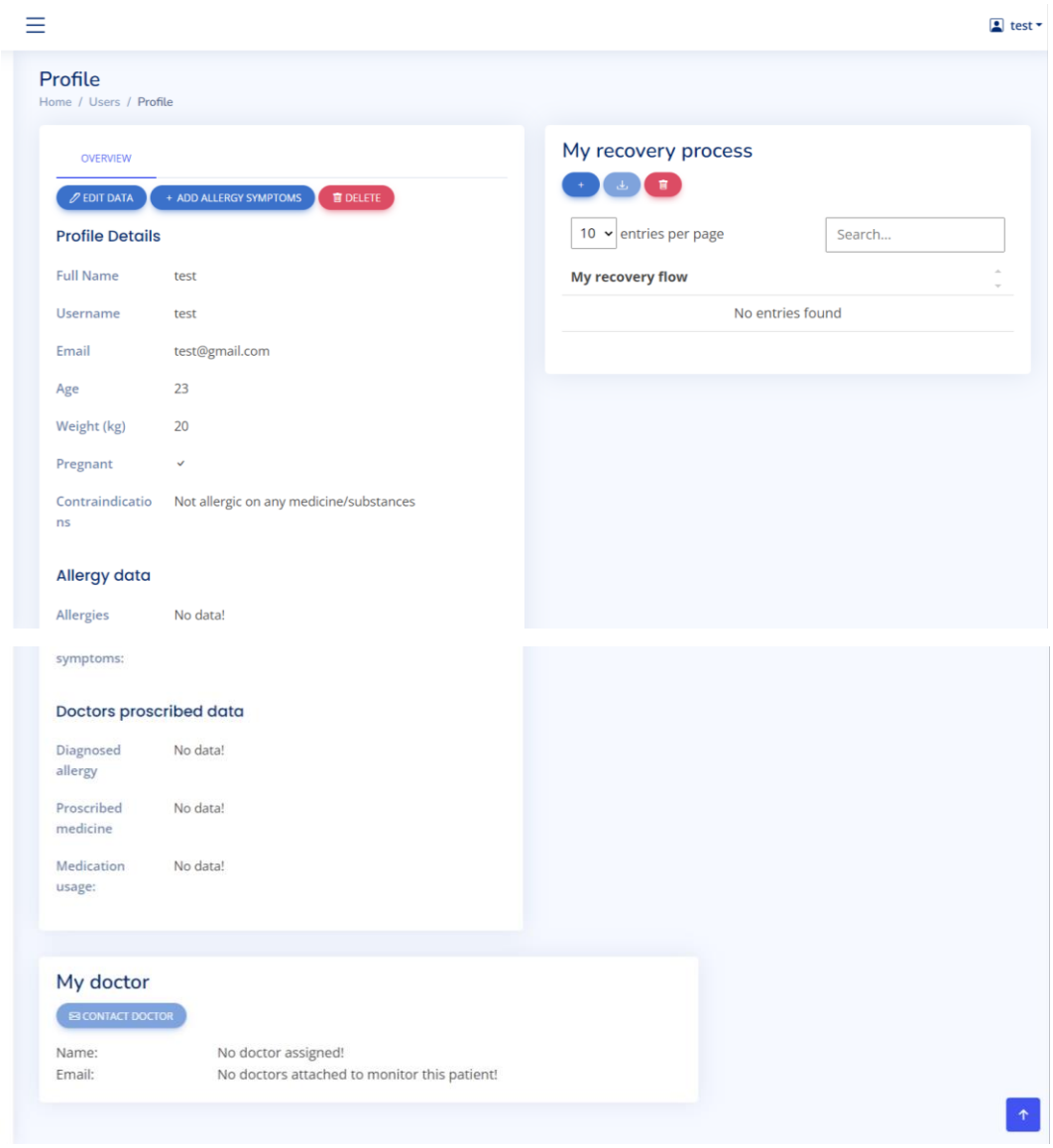
Slika 4.11. *Prikaz izgleda stranice dobrodošlice (a) i pripadajući programskog kod (b)*

Dolaskom na početnu stranicu, pacijent ima mogućnost pristupa svome profilu i pristupu stranice za preporuku alergije. Svaki od korisnika koji je unutar aplikacije, ovisno o svojoj dodijeljenoj ulozi, vidi određene izbornike s lijeve strane. Dodjela uloga na korisnike biti će kasnije detaljnije objašnjena. Na temelju uloge, generiraju se samo određeni izbornici.

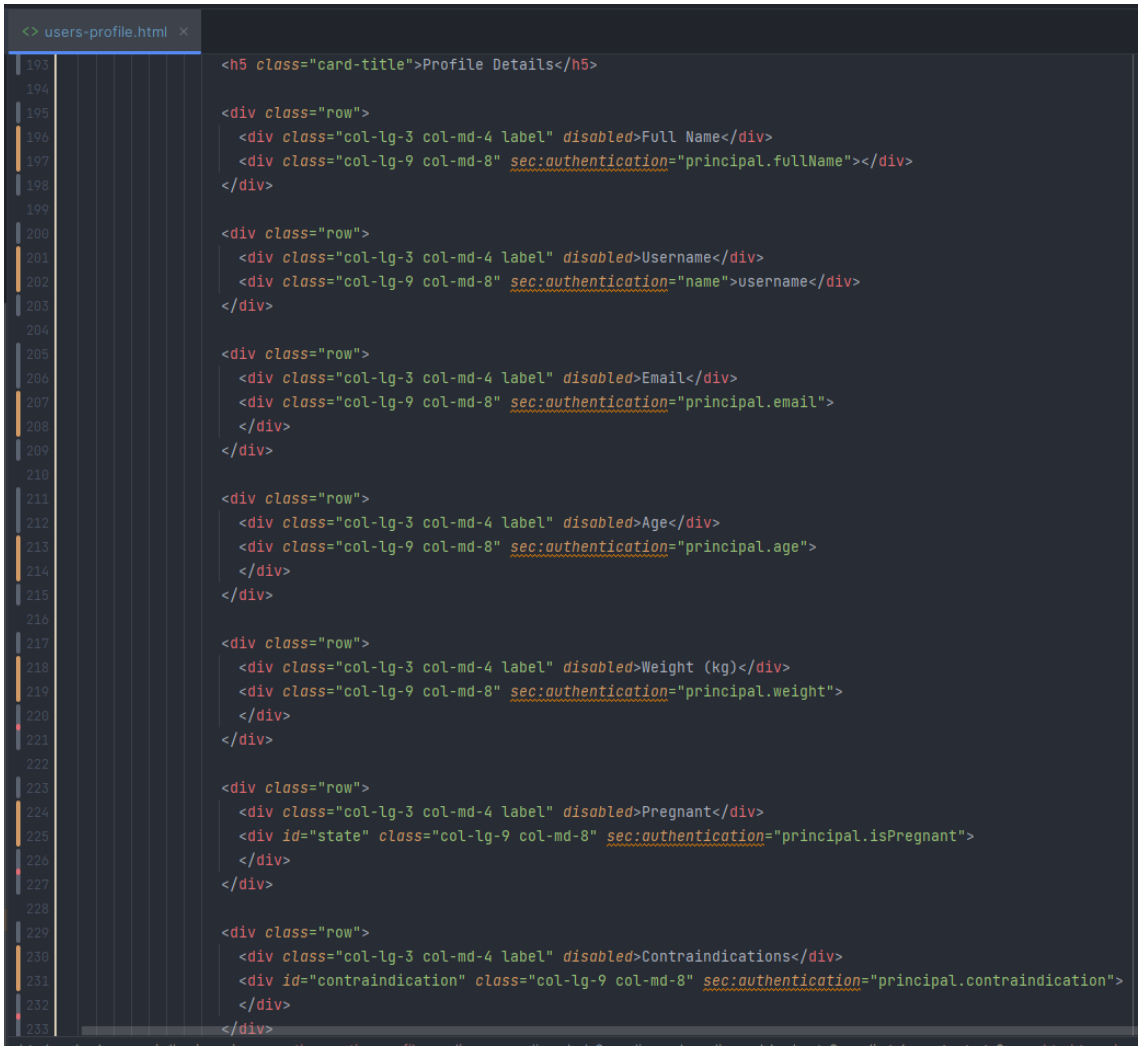
Izraz sec:authorize=hasRole(USER) se koristi u Spring Security za provjeru ima li trenutno prijavljeni korisnik određenu ulogu, u ovom slučaju USER. Ovaj izraz se koristi u HTML pogledu kako bi se kontroliralo koji dijelovi stranice su dostupni samo korisniku s tom specifičnom ulogom. Kada korisnik zatraži određenu stranicu koja sadrži ovaj izraz, Spring Security će provjeriti autorizacijske podatke tog korisnika kako bi vidio ima li ulogu USER. Ako korisnik ima tu ulogu, bit će mu dopušten pristup sadržaju obuhvaćenom ovim izrazom. Jedan od tih sadržaja je pacijentov profil koji je prikazan slikom 4.13. Međutim, ako korisnik nema ulogu USER, pristup takvom sadržaju će mu biti uskraćen ili će ga preusmjeriti na drugu stranicu koja je dostupna korisniku s drugom ulogom. Svaki od postojećih izbornika je definiran kao prema slici 4.12. Slikom 4.14 prikazan je HTML programski kod profila pacijenta.

```
<li sec:authorize="hasRole('USER')" class="nav-item">
  <a class="nav-link collapsed" th:href="@{/pages-identify_alergy}">
    <i class="bi bi-pencil-square"></i>
    <span>Identify Allergy</span>
  </a>
</li>

<li id="patients" sec:authorize="hasRole('DOCTOR')" class="nav-item">
  <a class="nav-link collapsed" th:href="@{/doctors-patients}">
    <i class="bi bi-file-medical"></i>
    <span>Patients</span>
  </a>
</li>

<li sec:authorize="hasRole('ADMIN')" class="nav-item">
  <a class="nav-link collapsed" th:href="@{/pages-all_users}">
    <i class="bi bi-person"></i>
    <span>Users</span>
  </a>
</li>
```

Slika 4.12. *Prikaz načina skrivanja/omogućavanja vidljivosti izbornika korisnicima*



Slika 4.13. *Prikaz profila pacijenta*



Slika 4.14. *Prikaz programskog koda profila stranice*

Zbog prevelikog sadržaja html dokumenta sa slike 4.14, nije moguće prikazati cijeli izgled stranice putem programskog koda. Izdvojen je dio koji je jednak za cijelu stranicu i način prikaza podataka je za sve isti te će on biti objašnjen. Kada se korisnik uspješno prijavi na web stranicu, njegovi korisnički podaci spremaju u principal objekt. Taj objekt sadrži sve relevantne informacije o prijavljenom korisniku. Na ovaj način omogućuje se prikaz atributa korisnika na odgovarajućem mjestu na web stranici. Na primjer, ako je korisnikov uzrast 25 godina, tada će se na stranici prikazati 25. Ova tehnika omogućuje prilagodbu sadržaja web stranice za svakog pojedinog korisnika i prikazuje mu relevantne informacije, kao što su dob, ime, e-mail adresa i slično. Svaki atribut pacijenta je prikazan na ovaj način. Korisnikov profil ima mogućnost uređivanja vlastitih podataka, dodavanja simptoma, brisanja podataka tijeka liječenja, kontaktiranje liječnika, dodavanje procesa oporavka te preuzimanje njega samog kao i njegovo potpuno brisanje.

Ulaskom u prozor uređivanja podataka, korisnik ima mogućnost izmjene svojih podataka, svih osim korisničkog imena i njegovog imena. Tu opciju ima samo ADMIN korisnik. Ispunjavanjem svakih od navedenih polja, okida se API za ažuriranje podataka prijavljenog pacijenta koji je prikazan slikom 4.15. Metoda koja se izvodi unutar API-ja prikazana je slikom 4.16.

```
@PostMapping("/edit-current-user/update")
public String saveDetails(User user, @AuthenticationPrincipal ShowMeLoggedUser loggedUser) {
    editUserProperty(user, property: "mainDataSave", newValue: "", loggedUser);
    return "redirect:/users-profile?success";
}
```

Slika 4.15. *Prikaz API-ja za ažuriranje podataka prijavljenog pacijenta*

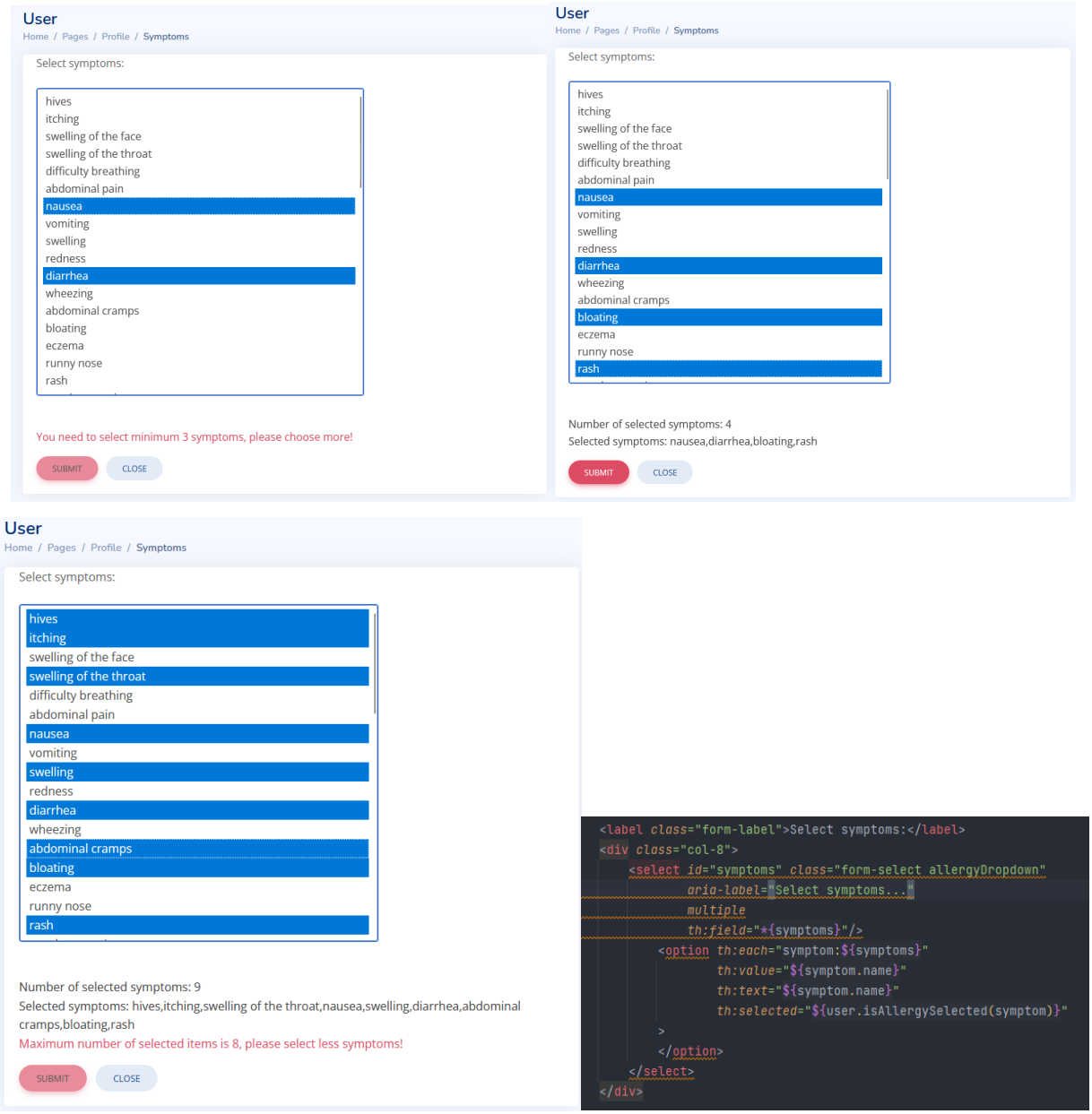
```
public void editUserProperty(User user, String property, String newValue, ShowMeLoggedUser loggedUser) {
    switch (property) {
        case "mainDataSave": {
            String checkboxResult = dataLoader.getCheckboxValue(request);

            loggedUser.setName(user.getName());
            loggedUser.setUsername(user.getUsername());
            loggedUser.setEmail(user.getEmail());
            loggedUser.setPassword(user.getPassword());
            loggedUser.setSymptoms(user.getSymptoms());
            loggedUser.setAge(user.getAge());
            loggedUser.setIsPregnant(checkboxResult);
            loggedUser.setWeight(user.getWeight());
            loggedUser.setContraindication(user.getContraindication());

            user.setId(loggedUser.getId());
            user.setDetectedAllergy(loggedUser.getDetectedAllergy());
            user.setProscribedMedicine(loggedUser.getProscribedMedicine());
            user.setSymptoms(loggedUser.getSymptoms());
            user.setEmail(loggedUser.getEmail());
            user.setName(loggedUser.getFullName());
            user.setUsername(loggedUser.getUsername());
            user.setPassword(loggedUser.getPassword());
            user.setMedicationUsage(loggedUser.getMedicationUsage());
            user.setRecommendedMedicine(loggedUser.getRecommendedMedicine());
            user.setRecoveryProcess(loggedUser.getRecoveryProcess());
            user.setDoctor(loggedUser.getDoctor());
            user.setAge(loggedUser.getAge());
            user.setWeight(loggedUser.getWeight());
            user.setContraindication(loggedUser.getContraindication());
            user.setIsPregnant(checkboxResult);
            setAndCheckAnyRole( roleName: "ROLE_USER", user);
            userService.updateUser(user);
            break;
        }
```

Slika 4.16. *Metoda za ažuriranje podataka prijavljenog pacijenta*

Na slici 4.16 postoje dvije metode: editUserProperty i saveDetails. Metoda editUserProperty koristi se za uređivanje svojstava korisnika na temelju predanog parametra property. Ovisno o vrijednosti property, metoda će izvršiti odgovarajuće promjene na korisnikovim svojstvima. Na kraju, metoda će spremiti promjene u bazu podataka pozivom metode userService.updateUser(user). Metoda saveDetails se koristi za spremanje promjena u glavne korisnikove podatke. Metoda poziva editUserProperty metodu s vrijednošću mainDataSave za parametar property, što signalizira da se žele spremiti svi parametri koji su prisutni u prozoru za uređivanje. Nakon što se promjene spreme, korisnika se preusmjerava na stranicu /users-profile?success. Nakon toga, bit će dostupna opcija za dodavanje simptoma. Taj korak je vrlo bitan iz razloga što ako se taj korak ne odradi, preporuka alergije nije moguća. Prilikom odabira simptoma potrebno je paziti na pravila koja su pri tome određena. Najmanji broj simptoma koji je dopušten pri odabiranju je 3, a najveći broj 8. Razlog tome je velik broj kreiranih simptoma, te samim time pri daljnjem slanju tih simptoma, svaka od alergija dijeli dosta sličnih simptoma, te bi onda prilikom generiranja algoritma došlo do potencijalno loših rješenja. Prilikom odabira pomoću JavaScripta se prikazuju koji simptomi su odabrani kao i informacija koliko simptoma je odabrano i koliko ih je potrebno odabrati. Slika 4.17 prikazuje prikaz programskog koda ograničavanja broja odabira simptoma, dok je na slici 4.18 prikazan izgled web aplikacije prilikom odabira simptoma.

```
    //Select data input shown on web
if (url.endsWith("/addSymptomsOnUser")) {
    document.querySelector( selectors: 'select#symptoms').addEventListener( type: 'input',  listener: event : Event  => {
        let selectData : any[]  = Array.from(event.target.selectedOptions).reduce((data : any[] , opt : HTMLOptionElement ) : any[]  => {
            data.push([opt.value])
            return data
        }, [])
        document.querySelector( selectors: '#saveData').disabled = selectData.length > 8 || selectData.length <= 2;
        if (selectData.length > 8) {
            document.querySelector( selectors: '#numberOfSymptoms').textContent = `Number of selected symptoms: ${selectData.length}`
            document.querySelector( selectors: '#selectedSymptoms').textContent = `Selected symptoms: ${selectData}`
            document.querySelector( selectors: '#errorMessage').textContent = 'Maximum number of selected items is 8, please select less symptoms!'
        }else if(selectData.length <= 2){
            document.querySelector( selectors: '#numberOfSymptoms').textContent = ""
            document.querySelector( selectors: '#selectedSymptoms').textContent = ""
            document.querySelector( selectors: '#errorMessage').textContent = 'You need to select minimum 3 symptoms, please choose more!'
        }
        else {
            document.querySelector( selectors: '#errorMessage').textContent = ''
            document.querySelector( selectors: '#numberOfSymptoms').textContent = `Number of selected symptoms: ${selectData.length}`
            document.querySelector( selectors: '#selectedSymptoms').textContent = `Selected symptoms: ${selectData}`
        }
    })
}
```

Slika 4.17. *Prikaz JavaScript programskog koda za prikaz označenih simptoma*



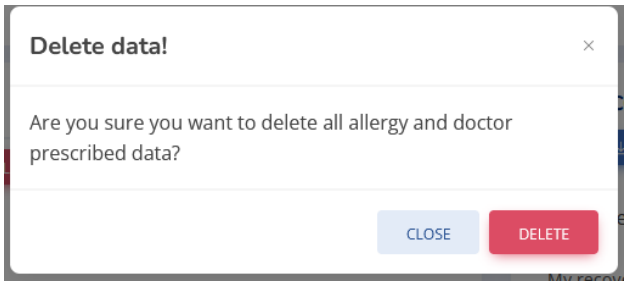
Slika 4.18. *Prikaz programskog koda načina prikaza i označavanja simptoma te prikaz web načina označavanja simptoma*

Nakon označavanja željenih simptoma, klikom na gumb potvrde, podaci se šalju u formu te se okida API za dodavanja simptoma koja je prikazana slikom 4.19. Metoda koja se okida unutar sebe poziva istu metodu kao i za ažuriranje podatak pacijenta koja je prethodno prikazana slikom 4.16. Na ovaj način se novi simptomi šalju a ostali podaci se uzimaju i spremaju onakvima kakvim su i bili.

```
@PostMapping("/addSymptomsOnUser/add")
public String addSymptomsOnUser(User user, @AuthenticationPrincipal ShowMeLoggedUser loggedUser) {
    editUserProperty(user, property: "mainDataSave", newValue: "", loggedUser);
    return "redirect:/users-profile?success";
}
```

Slika 4.19. *Prikaz API za dodavanje simptoma pacijenta*

Pacijent također ima mogućnost vraćanja na mogućnost gdje mu niti jedan podatak nije predložen od strane sustava osim njegovih podataka koje je unio tokom registracije. Samim time pacijent ima mogućnost krenuti s unošenjem podataka ispočetka. Klikom na gumb za brisanje, otvara se modal koji nas upozorava da će se podaci obrisati. Opisani modal je prikazan slikom 4.20.



Slika 4.20. *Prikaz modala potvrde brisanja podataka*

Klikom pacijenta na gumb potvrde brisanja, pokreće se forma koja zove API /removeUserSymptoms koja je prikazana slikom 4.21. Taj API poziva metodu editUserProperty, ali ovaj puta kao property šalje podatak resetAllergyData, što govori programu da mjesta koja se trebaju obrisati dobiju vrijednost No Data!. Nakon uspješnog brisanja, korisnik na profilu više ne vidi prijašnje generirane podatke.

```
@GetMapping("/removeUserSymptoms")
public String removeUserSymptoms(@AuthenticationPrincipal ShowMeLoggedUser loggedUser, User user) {
    editUserProperty(user, property: "resetAllergyData", newValue: "No data!", loggedUser);
    return "redirect:/users-profile";
}
```

Slika 4.21. *Prikaz API-ja za brisanje podataka*

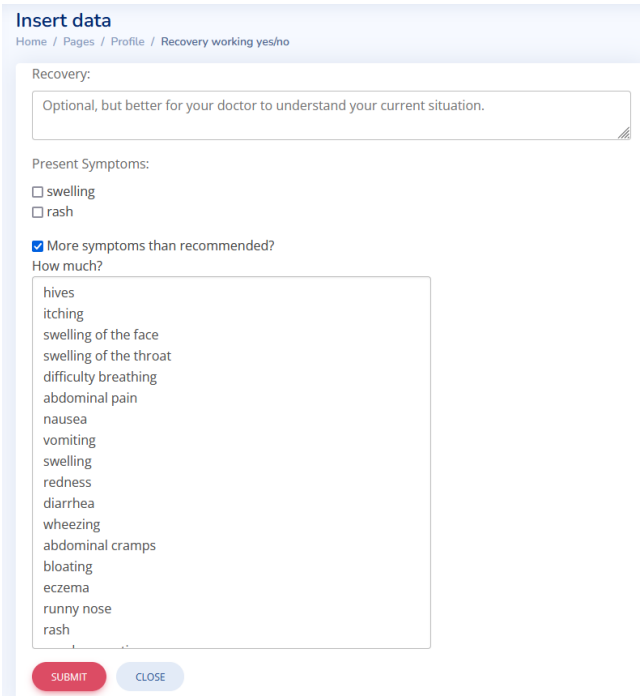
Svaki pacijent ima mogućnost kontaktiranja liječnika, naravno ako je liječnik preuzeo tog pacijenta na sebe. Ako je liječnik prikazan, gumb za kontaktiranje se omogućuje te klikom na njega otvara se prozor elektroničke pošte sa već unesenim mailom liječnika. Klikom na gumb, okida se JavaScript metoda openGmail() koja otvara URL unutar kojega nadopisuje mail od liječnika koji je uzet iz html

elementa sa id=doctorEmail. Gumb je onemogućen ako je prisutan tekst da liječnik ne postoji (engl. *No doctors attached to monitor this patient!*). Opisani kod prikazan je slikom 4.22.

```
313        <button class="btn btn-primary btn-rounded btn-sm mb-3 bi bi-envelope" id="sendMail" onclick="openGmail()"> Contact doctor</button>

1 usage
function openGmail() : void  {
    window.open( url: 'https://mail.google.com/mail/?view=cm&to=' + doctorEmail.innerText, target: '_blank');
}
```

Slika 4.22. *Prikaz html programskog koda i JavaScript metode kontaktiranja liječnika*

Među navedenim funkcionalnostima, pacijent također ima mogućnost praćenja u unošenja svog tijeka liječenja. Ima mogućnost odabira trenutnih simptoma koje ima nakon propisanoga lijeka te dodavanje nekih novih ako mu je taj lijek naštetio i pogoršao situaciju. Odabirom simptoma i potvrdom, tijekom idućeg unosa osjećanja pacijenta prikazuju se samo simptomi iz zadnje iteracije koje je prethodno označio. Prilikom potvrde unesenih podataka, kreira se Excel datoteka za svakog pacijenta unutar kojeg se spremaju uneseni podaci kao i broj prisutnih simptoma. Broj simptoma će kasnije biti korišten za iscrtavanje grafa liječniku kako bi vizualno imao predodžbu kako napreduje tijek liječenja pacijenta. Programski kod opisanog sadržaja (4.23.a) i programski kod izgleda stranice može se vidjeti na slici (4.23.b).



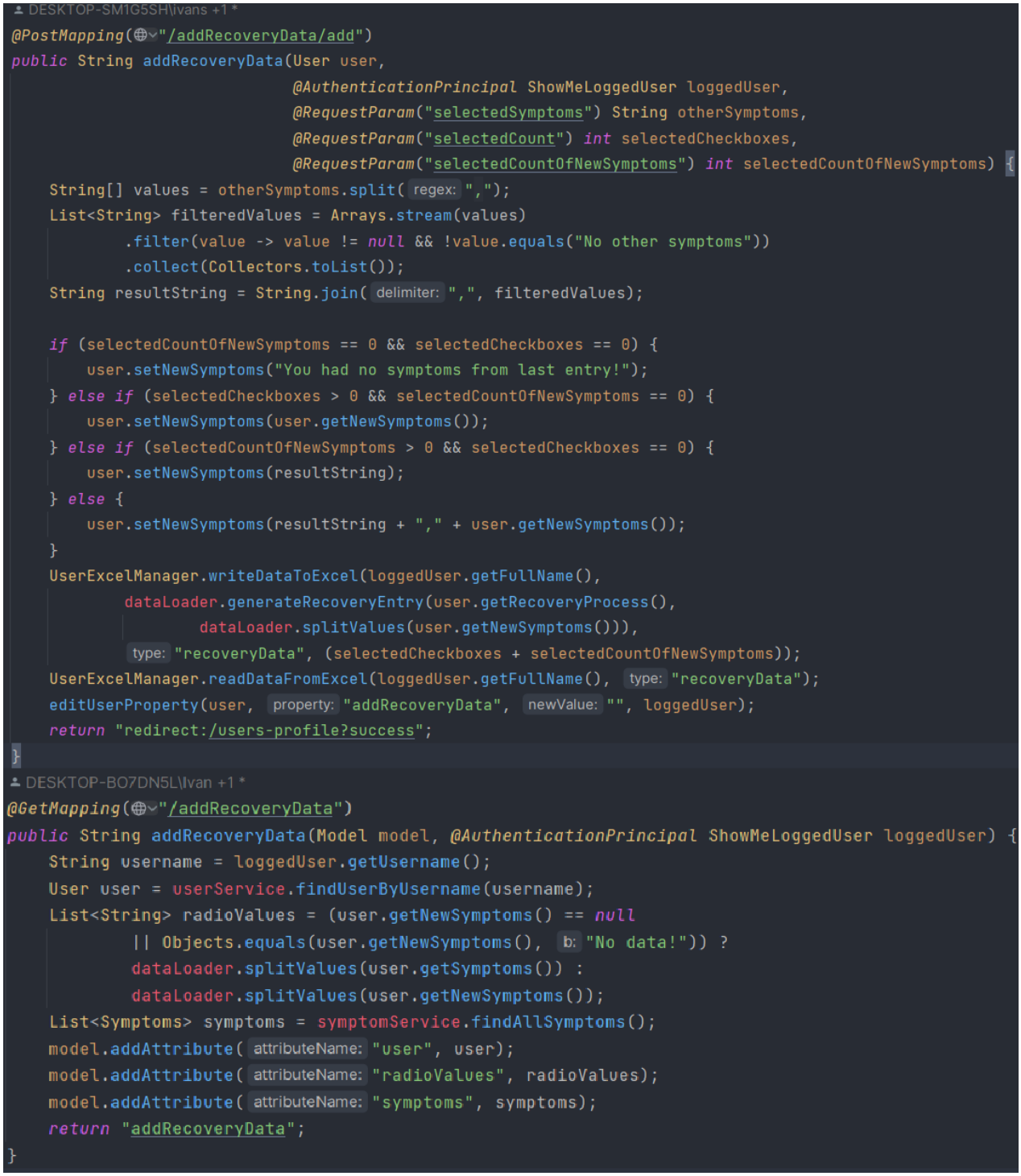
a)

```
<form class="row g-3 needs-validation" th:action="@{/addRecoveryData/add}" th:object="${user}" method="post" enctype="application/x-www-form-u
  <div class="col-12">
    <label class="form-label">Recovery:</label>
    <textarea type="text" name="recoveryProcess" class="form-control"
              th:field="*{recoveryProcess}" placeholder="Optional, but better for your doctor to understand your current situation."></textarea>
  </div>
  <div class="col-12" id="checkboxContainer">
    <label class="form-label">Present Symptoms:</label>
    <div th:each="element : ${radioValues}">
      <input type="checkbox" th:id="'checkbox_' + ${element}" name="newSymptoms" th:value="${element}"
             th:field="*{newSymptoms}" th:checked="${element != 'You had no symptoms from last entry!'}"
             th:disabled="${element == 'You had no symptoms from last entry!'}"/>
      <label th:for="'checkbox_' + ${element}" th:text="${element}"></label>
    </div>
    <input type="hidden" name="selectedCount" id="selectedCount" value="0">
  </div>
  <div class="col-12">
    <input type="checkbox" id="checkbox" onchange="toggleInputField()">
    <label for="checkbox">More symptoms than recommended?</label>
    <div id="inputContainer" style="...">
      <label for="inputField">How much?</label>
      <input type="hidden" name="selectedSymptoms" value="No other symptoms">
      <div class="col-8">
        <select id="inputField" class="form-select allergyDropdown"
                aria-label="Select symptoms..."
                multiple
                name="selectedSymptoms">
          <option th:each="symptom:${symptoms}"
                  th:value="${symptom.name}"
                  th:text="${symptom.name}"
          >
          </option>
        </select>
      </div>
      <input type="hidden" name="selectedCountOfNewSymptoms" id="selectedCountOfNewSymptoms" value="0">
    </div>
  </div>
  <div class="col-12">
    <button class="btn btn-danger btn-rounded" id="saveData" type="submit">Submit
    </button>
```

b)

Slika 4.23. *Prikaz izgleda stranice (a) i HTML programskog koda izgleda stranice (b)*

Način slanja simptoma na web, odrađen je pomoću modela, koji uzima simptome koje korisnik ima te ih razdvaja po zarezu te vraća polje sa simptomima pacijenta što je prikazano slikom 4.24. Kreirano polje se šalje na web, te se svaki iterira pojedinačno te se kreira kučica za označavanje (engl. *checkbox*) za svaki simptom u polju.

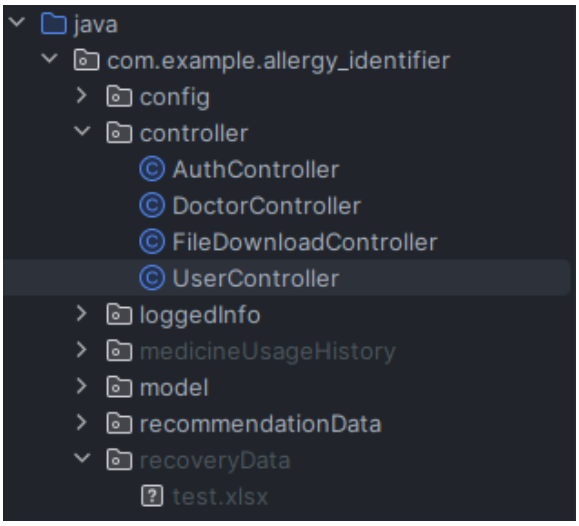


```
DESKTOP-SM1G5SH\ivans +1 *
@PostMapping("/addRecoveryData/add")
public String addRecoveryData(User user,
                              @AuthenticationPrincipal ShowMeLoggedUser loggedUser,
                              @RequestParam("selectedSymptoms") String otherSymptoms,
                              @RequestParam("selectedCount") int selectedCheckboxes,
                              @RequestParam("selectedCountOfNewSymptoms") int selectedCountOfNewSymptoms) {
    String[] values = otherSymptoms.split( regex: ",");
    List<String> filteredValues = Arrays.stream(values)
            .filter(value -> value != null && !value.equals("No other symptoms"))
            .collect(Collectors.toList());
    String resultString = String.join( delimiter: ",", filteredValues);

    if (selectedCountOfNewSymptoms == 0 && selectedCheckboxes == 0) {
        user.setNewSymptoms("You had no symptoms from last entry!");
    } else if (selectedCheckboxes > 0 && selectedCountOfNewSymptoms == 0) {
        user.setNewSymptoms(user.getNewSymptoms());
    } else if (selectedCountOfNewSymptoms > 0 && selectedCheckboxes == 0) {
        user.setNewSymptoms(resultString);
    } else {
        user.setNewSymptoms(resultString + "," + user.getNewSymptoms());
    }
    UserExcelManager.writeDataToExcel(loggedUser.getFullName(),
            dataLoader.generateRecoveryEntry(user.getRecoveryProcess(),
                    dataLoader.splitValues(user.getNewSymptoms())),
            type: "recoveryData", (selectedCheckboxes + selectedCountOfNewSymptoms));
    UserExcelManager.readDataFromExcel(loggedUser.getFullName(), type: "recoveryData");
    editUserProperty(user, property: "addRecoveryData", newValue: "", loggedUser);
    return "redirect:/users-profile?success";
}
DESKTOP-BO7DN5L\Ivan +1 *
@GetMapping("/addRecoveryData")
public String addRecoveryData(Model model, @AuthenticationPrincipal ShowMeLoggedUser loggedUser) {
    String username = loggedUser.getUsername();
    User user = userService.findUserByUsername(username);
    List<String> radioValues = (user.getNewSymptoms() == null
            || Objects.equals(user.getNewSymptoms(), b: "No data!")) ?
            dataLoader.splitValues(user.getSymptoms()) :
            dataLoader.splitValues(user.getNewSymptoms());
    List<Symptoms> symptoms = symptomService.findAllSymptoms();
    model.addAttribute( attributeName: "user", user);
    model.addAttribute( attributeName: "radioValues", radioValues);
    model.addAttribute( attributeName: "symptoms", symptoms);
    return "addRecoveryData";
}
```

Slika 4.24. *Prikaz API za potvrdu podataka liječenja*

Programski kod sa slike 4.24. predstavlja dio poslužiteljske logike koji se koristi za unos podataka o oporavku korisnika. Kada korisnik unese svoje podatke, ova funkcija obrađuje te podatke korak po korak. Na početku funkcija prima nekoliko parametara. Pacijent predstavlja objekt s informacijama o

korisniku, dok prijavljeni pacijent označava trenutno prijavljenog korisnika. otherSymptoms je niz simptoma koje je korisnik odabrao i razdvojio zarezima, dok selectedCheckboxes označava ukupan broj odabranih okvira za potvrdu. selectedCountOfNewSymptoms sadrži broj novih simptoma koje je korisnik odabrao. Najprije se polje otherSymptoms razdvaja na pojedinačne simptome pomoću zareza i pohranjuje se u niz stringova. Zatim se vrši filtriranje ovih simptoma kako bi se uklonile prazne vrijednosti i *"No other symptoms"*. Nakon toga, filtrirani simptomi se spajaju natrag u string koristeći zarez kao separator. Nadalje, kod provjerava kako su odabrani simptomi i poduzima odgovarajuće akcije. Ako nema odabranih novih simptoma i nema odabranih preporučenih simptoma, korisniku se postavlja poruka da nema odabranih simptoma. U slučaju da su odabrani preporučeni simptomi, ali nema novih simptoma, koristi se postojeći niz simptoma korisnika. Ako su odabrani novi simptomi, ali nema odabranih preporučenih, koriste se filtrirani novi simptomi. U svim ostalim situacijama, filtrirani novi simptomi se dodaju na postojeće simptome korisnika. Nakon obrade simptoma, podaci se zapisuju u Excel datoteku putem UserExcelManager modula. Ovaj modul omogućava pisanje i čitanje podataka iz Excel datoteka. Također se izvodi funkcija za čitanje podataka iz Excel datoteke. Konačno, funkcija ažurira svojstvo korisnika putem editUserProperty funkcije te korisnika preusmjerava na stranicu "users-profile" uz obavijest o uspješnom izvršenju akcije. Slikom 4.25 prikazano je pisanje unutar Excel datoteke.



a)

```java
public static void writeDataToExcel(String name, String data, String type, int userState) {
    try {
        Path directory = Objects.equals(type, b: "recoveryData") ? Paths.get(EXCEL_DIRECTORY)
                : Paths.get(EXCEL_MEDICINE_DIRECTORY);
        if (!Files.exists(directory)) {
            Files.createDirectories(directory);
        }
        File file = new File(Objects.equals(type, b: "recoveryData") ? EXCEL_DIRECTORY + name + ".xlsx"
                : EXCEL_MEDICINE_DIRECTORY + name + ".xlsx");
        Workbook workbook;

        if (file.exists()) {
            FileInputStream fis = new FileInputStream(file);
            workbook = new XSSFWorkbook(fis);
            fis.close();
        } else {
            workbook = new XSSFWorkbook();
        }
        Sheet sheet = workbook.getSheet( s: "Data");
        if (sheet == null) {
            sheet = workbook.createSheet( s: "Data");
        }

        int lastRowNum = sheet.getLastRowNum();
        Row newRow = sheet.createRow( i: lastRowNum + 1);
        Cell descriptionCell = newRow.createCell( i: 0);
        Cell userStateEntry = newRow.createCell( i: 1);
        descriptionCell.setCellValue(now + " --- " + data);
        userStateEntry.setCellValue(userState);

        FileOutputStream fos = new FileOutputStream(file);
        workbook.write(fos);
        fos.close();

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

b)

Slika 4.25. *Prikaz kreirane datoteke (a)i programskog koda metode pisanja u Excel (b)*

Metoda writeDataToExcel služi za pisanje podataka u Excel datoteku (.xlsx). Metoda prima nekoliko parametara: name (ime korisnika), data (podaci koji se žele zapisati), type (vrsta podataka - recoveryData ili medicineData), te userState (stanje korisnika). Metoda prvo provjerava postojanje direktorija za spremanje Excel datoteke i stvara ga ako ne postoji. Zatim odabire putanju i ime datoteke na temelju vrste podataka i imena korisnika. Nakon toga, provjerava postoji li već Excel datoteka s odabranim imenom. Ako postoji, otvara postojeću radnu knjigu kako bi se podaci ažurirali, a ako ne postoji, stvara novu radnu knjigu. Metoda zatim dohvaća ili stvara listu s imenom "Data" u radnoj knjizi. Na temelju postojećeg stanja liste, dodaje novi redak na kraj kako bi se zapisali novi podaci. U novom retku stvaraju se dvije ćelije: prva ćelija sadrži tekstualni opis koji uključuje trenutni datum

i predani podatak, dok druga ćelija sadrži stanje korisnika. Nakon što su podaci zapisani, radna knjiga se sprema na disk. Ako prilikom čitanja ili pisanja datoteke dođe do greške, ispisuje se stog izuzetaka kako bi se olakšalo pronalaženje greške u izvršavanju. Ovaj kod koristi Apache POI biblioteku za rad s Excel datotekama. Zatim se na pacijentovom profilu čita Excel datoteka te se podaci prikazuju u pregledniku. Programski kod čitanja datoteke prikazan je slikom 4.26.

```java
@GetMapping("/users-profile")
public String userProfile(Model model, @AuthenticationPrincipal ShowMeLoggedUser loggedUser, User user) {
    List<String> recoveryList = UserExcelManager.readDataFromExcel(loggedUser.getFullName(), type: "recoveryData");
```

a)

```java
public static List<String> readDataFromExcel(String name, String type) {
    List<String> newList = new ArrayList<>();
    try {
        File file = new File(Objects.equals(type, b: "recoveryData") ? EXCEL_DIRECTORY + name + ".xlsx"
                : EXCEL_MEDICINE_DIRECTORY + name + ".xlsx");
        if (file.exists()) {
            FileInputStream fis = new FileInputStream(file);
            Workbook workbook = new XSSFWorkbook(fis);
            Sheet sheet = workbook.getSheet( s: "Data");

            if (sheet != null) {
                for (Row row : sheet) {
                    Cell firstCell = row.getCell( i: 0); // Get the first cell of the row
                    if (firstCell != null) {
                        newList.add(firstCell.getStringCellValue()); // Add the value to the list
                    }
                }
            }

            fis.close();
        } else {
            System.out.println("Excel file does not exist for user: " + name);
        }

    } catch (IOException e) {
        e.printStackTrace();
    }
    return newList;
}
```

b)

Slika 4.26. *Prikaz poziva funkcije (a) i programskog koda čitanja Excel datoteke (b)*

Metoda readDataFromExcel ima zadatak čitati podatke iz Excel datoteke (.xlsx) i spremiti ih u listu. Prima dva parametra: name (ime korisnika) i type (vrsta podataka - recoveryData ili medicineData). Najprije se odabire putanja i ime datoteke na temelju vrste podataka i imena korisnika. Zatim se provjerava postoji li Excel datoteka s tim imenom. Ako postoji, datoteka se otvara za čitanje. U suprotnom, ispisuje se poruka da datoteka ne postoji za navedenog korisnika. Nakon što se otvori radna knjiga (engl. *workbook*), dohvaća se lista s imenom data. Ako lista postoji, metoda prolazi kroz svaki redak u listi i dohvaća vrijednost prve ćelije (engl. *cell*) u retku. Ako ta ćelija nije prazna,

vrijednost se dodaje u listu. Nakon što se prođe kroz sve retke u listi, FileInputStream se zatvara kako bi se oslobodili resursi, a zatim se vraća lista s pročitanim podacima.

Ta lista se prosljeđuje modelu koji prolazi kroz listu i prikazuje svaki podatak u tablici koja se generira programskim kodom sa slike 4.27.

```
<table class="table table-borderless datatable" id="firstTable">
  <thead>
  <tr>
    <th scope="col">My recovery flow</th>
  </tr>
  </thead>
  <tbody>
  <tr th:each="recoveryLists : ${recoveryList}">
    <td th:text="${recoveryLists}"></td>
  </tr>
  </tbody>
</table>
```

Slika 4.27. *Način prikaza podataka iz Excel datoteke u pregledniku*

Pacijent također ima mogućnost preuzimanja svoje datoteke tijeka liječenja ako ona postoji. Programski kod preuzimanja datoteke prikazan je slikom 4.28.a, a slikom 4.28.b prikazana implementacija metode preuzimanja datoteke.

Metoda downloadFile služi za preuzimanje Excel datoteke iz sustava. Metoda prima tri parametra: odgovor koji će se poslati korisniku (engl. *response*), putanja do Excel datoteke (engl. *path*) i vrsta podataka - *recoveryData* ili *medicineUsageHistory* (engl. *type*). Metoda prvo konstruira stvarnu putanju do Excel datoteke na temelju vrste podataka i predanog imena datoteke. Ova putanja uključuje direktorij u kojem se nalaze Excel datoteke povezane s vrstom podataka (recoveryData ili medicineUsageHistory).

```
@GetMapping("/download")
protected void doGet(HttpServletResponse response, @AuthenticationPrincipal ShowMeLoggedUser loggedUser)
        throws IOException {
    downloadFile(response, loggedUser.getFullName(), type: "recoveryData");
}
```

a)

```
private void downloadFile(HttpServletResponse response, String path, String type) throws IOException {
    String filePath = Objects.equals(type, b: "recoveryData")
            ? System.getProperty("user.dir") + "\\src\\main\\java\\com\\example\\allergy_identifier\\recoveryData\\"
                    + path + ".xlsx"
            : System.getProperty("user.dir")
                    + "\\src\\main\\java\\com\\example\\allergy_identifier\\medicineUsageHistory\\" + path
                    + ".xlsx";
    File file = new File(filePath);
    FileInputStream fis = new FileInputStream(file);

    response.setContentType("application/octet-stream");
    response.setContentLength((int) file.length());
    response.setHeader( s: "Content-Disposition", s1: "attachment; filename=\"" + file.getName() + "\"");

    byte[] buffer = new byte[4096];
    int bytesRead;
    while ((bytesRead = fis.read(buffer)) != -1) {
        response.getOutputStream().write(buffer, off: 0, bytesRead);
    }

    fis.close();
}
```

b)

Slika 4.28. *Prikaz poziva funckije preuzimanja Excel datoteke pomoću API-ja (a) i njen programski kod (b)*

Nakon konstrukcije stvarne putanje, otvara se FileInputStream koji će omogućiti čitanje podataka iz Excel datoteke. Zatim se postavljaju zaglavlja i informacije za HTTP odgovor. Tip sadržaja je postavljen je na "application/octet-stream", što označava generički binarni format, a dužina sadržaja postavljena je na veličinu datoteke kako bi se osiguralo ispravno preuzimanje. Također, postavlja se "Content-Disposition" zaglavlje kako bi se pregledniku reklo da se očekuje preuzimanje datoteke, a ne njeno otvaranje u pregledniku. Ime datoteke koje će korisnik dobiti prilikom preuzimanja bit će isto kao i ime izvorne datoteke. Nakon postavljanja zaglavlja, metoda čita podatke iz Excel datoteke i šalje ih u odgovor prema klijentu koristeći response.getOutputStream(). To se postiže čitanjem datoteke u manjim blokovima od 4096 bajtova te slanjem tih blokova u odgovor. Nakon što su svi podaci pročitani i poslani u odgovor, FileInputStream se zatvara kako bi se oslobodili resursi. Korisnik ima mogućnost i potpunog brisanja tijeka liječenja, čime se u potpunosti briše Excel datoteka tog pacijenta što je prikazano slikom 4.29.

```java
@GetMapping("/deleteExcelFile")
public String deleteExcelFile(@AuthenticationPrincipal ShowMeLoggedUser loggedUser) {
    UserExcelManager.deleteExcelFile(loggedUser.getFullName(), type: "recoveryData");
    return "redirect:/users-profile";
}
```

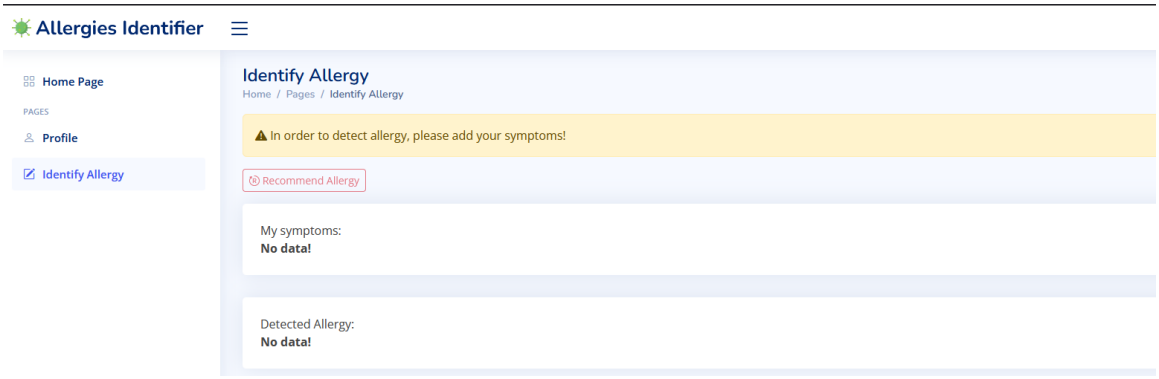
```java
public static void deleteExcelFile(String fileName, String type) {
    File file = Objects.equals(type, b: "recoveryData") ? new File( pathname: EXCEL_DIRECTORY + fileName + ".xlsx")
            : new File( pathname: EXCEL_MEDICINE_DIRECTORY + fileName + ".xlsx");
    if (file.exists()) {
        file.delete();
    }
}
```

Slika 4.29. *API programski kod brisanja Excel datoteke*

Metoda deleteExcelFile koristi informacije o vrsti podataka i imenu datoteke (*engl.* fileName) kako bi konstruirala putanju do odgovarajuće Excel datoteke. Ta putanja uključuje direktorij u kojem se nalaze Excel datoteke povezane s tom vrstom podataka, bilo recoveryData ili medicineData. Nakon što je putanja konstruirana, metoda provjerava postoji li datoteka na toj putanji. Ako je datoteka prisutna, ona se briše pozivom metode file.delete(). Ovaj postupak omogućuje čišćenje i uklanjanje Excel datoteka koje više nisu potrebne u aplikaciji.

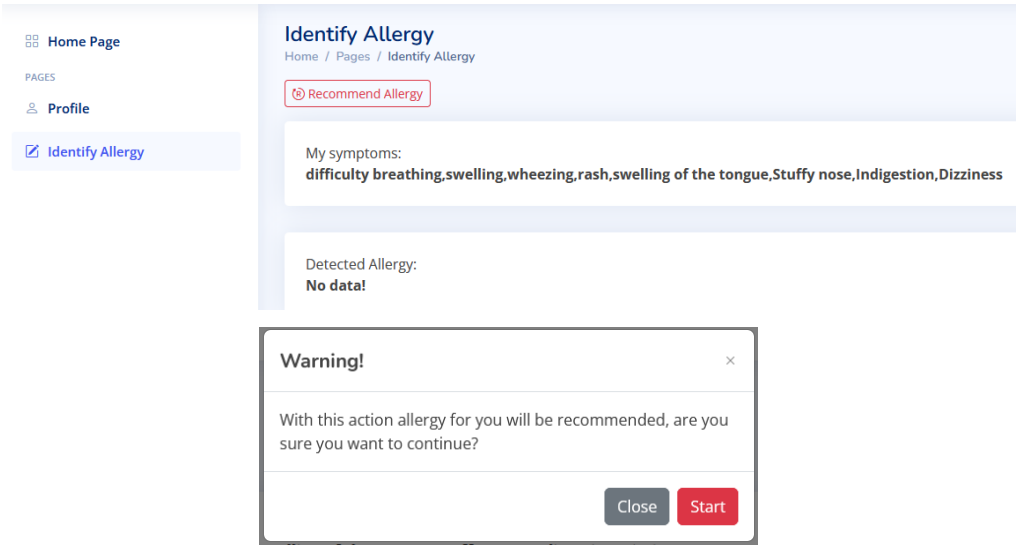
Na posljetku se dolazi do najvažnijeg dijela koji pacijent može odraditi, a to je dijagnoza alergije. Uvjet koji mora biti zadovoljen je taj da simptomi ne smiju biti prazni, inače dobivamo poruku prikazanu slikom 4.29. Ako je na stranici prisutan tekst No data!, tada se pomoću JavaScripta generira upozorenje pacijentu s porukom što je prikazano slikom 4.30.

```
if (url.endsWith("/pages-identify_alergy")) {
    const savedSymptoms : HTMLElement  = document.getElementById( elementId: 'mySymptoms');
    const recommendAllergy : HTMLElement  = document.getElementById( elementId: 'recommendAllergy');
    const emptySymptomsWarning : HTMLElement  = document.getElementById( elementId: 'emptySymptomsWarning');
    recommendAllergy.disabled = savedSymptoms.innerHTML.includes("No data!");

    if (savedSymptoms.innerText.includes("No data!")) {
        emptySymptomsWarning.classList.add('alert', 'alert-warning', 'bi', 'bi-exclamation-triangle-fill');
        emptySymptomsWarning.innerText = " In order to detect allergy, please add your symptoms!"
    }
}
```



Slika 4.30. *Prikaz programskog koda i izgled stranice praznih simptoma korisnika*

Dodavanjem simptoma, izgled stranice se mijenja te se gumb za dijagnozu alergije omogućava. Klikom na gumb, otvara se modalni prozor koji pokazuje upozorenje pacijentu da pokreće akciju dijagnoze alergije. Slika 4.31 prikazuje izgled stranice dijagnoze alergije prilikom prije odabranih simptoma te modal potvrde pokretanja dijagnoze alergije.



Slika 4.31. *Prikaz potvrde dijagnoze alergije*

Klikom na gumb Start, pokreće se API za dijagnozu alergije koji je prikazan slikom 4.33 koji poziva metodu dijagnosticiranja alergije prikazane slikom 4.34. Pomoću navedenog API-ja pokreće se algoritam za pronalazak alergije. Baza alergija i njenih simptoma pohranjena je unutar Excel datoteke iz koje algoritam čita potrebne podatke i dolazi do rješenja. Baza alergija prikazana je slikom 4.32. Baza je osmišljena na način da u koloni A Excel datoteke se nalaze imena alergija, dok su u koloni B navedeni svi simptomi koji se mogu pojaviti vezano za alergiju iz stupca A.

A	B
Peanut Allergy	vomiting, Stomach cramps, Indigestion, Diarrhea, Wheezing, Shortness of breath, Repetitive cough, Tightness in throat, Weak pulse, Pale or blue coloring of the skin, Dizziness, Confusion
Nut Allergy	difficulty breathing, wheezing, swelling of the tongue, swelling of the throat, difficulty talking, Confusion, becoming pale and floppy
Milk Allergy	hives, wheezing, vomiting, diarrhea, bloating, runny nose, eczema
Egg Allergy	hives, rash, nasal congestion, abdominal pain, nausea, vomiting, asthma symptoms, swelling of the lips, swelling of the tongue, swelling of the face
Soy Allergy	hives, itching, abdominal pain, diarrhea, vomiting, swelling, nasal congestion, wheezing, difficulty breathing
Wheat Allergy	swelling, itching, hives, eczema, abdominal pain, diarrhea, nausea, vomiting, difficulty breathing
Fish Allergy	hives, swelling, itching, abdominal pain, nausea, vomiting, diarrhea, nasal congestion, difficulty breathing
Shellfish Allergy	swelling, itching, hives, eczema, abdominal pain, nausea, vomiting, diarrhea, nasal congestion, difficulty breathing
Sesame Allergy	hives, itching, redness, abdominal pain, vomiting, diarrhea, nasal congestion, difficulty breathing
Sulfite Allergy	hives, flushing, itching, abdominal pain, diarrhea, difficulty breathing, wheezing, asthma symptoms
Pollen allergie	Itchy nose - eyes - ears and mouth, runny nose, Stuffy nose, Red and watery eyes, Swelling around the eyes, tired
Seasonal allergies	itching, runny nose, Stuffy nose, Temporary loss of smell, Headache, Facial pressure and pain
Grass pollen allergye	runny nose, blocked nose, sneezing, Itchy nose and eyes, throat irritation, Headache
Tree Pollen Allergy	sneezing, Itchy nose - eyes - ears and mouth, Red and watery eyes, Swelling around the eyes, Headache
Weed Pollen Allergy	Stuffy nose, runny nose, sneezing, Itchy nose, Post-nasal drip (the feeling of mucus moving down the back of your throat)
Dust Allergy	sneezing, runny nose, Red or Wattery eyes, nasal congestion, Itchy nose - eyes - ears and mouth, Post-nasal drip (the feeling of mucus moving down the back of your throat), Repetitive cough, Facial pressure and pain
Allergy to pets	sneezing, runny, Stuffy nose, Facial pressure and pain, Repetitive cough, hives, rash
Mold allergy	nasal congestion, runny nose, sneezing, irritated eyes, Repetitive cough, wheezing, itchy throat
Allergy to chemical substances	Increased heart rate, chest pain, sweating, difficulty breathing, fatigue, Dizziness

Slika 4.32. *Prikaz baze podataka alergija i njihovih simptoma*

```
// recommendation system for finding allergy and saving on logged user
@PostMapping("/identifyUserAllergy")
public String identifyUserAllergy(@AuthenticationPrincipal ShowMeLoggedUser loggedUser, User user) {
    String diagnosedAllergy = recommendationSystemUtils.recommendAllergy(loggedUser.getSymptoms());
    editUserProperty(user, property: "allergy", loggedUser.setDiagnosedAllergy(diagnosedAllergy), loggedUser);
    return "redirect:/pages-identify_alergy?success";
}
```

Slika 4.33. *Prikaz API-ja za dijagnozu alergije*

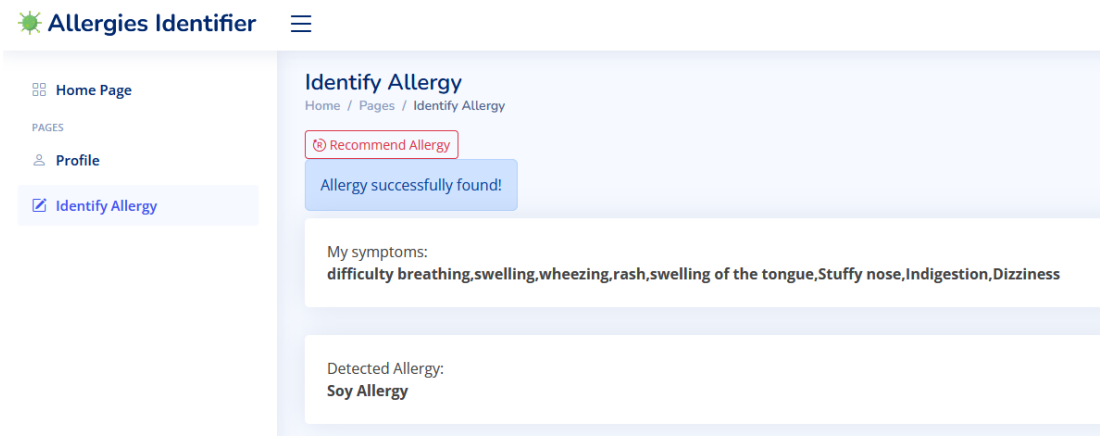
```java
public String recommendAllergy(String userSymptoms) {
    String filePath = System.getProperty("user.dir") +
            "\\src\\main\\java\\com\\example\\allergy_identifier\\recommendationData\\allergyData\\allergyes.xlsx";
    String correctValue = "";
    FileInputStream fis = null;
    Workbook workbook = null;

    try {
        fis = new FileInputStream(filePath);
        workbook = WorkbookFactory.create(fis);
        Sheet sheet = workbook.getSheetAt( i: 0);
        List<Double> matchPercentages = new ArrayList<>();

        for (Row row : sheet) {
            Cell symptomsCell = row.getCell( i: 1);
            if (symptomsCell != null) {
                String symptoms = symptomsCell.getStringCellValue();

                String[] userSymptomsArray = userSymptoms.split( regex: ",");
                String[] rowSymptomsArray = symptoms.split( regex: ",");

                int matchCount = 0;
                for (String userSymptom : userSymptomsArray) {
                    for (String rowSymptom : rowSymptomsArray) {
                        if (userSymptom.trim().contains(rowSymptom.trim())) {
                            matchCount++;
                            break;
                        }
                    }
                }
                DecimalFormat df = new DecimalFormat( pattern: "0.00");
                int totalSymptoms = rowSymptomsArray.length;
                double matchPercentage = matchCount * 100.0 / totalSymptoms;
                matchPercentages.add(Double.valueOf(df.format(matchPercentage)));
            }
        }
        System.out.println(matchPercentages);
        int bestRowIndex = matchPercentages.indexOf(Collections.max(matchPercentages));
        Row bestRow = sheet.getRow(bestRowIndex);
        Cell correctAllergyCell = bestRow.getCell( i: 0);
        correctValue = correctAllergyCell.getStringCellValue();
    } catch (IOException | InvalidFormatException e) {
        e.printStackTrace();
    } finally {
        try {
            if (fis != null) {
                fis.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    return correctValue;
}
```

Slika 4.34. *Prikaz metode određivanja alergije*

Metoda recommendAllergy ima zadatak preporučiti alergiju korisniku na temelju unesenih simptoma. Metoda prima simptome pacijenta - simptome koje je korisnik unio. Metoda prvo konstruira putanju

do Excel datoteke koja sadrži podatke o alergijama. Zatim otvara FileInputStream i kreira radnu knjigu iz Excel datoteke. Podaci o alergijama nalaze se u prvoj listi u radnoj knjizi. Metoda prolazi kroz svaki redak u listi i dohvaća ćeliju koja sadrži simptome određene alergije. Zatim uspoređuje te simptome s simptomima koje je korisnik unio. Svaki simptom korisnika provjerava se protiv svih simptoma u retku alergije. Ako postoji podudaranje, povećava se brojač podudaranja. Na temelju broja podudaranja i ukupnog broja simptoma alergije, izračunava se postotak podudaranja. Nakon što su izračunati postoci podudaranja za sve alergije, odabire se najveći postotak. Pronalazi se redak s najvećim postotkom i dohvaća se ćelija koja sadrži naziv alergije. Metoda vraća preporučenu alergiju na temelju simptoma koje je korisnik unio. Na samome kraju, ti podaci se pomoću već objašnjene metode editUserProperty šalju te se dijagnosticirana alergija sprema na pacijenta.

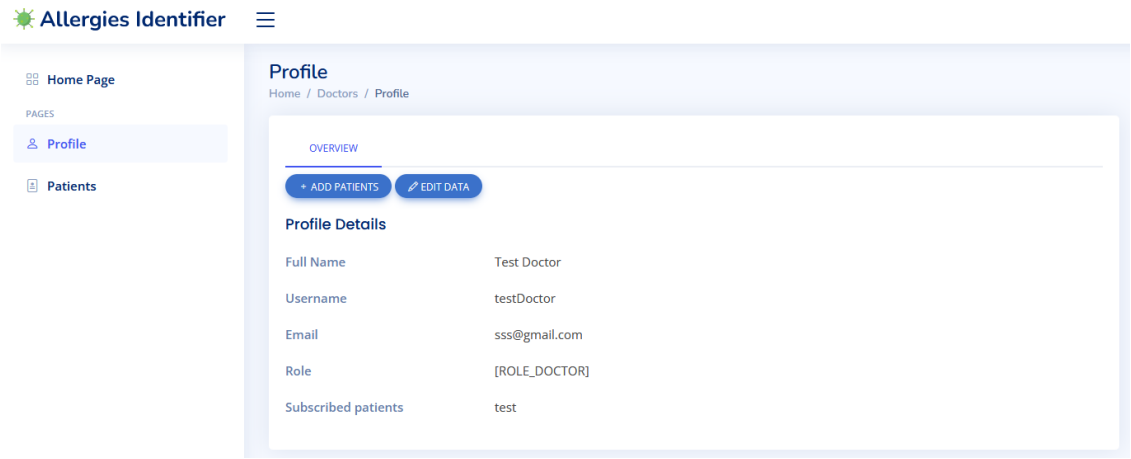
Nakon uspješnog pronalaska alergije, korisnik dobiva informaciju o pronađenoj alergiji što je prikazano slikom 4.35. Taj podatak ne mora u potpunosti biti točan, na liječniku je da potvrdi rezultat koji je generirao algoritam.



Slika 4.35. *Prikaz dijagnosticirane alergije*

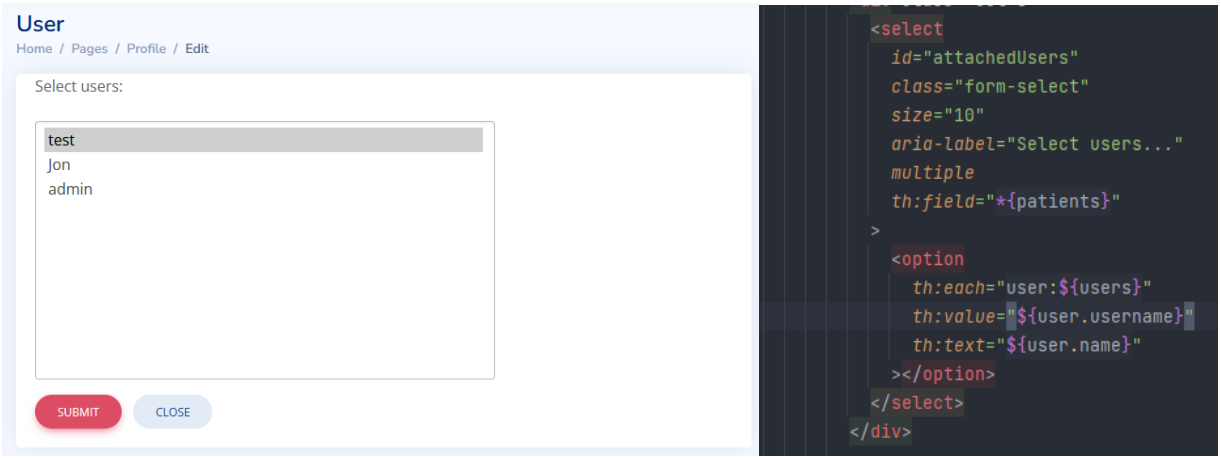
4.2.3. Mogućnosti prijavljenog liječnika

Liječnika kreira ADMIN, a sami postupak će biti objašnjen u potpoglavlju 4.2.4. Autorizacijom liječnika, dolazi na početnu stranicu koja je za sve uloge ista. Navigacijom na profil, ima pristup svojim podacima i mogućnosti dodavanja pacijenata koje želi liječiti. Prikaz profila liječnika priazan je slikom 4.36.



Slika 4.36. *Prikaz profila liječnika*

HTML kod korišten za izgled profila i samih atributa liječnika je identičan načinu na koji se prikazuju podaci pacijenta, tako da se način prikaza podataka može pogledati prije navedenoj slici 4.13 gdje je također objašnjeno kako se podaci prikazuju. Liječnik ima mogućnost dodavanja neograničenog broja pacijenata koje želi liječiti što se može vidjeti prema slici 4.37.



Slika 4.37. *Prikaz odabira pacijenata i programski kod za prikaza svih pacijenata*

Potvrdom odabranih pacijenata, klikom na gumb potvrde, okida se API za spremanje pacijenata prikazan slikom 4.38 te se unutar API-ja šalju odabrani pacijenti. Kao što je prikazano na slici 4.37 th:field="*{patients}" je atribut koji ima liječnik, te se okidanjem forme u tu varijablu u bazu spremaju pacijenti.

```java
@PostMapping("/attachPatient/add")
public String attachPatients(Doctor doctor, @AuthenticationPrincipal ShowMeLoggedDoctor showMeLoggedDoctor) {
    Doctor updatedDoctor = updateLoggedDoctorAttributes(doctor, showMeLoggedDoctor);
    setAndCheckAnyRole( roleName: "ROLE_DOCTOR", updatedDoctor);
    doctorService.updateDoctor(updatedDoctor);
    return "redirect:/doctor-profile?success";
}
```

```java
public Doctor updateLoggedDoctorAttributes(Doctor doctor,
        @AuthenticationPrincipal ShowMeLoggedDoctor loggedDoctor) {
    Doctor updatedDoctor = new Doctor();

    loggedDoctor.setName(doctor.getName());
    loggedDoctor.setUsername(doctor.getUsername());
    loggedDoctor.setEmail(doctor.getEmail());
    loggedDoctor.setPassword(doctor.getPassword());
    loggedDoctor.setPatients(doctor.getPatients());

    updatedDoctor.setId(loggedDoctor.getId());
    updatedDoctor.setName(loggedDoctor.getFullName());
    updatedDoctor.setUsername(loggedDoctor.getUsername());
    updatedDoctor.setEmail(loggedDoctor.getEmail());
    updatedDoctor.setPatients(loggedDoctor.getPatients());
    updatedDoctor.setPassword(passwordEncoder.encode(loggedDoctor.getPassword()));
    return updatedDoctor;
}
```

Slika 4.38. *Programski kod za prikaz spremanja pacijenata*

Metoda updateLoggedDoctorAttributes služi za ažuriranje atributa trenutno prijavljenog liječnika. Kao ulazne parametre prima objekt tipa Doctor, koji sadrži nove vrijednosti atributa liječnika, te objekt loggedDoctor, koji predstavlja trenutno prijavljenog liječnika. U metodi se stvara novi objekt updatedDoctor, koji će sadržavati ažurirane podatke. Zatim se atributi trenutno prijavljenog liječnika ažuriraju s novim vrijednostima iz objekta Doctor. Nakon toga, ažurirani atributi iz objekta loggedDoctor kopiraju se u objekt updatedDoctor. Metoda zatim vraća objekt updatedDoctor s ažuriranim podacima. Druga metoda, attachPatients, povezana je s POST zahtjevom na ruti /attachPatient/add. Kao ulazne parametre prima objekt Doctor i objekt showMeLoggedDoctor, koji predstavlja trenutno prijavljenog liječnika. U metodi se prvo poziva metoda updateLoggedDoctorAttributes s predanim objektom Doctor i trenutno prijavljenim liječnikom kako bi se ažurirali atributi liječnika s novim vrijednostima. Zatim se postavlja uloga liječnika na ROLE_DOCTOR kako bi se osiguralo da liječnik ima potrebne ovlasti. Nakon ažuriranja atributa i postavljanja uloge, koristi se doctorService kako bi se ažurirali podaci o liječniku u bazi podataka. Na kraju, metoda vraća preusmjeravanje na "/doctor-profile?success", što označava uspješno izvršenu operaciju. Liječnik zatim ima ponuđeni izbornik pacijenata, gdje je lista svih pacijenata koje je on

prethodno odabrao da će liječiti. Način prikaza pacijenata i njihovo dohvačanje prikazano je slikom 4.39.

```java
@GetMapping("/doctors-patients")
public String doctorsPatients(@AuthenticationPrincipal ShowMeLoggedDoctor showMeLoggedDoctor, Model model) {
    String username = showMeLoggedDoctor.getUsername();
    Doctor doctor = doctorService.findDoctorByUsername(username);
    if (doctor.getPatients() == null) {
        return "pages-error-404";
    }
    String patients = doctor.getPatients();
    String[] split = patients.split( regex: ",");
    List<User> myPatients = findDoctorsPatients(split);
    model.addAttribute( attributeName: "attachedPatients", myPatients);
    return "doctors-patients";
}
```

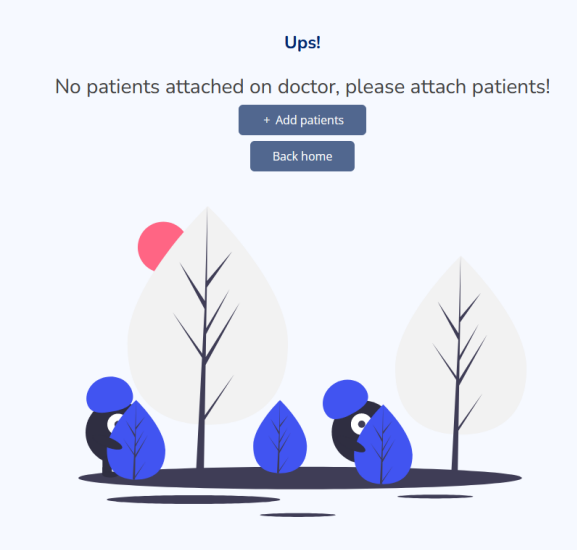
```java
1 usage
private List<User> findDoctorsPatients(String[] array) {
    List<User> users = new ArrayList<>();
    for (String username : array) {
        User user = userService.findUserByUsername(username);
        users.add(user);
    }
    return users;
}
```

Slika 4.39. *Programski kod za način prikazivanja i pronalaženja pacijenta liječnika*

Metoda findDoctorsPatients prima niz korisničkih imena kao ulazni parametar i stvara praznu listu objekata tipa User pod imenom users. Zatim, za svako korisničko ime u predanom nizu, koristi userService.findUserByUsername(username) kako bi pronašla korisnika s tim korisničkim imenom te ga dodaje u listu users. Nakon što su svi korisnici pronađeni, metoda vraća tu listu. Metoda doctorsPatients je mapirana na GET zahtjev na ruti /doctors-patients. Prima objekt showMeLoggedDoctor, koji predstavlja trenutno prijavljenog liječnika, i objekt model za slanje podataka u predložak. Prvo se dohvaća korisničko ime trenutno prijavljenog liječnika pomoću showMeLoggedDoctor.getUsername().

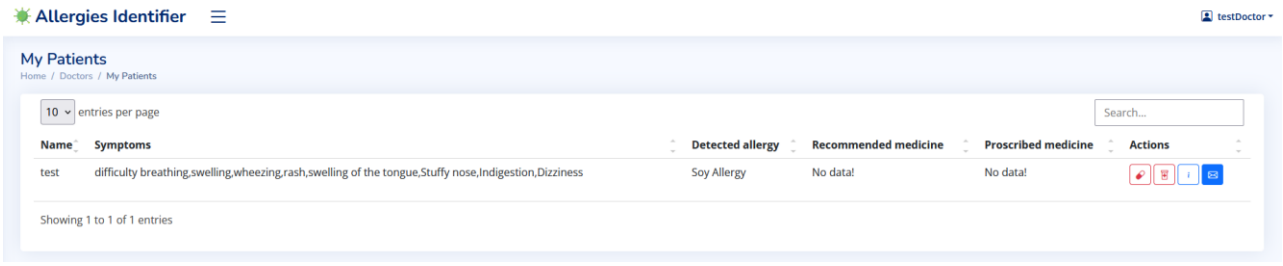
Zatim se koristi doctorService.findDoctorByUsername(username) kako bi se pronašao liječnik s tim korisničkim imenom. Ako liječnik nema pacijente (dohvaćeno iz doctor.getPatients()), prikazuje se predložak pages-error-404 prikazan slikom 4.40, koji omogućava liječniku da doda pacijente koje želi liječiti. Ako liječnik ima pacijente, njihova korisnička imena su odvojena zarezima u stringu patients. Koristi se split(",") kako bi se dobilo polje stringova s korisničkim imenima pacijenata. Zatim se poziva metoda findDoctorsPatients(split) s tim nizom korisničkih imena kako bi se pronašli svi pacijenti liječnika. Pronađeni pacijenti (objekti tipa User) dodaju se u model s atributom

attachedPatients. Na kraju, metoda vraća predložak "doctors-patients", koji prikazuje listu pacijenata liječnika na odgovarajućoj stranici. Metoda doctorsPatients služi za prikazivanje popisa pacijenata trenutno prijavljenog liječnika, dok se metoda findDoctorsPatients koristi za pronalaženje korisnika (pacijenata) na temelju njihovih korisničkih imena.



Slika 4.40. *Prikaz stranice kada liječnik nema označenih pacijenata*

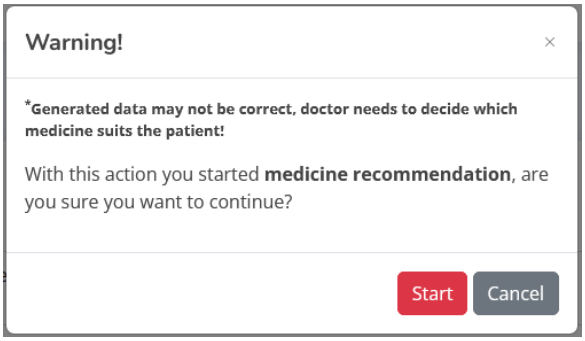
Nakon uspješnog odabira pacijenata koji se žele pratiti, liječnik klikom na izbornik dobiva sve informacije o tim pacijentima gdje ima mogućnost preporuke lijeka, potvrde lijeka, pregleda cjelokupnog stanja pacijenta i sami kontakt pacijenta pomoću elektroničke pošte. Pomoću programskog koda na slici 4.39 generira se stranica prikazana slikom 4.41. Liječnik u tablici vidi pacijente koje je označio te ima mogućnost pretraživanja tablice po bilo kojemu tekstu.



Slika 4.41. *Prikaz izgleda tablice pacijenata*

Pomoću prvog gumba unutar kolone akcije, liječnik ima mogućnost preporuke lijeka za pacijenta čime se okida API prikazan na slici 4.43 koji poziva programski kod prikazan slikom 4.44. Klikom

na taj gumb, otvara se prozor kojim liječnik potvrđuje odabranu akciju, a prikaza je slikom 4.42.



Slika 4.42. *Prikaz potvrde preporuke lijeka*

```
@PostMapping("/doctors-patients/proscribeMedicine")
public String proscribeMedicine(@RequestParam("userId") Long userId) throws Exception {
    User user = userService.getUserById(userId);
    List<String> allMedicines = new ArrayList<>();
    String detectedAllergy = user.getDetectedAllergy();
    String age = user.getAge();
    String pregnant = user.getIsPregnant();
    String weight = user.getWeight();
    // Recomm system implementation
    List<Medicine> medicine = DataLoader.findCorrespondingMedicine(detectedAllergy, age, pregnant, weight);
    for (Medicine singleMedicine : medicine) {
        allMedicines.add("\n" + "Medicine name: " + singleMedicine.getName() + "\n" + "Medicine description for "
                + singleMedicine.getName() + ": " + singleMedicine.getDescription());
    }

    user.setId(userId);
    user.setDetectedAllergy(user.getDetectedAllergy());
    user.setProscribedMedicine(user.getProscribedMedicine());
    user.setSymptoms(user.getSymptoms());
    user.setEmail(user.getEmail());
    user.setName(user.getName());
    user.setUsername(user.getUsername());
    user.setPassword(user.getPassword());
    user.setMedicationUsage(user.getMedicationUsage());
    user.setRecommendedMedicine(allMedicines.toString());
    user.setRecoveryProcess(user.getRecoveryProcess());
    user.setDoctor(user.getDoctor());
    user.setAge(user.getAge());
    user.setIsPregnant(user.getIsPregnant());
    user.setWeight(user.getWeight());
    user.setDoctor(user.getDoctor());
    userService.updateUser(user);
    return "redirect:/doctors-patients";
}
```

Slika 4.43. *Prikaz API programskog koda za preporuku lijeka*

```
2 usages
public static List<Medicine> findCorrespondingMedicine(String detectedAllergy, String age, String pregnant,
        String weight) throws Exception {
    List<Medicine> allData = new ArrayList<>(Collections.emptyList());
    Workbook wb = new Workbook( file: System.getProperty("user.dir")
            + "\\src\\main\\java\\com\\example\\allergy_identifier\\recommendationData\\medicineData\\medicine.xlsx");
    WorksheetCollection collection = wb.getWorksheets();

    for (int worksheetIndex = 0; worksheetIndex < collection.getCount(); worksheetIndex++) {
        Worksheet worksheet = collection.get(worksheetIndex);
        int rows = worksheet.getCells().getMaxDataRow();
        for (int i = 0; i < rows + 1; i++) {
            String readAge = worksheet.getCells().get(i,  column: 3).getStringValue();
            String readPregnant = worksheet.getCells().get(i,  column: 4).getStringValue();
            String readWeight = worksheet.getCells().get(i,  column: 5).getStringValue();
            if (Objects.equals(readPregnant,  b: "TRUE")) {
                if (isMedicineUsable(detectedAllergy, i, worksheet, age, readAge, weight, readWeight)) {
                    collectFoundData(worksheet, i, allData);
                }
            } else {
                if (!Objects.equals(pregnant,  b: "true")
                        && isMedicineUsable(detectedAllergy, i, worksheet, age, readAge, weight, readWeight)) {
                    collectFoundData(worksheet, i, allData);
                }
            }
        }
    }
    return allData;
}
```

```
2 usages
public static boolean isMedicineUsable(String detectedAllergy, int index, Worksheet worksheet, String age,
        String readAge, String weight, String readWeight) {
    return detectedAllergy != null
            && detectedAllergy.contains((String) worksheet.getCells().get(index,  column: 0).getValue())
            && worksheet.getCells().get(index,  column: 0).getValue() != null
            && Integer.parseInt(age) >= Integer.parseInt(readAge)
            && Integer.parseInt(weight) >= Integer.parseInt(readWeight);
}
```

Slika 4.44. *Prikaz programskog koda za preporuku lijeka*

Klikom na gumb start pokreće se API za preporuku lijeka sa slike 4.43 koja poziva metodu findCorrespondingMedicine sa slike 4.44. Metoda proscribeMedicine je mapirana na POST zahtjev na ruti /doctors-patients/proscribeMedicine. Prima parametar userId koji predstavlja identifikacijski broj korisnika (user) kojem će propisati lijekove koji se dobije iz odziva stranice klikom na gumb Start. Metoda pomoću ovog ID-a dohvaća korisnika iz baze podataka putem userService.getUserById(userId). Nakon toga, stvara se prazna lista allMedicines koja će sadržavati informacije o propisanim lijekovima. Zatim se dohvaćaju različiti atributi korisnika, kao što su dijagnosticirana alergija, godine, trudnoća i težina, kako bi se koristili u preporučivanju odgovarajućih lijekova. Sljedeći korak je implementacija sustava preporuke lijekova koja se koristi se za pronalaženje lijekova temeljem navedenih atributa korisnika.

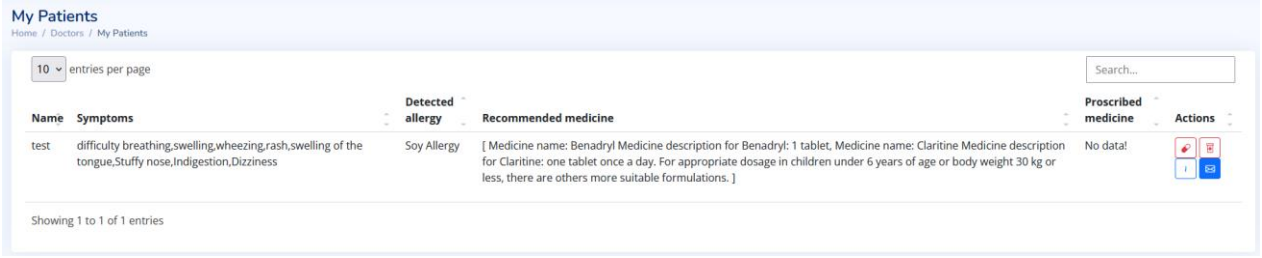
Metoda findCorrespondingMedicine je statička metoda koja prima četiri parametra: dijagnosticirana alergija, godine, trudnoća i težina. Ova metoda koristi se za pronalaženje lijekova koji odgovaraju zadanim kriterijima. Metoda prvo stvara praznu listu allData koja će sadržavati pronađene lijekove. Zatim otvara Excel datoteku (medicine.xlsx) koja sadrži informacije o različitim lijekovima i njihovim svojstvima. Nakon što je otvorena Excel datoteka, metoda prolazi kroz sve radne listove u datoteci. Za svaku radnu listu, prolazi kroz sve retke kako bi provjerila podatke o lijekovima. Metoda dohvaća vrijednosti za dob, trudnoću i težinu za svaki redak i uspoređuje ih s zadanim vrijednostima (dob, trudnoća, težina) te pretražuje za odgovarajuće alergije u skladu s metodom isMedicineUsable. Metoda isMedicineUsable je statička metoda koja služi za provjeru korisnikove sposobnosti korištenja određenog lijeka. Metoda prima nekoliko parametara: dijagnosticirana alergija, index, radni list, dob pacijenta, pročitana dob iz Excel datoteke za određeni lijek, težina pacijenta i pročitana težina iz Excel datoteke za određeni lijek. Metoda prvo provjerava da li je dijagnosticirana alergija različita od null i da li sadrži vrijednost iz prve ćelije (stupca) na indeksu index radne liste. Također, provjerava se da li ta vrijednost nije null. Ove provjere se koriste kako bi se osiguralo da korisnik ima detektiranu alergiju i da se alergija poklapa s imenom lijeka iz radne liste. Zatim se provjerava da li je korisnikova dob veća ili jednaka od dobne granice koja se nalazi u radnoj listi na istom indeksu. Također, provjerava se da li je korisnikova težina veća ili jednaka od granice težine koja se nalazi u radnoj listi na istom indeksu. Ako su sve provjere zadovoljene, metoda vraća *true*, što znači da je korisnik sposoban koristiti taj lijek. U suprotnom, metoda vraća *false*, što znači da korisnik nije sposoban koristiti taj lijek. Ako je pronađen odgovarajući lijek za zadane kriterije, metoda poziva collectFoundData kako bi prikupila informacije o lijeku i dodala ga u listu allData. Kada su sve radne liste i reci obrađeni, metoda vraća listu allData koja sadrži sve pronađene lijekove koji odgovaraju zadanim kriterijima. Ova metoda predstavlja sustav preporuke lijekova temeljen na unaprijed definiranim svojstvima lijekova i korisničkim kriterijima (dob, trudnoća, težina, alergije) kako bi se pronašli odgovarajući lijekovi za svakog korisnika. Nakon što su pronađeni odgovarajući lijekovi, svaki pojedini lijek se dodaje u listu allMedicines zajedno s opisom, a sve informacije se formatiraju u obliku pogodnom za prikaz. Zatim slijedi postavljanje ažuriranih vrijednosti korisnika. Nakon što su promijenjeni atributi postavljeni, korisnik se ažurira u bazi podataka pozivom userService.updateUser(user). Na kraju, korisnik se preusmjerava na stranicu /doctors-patients putem return redirect:/doctors-patients.

Baza lijekova prikazana je slikom 4.45 i sastavljena je u Excel file, gdje se u svakome retku nalazi alergija te njen odgovarajući lijek. Da bi se taj lijek propisao, mora zadovoljiti kolonu D, koja predstavlja broj godina od kojeg pacijent smije koristit taj lijek uključujući i taj broj. E stupac predstavlja da li taj lijek smije koristiti trudnica, a kolona F, predstavlja s koliko kilograma pacijenti smiju koristiti taj lijek. Pacijent ako ima veću kilažu ili godine, smije koristiti taj lijek. Također se uz to gleda i kolona A, koja predstavlja prije dijagnosticiranu alergiju. Kada se svi parametri poslože dobiva se rješenje iz kolone B koje predstavlja konačni lijek, a kolona C se također generira uz propisani lijek kao uputa liječniku kako i na koji način bi pacijent mogao koristiti taj lijek.

	A	B	C	D	E	F
1	Peanut Allergy	EpiPen	The usual dose is 0.3 mg for intramuscular administration	18	FALSE	30
2	Peanut Allergy	EpiPen	For these patients, an EpiPen Jr auto-injector device containing 0.15 mg is available adrenaline per dose	3	FALSE	15
3	Nut Allergy	Claritine	one tablet once a day. For appropriate dosage in children under 6 years of age or body weight 30 kg or less, there are others more suitable formulations.	6	TRUE	20
4	Milk Allergy	EpiPen	The usual dose is 0.3 mg for intramuscular administration	18	TRUE	30
5	Milk Allergy	EpiPen	For these patients, an EpiPen Jr auto-injector device containing 0.15 mg is available adrenaline per dose	3	TRUE	15
6	Egg Allergy	Piriton	Take one tablet every 4 to 6 hours as needed. Do not take more than 6 tablets in 24 hours.	12	TRUE	30
7	Egg Allergy	Piriton	Talk to your doctor or pharmacist before you take this medicine as you may be more likely to get side effects including confusion and you may need to take a lower daily dose. Maximum daily dose: 3 tablets (12 mg) in any 24 hours	18	TRUE	50
8	Egg Allergy	Zyrtec	at daily doses of 5 or 10 mg	11	TRUE	30
9	Egg Allergy	Rhinocort	64 mcg (one spray per nostril) once daily	6	TRUE	20
10	Egg Allergy	Rhinocort	256 mcg (four sprays per nostril) once daily	12	TRUE	50
11	Soy Allergy	Benadryl	1 to 2 tablets	12	TRUE	50
12	Soy Allergy	Benadryl	1 tablet	6	TRUE	15
13	Soy Allergy	Chlor-Trimeton	adults and children 12 years and over 1 tablet. Do not exceed 6 tablets in 24 hours	12	FALSE	50
14	Soy Allergy	Chlor-Trimeton	children 6 to under 12 years 1/2 tablet (break tablet in half). Do not exceed 3 whole tablets in 24 hours.	6	FALSE	15
15	Soy Allergy	Claritine	one tablet once a day. For appropriate dosage in children under 6 years of age or body weight 30 kg or less, there are others more suitable formulations.	6	TRUE	20
16	Wheat Allergy	EpiPen	The usual dose is 0.3 mg for intramuscular administration - two injectible doses needed to carry by yourself	18	FALSE	30
17	Wheat Allergy	EpiPen	For these patients, an EpiPen Jr auto-injector device containing 0.15 mg is available adrenaline per dose - tv	3	FALSE	15
18	Fish Allergy	Claritine	one tablet once a day. For appropriate dosage in children under 6 years of age or body weight 30 kg or less, there are others more suitable formulations.	6	TRUE	20
19	Shellfish Allergy	EpiPen	The usual dose is 0.3 mg for intramuscular administration	18	FALSE	30
20	Shellfish Allergy	EpiPen	For these patients, an EpiPen Jr auto-injector device containing 0.15 mg is available adrenaline per dose	3	FALSE	15
21	Shellfish Allergy	Auvi-Q	AUVI-Q 0.3 mg	10	FALSE	30
22	Shellfish Allergy	Auvi-Q	AUVI-Q 0.15 mg	6	FALSE	15
23	Shellfish Allergy	Auvi-Q	AUVI-Q 0.1 mg	2	FALSE	7.5
24	Sesame Allergy	OIT	consult with doctor what dosage is needed	6	TRUE	30
25	Sulfite Allergy	Bleph-10	consult with doctor what dosage is needed	14	TRUE	30
26	Pollen allergie	Allegra	The recommended dose is one tablet (120 mg) per day. Take the tablet before a meal, with water. This med	12	FALSE	25
27	Pollen allergie	Xyzal	10 ml of solution	12	TRUE	25
28	Pollen allergie	Xyzal	2.5 ml of solution twice a day	2	TRUE	10

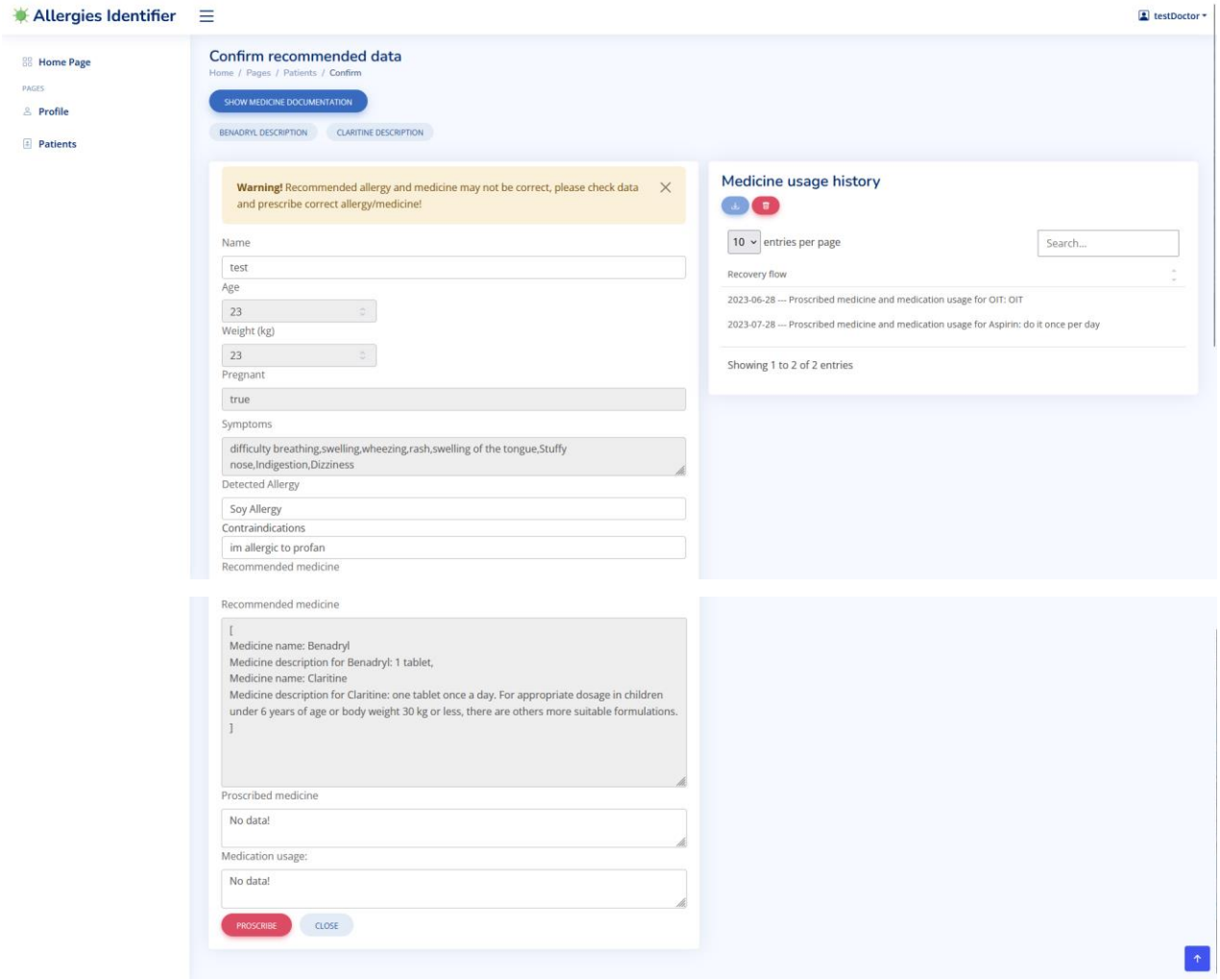
Slika 4.45. *Prikaz baze podataka preporučivanja lijekova*

Nakon uspješnog pronalaženja lijekova, liječnik dobiva informaciju koji lijekovi mogu doći u obzir tijekom liječenja što je prikazano slikom 4.46.



Slika 4.46. *Prikaz preporučenih lijekova*

Klikom na idući gumb, liječnik odlazi u finalnu potvrdu liječenja gdje pregledava sve prije generirane podatke i donosi završnu odluku. Prozor odluke liječnika prikazan je slikom 4.47.



Slika 4.47. *Prikaz potvrde tijeka liječenja*

Liječniku se nude i dokumentacija preporučenih lijekova kako bi mogao detaljnije proučiti i donijeti odluku. API preuzimanja dokumentacije lijekova prikazan je slikom 4.48. Pozivanjem API-a otvaranjem stranice, liječnik ima mogućnost preuzimanja tih dokumenata. Pregledavanjem generiranog sadržaja, liječnik može izmijeniti dijagnosticiranu alergiju, prepisati lijek koji on misli da je točan i dati uputu kako koristiti taj lijek. Nakon klika na gumb propiši lijek, okida se API koji ažurira podatke pacijenta te njemu samome na profil šalje podatke koje je liječnik unio.

```java
@PostMapping("/downloadFile")
public void downloadFile(@RequestParam("filePath") String filePath, HttpServletResponse response) {
    File file = new File(filePath);

    response.setContentType("application/octet-stream");
    response.setHeader( s: "Content-Disposition", s1: "attachment; filename=\"" + file.getName() + "\"");
    response.setContentLength((int) file.length());
    try (InputStream inputStream = Files.newInputStream(file.toPath())) {
        FileCopyUtils.copy(inputStream, response.getOutputStream());
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Slika 4.48. *Prikaz API za preuzimanje dokumentacije lijekova*

Liječnik također ima pregled tijeka liječenja svakog pacijenta, klikom na gumb detalji u tablici pacijenata. Unutar toga vidi svaku informaciju koju je pacijent zabilježio. Također je implementirano praćenje tijeka liječenja pomoću grafa. Korišten je Chart.js, biblioteka koja nudi mogućnost rada s grafovima. Klikom liječnika na gumb za prikaz grafa, otvara se graf koji na X osi prikazuje datume, a na Y osi broj simptoma tog datuma. Prema tim podacima crta se graf. Za broj simptoma koje je pacijent odabrao, uzima se njihov ukupan broj te se na temelju tog broja crta crvena linija koja ukazuje početni broj simptoma. Ako graf prijeđe crvenu liniju, liječnik odmah vidi da lijek koji je propisao tog datuma ne odgovara pacijentu, dok ako je graf ispod linije, stanje pacijenta se ne pogoršava. Slika 4.49 prikazuje dohvaćanje svih potrebnih varijabli za iscrtavanje grafa, slika 4.50 prikazuje programski kod iscrtavanja grafa i slikom 4.51 prikazan je iscrtani graf.

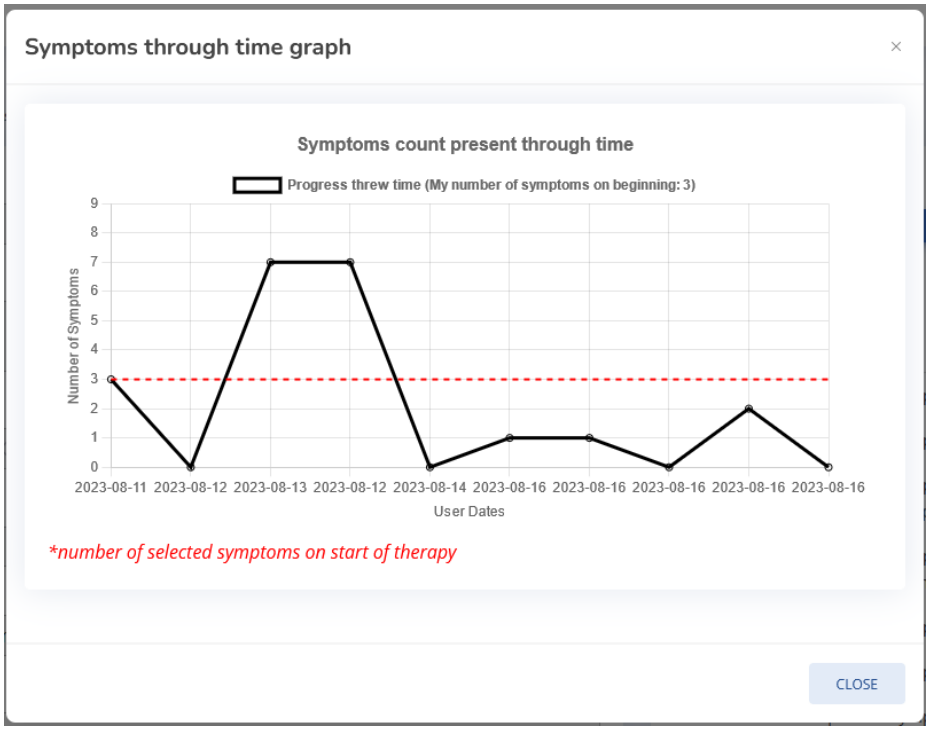
```java
@GetMapping("/doctors-patients/doctorAccessPatientDetails/{id}")
public String detailsDoctorsUser(@PathVariable Long id, Model model) {
    User currentUser = userService.getUserById(id);
    String name = currentUser.getName();
    List<String> recoveryList = UserExcelManager.readDataFromExcel(name, type: "recoveryData");
    List<String> medicineHistoryData = UserExcelManager.readDataFromExcel(currentUser.getName(), type: "medicineHistory");
    List<String> userDates = dataLoader.getDates(currentUser.getUsername());
    List<Integer> numberOfUserSymptoms = dataLoader.getNumberOfSymptoms(currentUser.getUsername());
    List<String> selectedSymptomsOnStart = dataLoader.splitValues(currentUser.getSymptoms());
    int numberOfSelectedSymptomsOnStart = selectedSymptomsOnStart.size();
    model.addAttribute( attributeName: "recoveryList", recoveryList);
    model.addAttribute( attributeName: "user", currentUser);
    model.addAttribute( attributeName: "userDates", userDates);
    model.addAttribute( attributeName: "numberOfUserSymptoms", numberOfUserSymptoms);
    model.addAttribute( attributeName: "numberOfSelectedSymptomsOnStart", numberOfSelectedSymptomsOnStart);
    model.addAttribute( attributeName: "medicineHistoryData", medicineHistoryData);
    return "doctorAccessPatientDetails";
}
```

Slika 4.49. *Prikaz dohvaćanja varijabli grafa*

```
<script th:inline="javascript">
  var userDates = /*[[${userDates}]]*/ [];
  var numberOfUserSymptoms = /*[[${numberOfUserSymptoms}]]*/ [];
  var maxNumberOfSymptoms = Math.max(...numberOfUserSymptoms) + 2;
  var numberOfSelectedSymptomsOnStart = /*[[${numberOfSelectedSymptomsOnStart}]]*/ [];

  // Custom plugin to draw a horizontal line on the chart
  Chart.register({
    id: "customLine",
    afterDraw: function (chart, args, options) {
      var ctx = chart.ctx;
      var scale = chart.scales["y"];
      var yValue = scale.getPixelForValue(numberOfSelectedSymptomsOnStart);

      ctx.save();
      ctx.beginPath();
      ctx.moveTo(chart.chartArea.left, yValue);
      ctx.strokeStyle = "red";
      ctx.lineWidth = 2;
      ctx.setLineDash([5, 5]);
      ctx.lineTo(chart.chartArea.right, yValue);
      ctx.stroke();
      ctx.restore();
    },
  });

  // Create a line chart
  var ctx = document.getElementById("lineChart").getContext("2d");
  var lineChart = new Chart(ctx, {
    type: "line",
    data: {
      labels: userDates,
      datasets: [
        {
          label: "Number of Symptoms (Max: " + numberOfSelectedSymptomsOnStart + ")",
          data: numberOfUserSymptoms,
          fill: false,
```

```
    options: {
      responsive: true,
      scales: {
        x: {
          display: true,
          title: {
            display: true,
            text: "User Dates",
          },
        },
        y: {
          display: true,
          title: {
            display: true,
            text: "Number of Symptoms",
          },
          suggestedMax: maxNumberOfSymptoms,
        },
      },
      plugins: {
        title: {
          display: true,
          text: "Symptoms count present through time",
          font: {
            weight: "bold",
            size: 16,
          },
        },
        legend: {
          display: true,
          labels: {
            font: {
              weight: "bold",
            },
          },
        },
        customLine: {},
      },
    },
  });
</script>
```

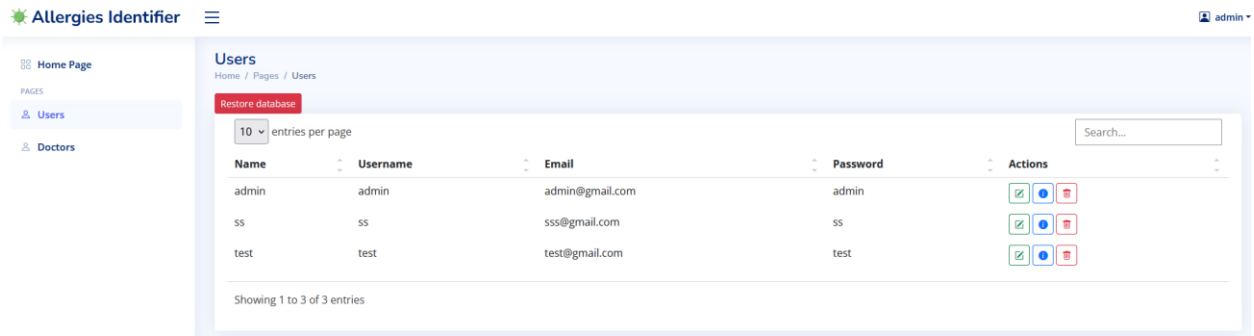
Slika 4.50. *Prikaz programskog koda iscrtavanja graf*



Slika 4.51. *Grafički prikaz simptoma pacijenta tijekom vremena*

4.2.4. Mogućnosti administrator korisnika

ADMIN korisnik se prilikom registracije kreira u trenutku kada mu korisničko ime glasi ADMIN. Kada se kreira, više nije moguće kreirati korisnika s tom rolom zbog uvedene validacije da može postoji samo jedinstveno korisničko ime što je prikazano prije navedenom slikom 4.9. ADMIN korisnik ima mogućnost pregleda baze cijele aplikacije kao i njeno vraćanje na sigurnosnu kopiju baze u slučaju prestanka rada sustava. Ima mogućnost uređivanja, brisanja i pregleda detalja postojećih korisnika i liječnika. Samo ADMIN ima mogućnost kreiranja liječnika. Slika 4.52 prikazuje prikaz svih korisnika registriranih u sustav.



Slika 4.52. *Prikaz registriranih korisnika ADMIN ulogom*

Uređivanje i pregled detalja korisnika objašnjen je kako radi u prethodnim poglavljima tako da taj dio neće biti objašnjen na ovome dijelu već će samo biti prikazan API koji to odrađuje. API programski kod za spremanje i dohvaćanje korisničkih podataka prikazan je slikom 4.53.

```
@GetMapping("/pages-all_users/edit/{id}")
public String editUser(@PathVariable Long id, Model model) {
    model.addAttribute( attributeName: "user", userService.getUserById(id));
    return "admin-edit-user";
}

@PostMapping("/pages-all_users/{id}")
public String updateUser(@PathVariable long id, @ModelAttribute("user") User user) {
    User existingUser = userService.getUserById(id);
    existingUser.setId(id);
    existingUser.setDetectedAllergy(user.getDetectedAllergy());
    existingUser.setProscribedMedicine(user.getProscribedMedicine());
    existingUser.setSymptoms(user.getSymptoms());
    existingUser.setEmail(user.getEmail());
    existingUser.setName(user.getName());
    existingUser.setUsername(user.getUsername());
    existingUser.setPassword(user.getPassword());
    existingUser.setMedicationUsage(user.getMedicationUsage());
    existingUser.setRecommendedMedicine(user.getRecommendedMedicine());
    existingUser.setRecoveryProcess(user.getRecoveryProcess());
    existingUser.setDoctor(user.getDoctor());
    existingUser.setAge(user.getAge());
    existingUser.setIsPregnant(user.getIsPregnant());
    existingUser.setWeight(user.getWeight());
    userService.updateUser(existingUser);
    return "redirect:/pages-all_users";
}
```

Slika 4.53. *Prikaz API programskog koda za dohvaćanje i spremanje korisničkih podataka*

AMIN ima mogućnost brisanja korisnika čime se brišu i sve međusobne relacije koje postoje. Iz odziva stranice se uzima koji ID je potrebno obrisati, pretražuje se korisnik sa tim identifikatorom te se vraća na početnu stranicu s listom korisnika. Programski kod načina brisanja pacijenta prikazan je slikom 4.54.

```
@PostMapping("/pages-all_users/delete")
public String deleteUser(@RequestParam("userId") Long userId) {
    userService.deleteUserByID(userId);
    return "redirect:/pages-all_users";
}
```

Slika 4.54. *Prikaz API za brisanje korisnika*

Za vraćanje baze u slučaju greške koristi se metoda executeScript koju API poziva. U metodi se izvršava shell skripta koja se nalazi na putanji path. Prvo se određuje operativni sustav (Windows ili Linux) kako bi se postavile odgovarajuće opcije za izvršavanje skripte. Ako je operativni sustav Windows, koristi se "cmd.exe" s opcijom "/c", dok se za Linux koristi bash s opcijom -c. Zatim se

koristi ProcessBuilder za izgradnju procesa koji će izvršiti skriptu. Skripta se izvršava pomoću ProcessBuilder.start(). Metoda waitFor() se koristi kako bi se sačekalo da se skripta završi prije nego što se nastavi s izvršavanjem programa. Ako izvršavanje skripte izazove iznimku, ispisuje se pogreška. Bez obzira na to što se dogodilo prilikom izvršavanja skripte, metoda uvijek preusmjerava korisnika na /pages-all_users. API za vraćanje stanja baze prikazan je slikom 4.55.

```
@PostMapping("/execute-script")
public String executeScript() {
    String path = System.getProperty("user.dir") +
            "\\src\\main\\java\\com\\example\\allergy_identifier\\security\\restore_snapshot.sh";
    String os = System.getProperty("os.name").toLowerCase();
    String command = os.contains("win") ? "cmd.exe" : "bash";
    String option = os.contains("win") ? "/c" : "-c";

    try {
        ProcessBuilder processBuilder = new ProcessBuilder(command, option, path);
        Process process = processBuilder.start();
        process.waitFor();
        return "redirect:/pages-all_users";
    } catch (IOException | InterruptedException e) {
        e.printStackTrace();
        return "redirect:/pages-all_users";
    }
}
```

Slika 4.55. *Prikaz API-ja za vraćanje baze podataka*

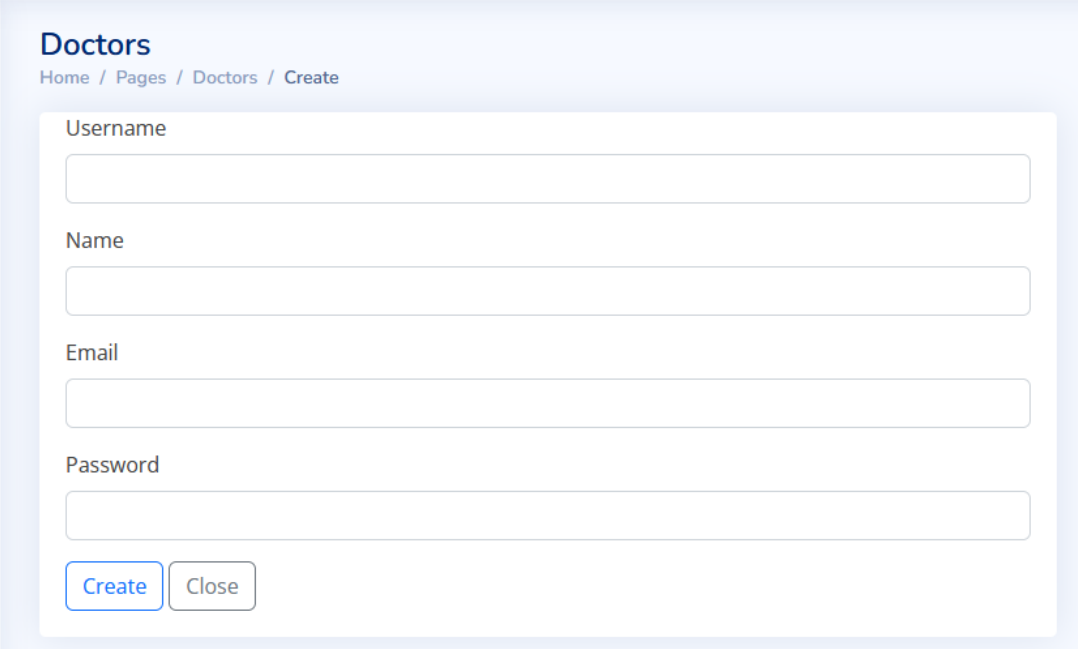
Skripta prikazana slikom 4.56 prvo briše bazu koja je trenutno postavljena, kreira novu public shemu te pomoću pg_restore i upisanih parametara spaja se na bazu i vraća trenutno stanje baze. Putanja gdje je spremljena kopija baze se automatski kreira ako ne postoji.

```
#!/bin/bash
DB_USER="postgres"
DB_NAME="allergy_identifier"
NEW_SCHEMA="public"
DROP_COMMAND="DROP SCHEMA public CASCADE;"

psql -U $DB_USER -d $DB_NAME -c "$DROP_COMMAND"
CREATE_QUERY="CREATE SCHEMA $NEW_SCHEMA;"
psql -U $DB_USER -d $DB_NAME -c "$CREATE_QUERY"
pg_restore -h localhost -p 5432 -U postgres -d allergy_identifier C:\\dbSnapshot\\database_snapshot.sql
```

Slika 4.56. *Prikaz skripte za vraćanje baze*

AMIN kreira liječnika. Prilikom kreiranja liječnika dodjeljuje mu se uloga DOCTOR. Ovisno o ulozi, liječnik vidi određene izbornike. API kreiranja liječnika i sami izgled web stranice kreiranja doktora prikazan je slikom 4.57.

```
@PostMapping("/pages-all_doctors/saveDoctor")
public String saveDoctor(@ModelAttribute("doctor") Doctor doctor) {
    doctorService.saveDoctor(doctor);
    return "redirect:/pages-all_doctors";
}
```



Slika 4.57. *Prikaz kreiranja liječnika i API za kreiranje liječnika*

4.2.5. Postojeće uloge unutar aplikacije

Unutar aplikacije postoje tri uloge, a to su admin, korisnik i liječnik. Dodavanje uloga odvija se na registraciji za admina i pacijenta, dok za liječnika prilikom kreiranja od strane admin korisnika. Slika 4.58 prikazuje odnose između tablica uloga korisnika i pacijenta.

```java
@ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
@JoinTable(name = "doctor_roles", joinColumns = {
        @JoinColumn(name = "DOCTOR_ID", referencedColumnName = "ID") }, inverseJoinColumns = {
                @JoinColumn(name = "ROLE_ID", referencedColumnName = "ID") })
private List<Role> roles = new ArrayList<>();
```

```java
    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinTable(name = "doctor_roles", joinColumns = {
            @JoinColumn(name = "DOCTOR_ID", referencedColumnName = "ID") }, inverseJoinColumns = {
                    @JoinColumn(name = "ROLE_ID", referencedColumnName = "ID") })
    private List<Role> roles = new ArrayList<>();
```

```java
@Table(name = "roles")
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String name;

    @ManyToMany(mappedBy = "roles")
    private List<User> users;

    @ManyToMany(mappedBy = "roles")
    private List<Doctor> doctors;
}
```

Slika 4.58. *Prikaz programskog koda odnosa između tablica*

Definirana je klasa Role, User i Doctor. Entitet ima tri atributa: id, ime i liste pacijenata i liječnika koje su povezane s drugim entitetima Korisnik i Liječnik preko veze Više naprema više. Ovo sugerira da svaka uloga može biti dodijeljena više korisnika i/ili liječnika, a svaki korisnik/liječnik može imati više uloga. Atribut id označava primarni ključ entiteta koji se automatski generira pomoću strategije GenerationType.IDENTITY. Atribut ime predstavlja naziv uloge i postavljen je kao obavezan te mora biti jedinstven u bazi podataka. Veza više naprema više s korisnik entitetom definirana je anotacijom @ManyToMany(mappedBy = roles). To znači da je veza već definirana u entitetu Korisnik i da se veza mapira na atribut roles tog entiteta. Veza više naprema više s liječnik entitetom također je definirana anotacijom @ManyToMany(mappedBy = roles), što znači da je veza već definirana u entitetu liječnik i mapira se na atribut roles tog entiteta. Važno je napomenuti da oba atributa roles imaju postavljenu opciju fetch = FetchType.EAGER, što znači da će se uloge uvijek dohvaćati odmah kada se dohvaća entitet korisnika ili liječnika, bez obzira na to jesu li uloge potrebne u tom trenutku. Osim toga, opcije cascade = { CascadeType.MERGE, CascadeType.PERSIST, CascadeType.REFRESH } za entitet korisnik i cascade = CascadeType.ALL za entitet liječnik

omogućuju da se operacije spremanja (poput dodavanja novih uloga) automatski prošire na povezane entitete. Slika 4.59 prikazuje način spremanja uloga unutar baze podataka.

```
@Override
public void saveUser(User user) {
    User newUser = new User();
    String checkboxResult = dataLoader.getCheckboxValue(request);

    newUser.setName(user.getName());
    newUser.setUsername(user.getUsername());
    newUser.setEmail(user.getEmail());
    newUser.setPassword(passwordEncoder.encode(user.getPassword()));
    newUser.setIsPregnant(checkboxResult);
    newUser.setAge(user.getAge());
    newUser.setWeight(user.getWeight());
    Role role = new Role();

    if (newUser.getUsername().equals("admin")) {
        Role user_role = roleRepository.findByName("ROLE_ADMIN");
        if (user_role == null) {
            role.setName("ROLE_ADMIN");
            roleRepository.save(role);
        }
        Role readRole = roleRepository.findByName("ROLE_ADMIN");
        newUser.setRoles(Collections.singletonList(readRole));
        userRepository.save(newUser);
    } else {
        Role existingRole = roleRepository.findByName("ROLE_USER");
        if (existingRole == null) {
            role.setName("ROLE_USER");
            roleRepository.save(role);
        }
        Role readRole = roleRepository.findByName("ROLE_USER");
        newUser.setRoles(Collections.singletonList(readRole));
        userRepository.save(newUser);
    }
}
```

Slika 4.59. *Prikaz programskog koda spremanja uloga ovisno o korisniku*

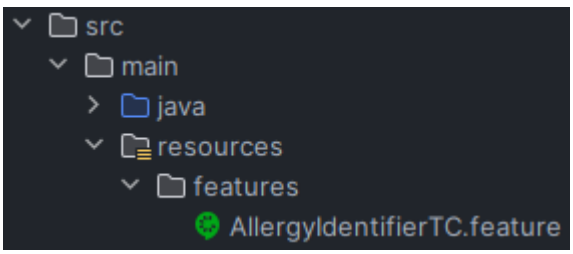
Metoda saveUser sprema novog korisnika u bazu podataka. Ako je korisničko ime admin, dodjeljuje mu ulogu "ROLE_ADMIN". Ako nije admin, dodjeljuje mu ulogu "ROLE_USER".

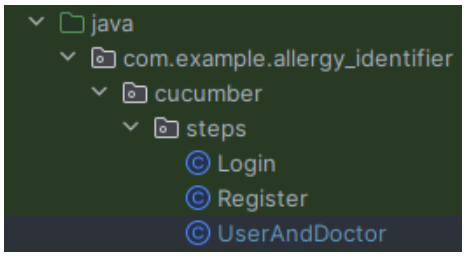
4.2.6. API testiranje

Thymeleaf API testiranje u Cucumberu predstavlja moćan pristup osiguravanju ispravnosti dinamičkog HTML sadržaja generiranog pomoću Thymeleaf-a unutar web aplikacija. Thymeleaf, kao popularna Java biblioteka za generiranje HTML-a, omogućava integraciju podataka i logike u HTML predloške. Cucumber, s druge strane, omogućava pisanje testova u prirodnom jeziku, koji se automatski prevode u izvršive skripte. Ovaj pristup pruža mogućnost testiranja različitih scenarija generiranja sadržaja unutar Thymeleaf predložaka, čime osigurava ispravnost prikaza i povezivanje s

backend podacima. Cucumber koristi definiranje koraka koji opisuju postupke i način testiranja, uključujući otvaranje stranica, izvlačenje generiranog HTML sadržaja i radnje nad njim.

Najprije je bilo potrebno definirati glavnu Java klasu koja će biti zadužena za inicijalizaciju programskog koda koji bi se pokretao prilikom izvođenja. Pokretanjem klase prikazane slikom 4.60. pokreću se svi napisani testovi na lokaciji features iznad napisane klase. Nakon izvođenja zbog prve postavke se generira html izvještaj o uspješnosti izvedenih testova u izbornom direktoriju projekta.





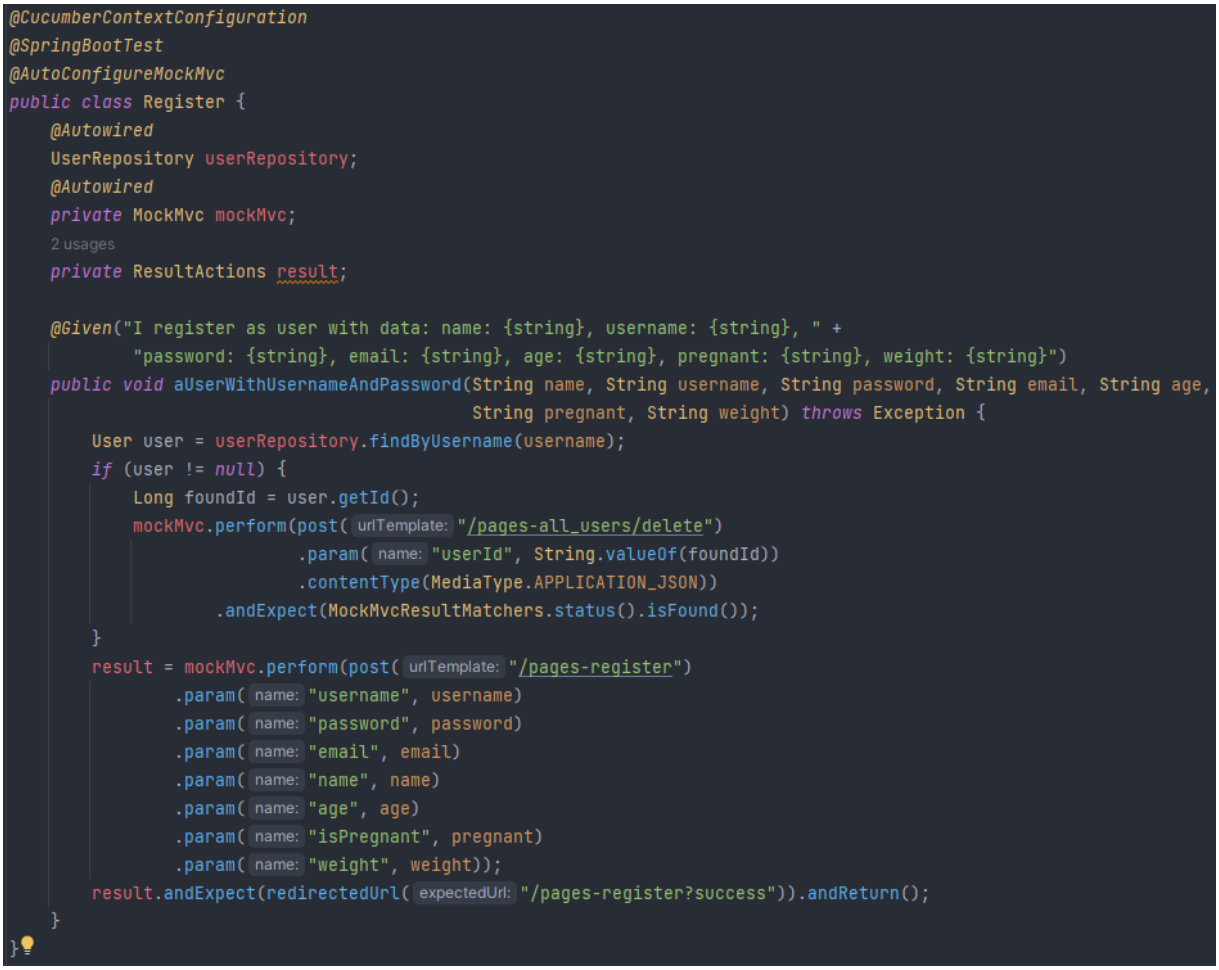

```
package com.example.allergy_identifier.cucumber;

import ...

DESKTOP-790S4V1\Ivan
@RunWith(Cucumber.class)
@CucumberOptions(plugin = {"html:cucumber-report.html"},
        features = "src/main/resources/features",
        glue = "com.example.allergy_identifier.cucumber")
public class CucumberTestRunner {
}
```

Slika 4.60. *Prikaz strukture i klase za inicijalizaciju testova*

Cucumber omogućuje jednostavno pisanje koraka izvođenja, gdje svakome tko i ne poznaje programske jezike, čitajući definirane korake zna o čemu je riječ. Najprije je bilo potrebno definirati scenarijo, tj. same korake testa. Glavne riječi koje cucumber nudi za definiranje koraka radi lakšeg shvaćanja su *given, when, and i then.* Za primjer će biti objašnjen jedan korak i način njegove implementacije. Na taj isti način su napravljeni i svi ostali. Slika 4.61 prikazuje scenarijo autorizacije pacijenta, dok slika 4.62 prikazuje opisani prvi korak scenarija.

```
Scenario: Successful login
  Given I register as user with data: name: "admin", username: "admin", password:
  "admin", email: "admin@gmail.com", age: "23", pregnant: "false", weight: "23"
  And I check if User with username: "admin" exists
  When the user logs in with username: "admin" and password "admin"
  And I delete user "admin"
```

Slika 4.61. *Prikaz scenarija autorizacije korisnika u sustav*



```
@CucumberContextConfiguration
@SpringBootTest
@AutoConfigureMockMvc
public class Register {
    @Autowired
    UserRepository userRepository;
    @Autowired
    private MockMvc mockMvc;
    2 usages
    private ResultActions result;

    @Given("I register as user with data: name: {string}, username: {string}, " +
            "password: {string}, email: {string}, age: {string}, pregnant: {string}, weight: {string}")
    public void aUserWithUsernameAndPassword(String name, String username, String password, String email, String age,
                                             String pregnant, String weight) throws Exception {
        User user = userRepository.findByUsername(username);
        if (user != null) {
            Long foundId = user.getId();
            mockMvc.perform(post( urlTemplate: "/pages-all_users/delete")
                            .param( name: "userId", String.valueOf(foundId))
                            .contentType(MediaType.APPLICATION_JSON))
                    .andExpect(MockMvcResultMatchers.status().isFound());
        }
        result = mockMvc.perform(post( urlTemplate: "/pages-register")
                .param( name: "username", username)
                .param( name: "password", password)
                .param( name: "email", email)
                .param( name: "name", name)
                .param( name: "age", age)
                .param( name: "isPregnant", pregnant)
                .param( name: "weight", weight));
        result.andExpect(redirectedUrl( expectedUrl: "/pages-register?success")).andReturn();
    }
}
```

Slika 4.62. *Prikaz koraka registracije pacijenta*

Kombinacija prikazanih anotacija sa slike 4.62 stvara optimalno okruženje za testiranje unutar Cucumber scenarija. @CucumberContextConfiguration osigurava da Cucumber i Spring Boot komuniciraju na odgovarajući način, @SpringBootTest postavlja Spring Boot kontekst za testiranje, dok @AutoConfigureMockMvc priprema MockMvc za simulaciju HTTP zahtjeva. Ovako postavljeno okruženje omogućava testiranje integrirane funkcionalnosti unutar stvarnog Spring Boot okruženja, uz podršku za Cucumber scenarije. Ova klasa predstavlja jedan od koraka u Cucumber scenariju testiranja registracije korisnika. Prije nego što izvrši simuliranu registraciju, provjerava se postoji li korisnik s istim korisničkim imenom u bazi podataka. Ako postoji, taj korisnik se briše kako bi se osigurala čistoća testiranja. Nakon toga, koristeći Spring Boot MockMvc, izvršava se HTTP zahtjev na putanju /pages-register s navedenim parametrima, što simulira registraciju korisnika. Nakon što je zahtjev izvršen, provjerava se očekivani rezultat putem rezultat objekta. Konkretno, očekuje se preusmjeravanje na /pages-register?success, što ukazuje na uspješno izvršenu registraciju.

Ovaj testni slučaj osigurava da registracijski proces pravilno funkcionira, uklanjajući eventualno postojeće korisnike s istim korisničkim imenom i provjeravajući ispravno preusmjeravanje nakon uspješne registracije. Time se osigurava kvaliteta i stabilnost funkcionalnosti registracije u okviru razvojne aplikacije. Nakon toga, prema slici 4.61, prelazi se na idući korak, a to je autorizacija u sustav koja je prikazana slikom 4.63.

```
@And("I check if User with username: {string} exists")
public void checkIfExists(String username) throws Exception {
    result = mockMvc.perform(get( urlTemplate: "/checkUserParams")
            .param( name: "username", username));
    int statusCode = result.andReturn().getResponse().getStatus();
    if (statusCode != 200) {
        fail("User with username: " + username + " does not exist.");
    }
}

@When("the user logs in with username: {string} and password {string}")
public void theUserLogsIn(String username, String password) throws Exception {
    result = mockMvc.perform(post( urlTemplate: "/pages-login")
            .param( name: "username", username)
            .param( name: "password", password));
    result.andExpect(MockMvcResultMatchers.redirectedUrl( expectedUrl: "/index")).andReturn();
}
```

Slika 4.63. *Prikaz koraka provjere postojanja i autorizacije pacijenta*

@And označava da se radi o dodatnom koraku unutar Cucumber scenarija. Metoda checkIfExists simulira korisničku radnju provjere postojanja korisnika s konkretnim korisničkim imenom. U prvoj liniji programskog koda, metoda perform klase MockMvc izvršava HTTP GET zahtjev prema API-u /checkUserParams, s dodanim parametrom username kako bi se provjerilo postojanje korisnika s navedenim korisničkim imenom. Dalje, putem getResponse().getStatus() dohvaća se HTTP statusni kod odgovora na zahtjev. Naposljetku, provjerava se statusni kod. Ako nije 200 (OK), test završava neuspjehom, obavještavajući da korisnik s navedenim korisničkim imenom ne postoji u sustavu. Zatim se pozivom idućeg koraka šalje poziv na API za autorizaciju korisnika gdje se šalju korisničko ime i lozinka, te se ako je autorizacija uspješna očekuje preusmjeravanje na /indeks web stranicu. Zatim se korisnik u zadnjemu koraku briše kako bi idući test mogao raditi efikasnije i točnije. Svaki test mora raditi pri samome pokretanju i pokretanju u cjelini, tj. regresiji. Na opisane načine su napisani i ostali koraci koji šalju podatke na druge API-e. Ovim putem je osigurana točnost aplikacije jer pri pokretanju svih testova, lako se može zaključiti koji dijelovi ne rade ako neki testovi padnu.

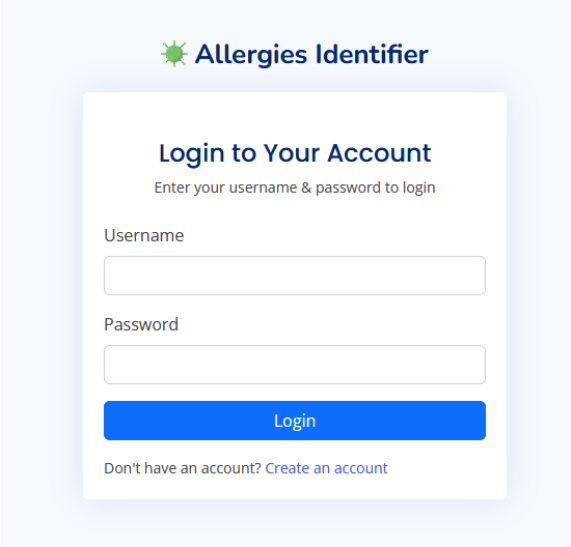
Ovaj način testiranja je brži i efikasniji jer se direktno pogađa API koji šalje podatke dalje te se koraci koje ručno radimo neko vrijeme, pokretanjem testova mogu odraditi u nekoliko sekundi [23].

5. PRIKAZ NAČINA RADA, ISPITIVANJE I TESTIRANJE WEB APLIKACIJE

Nakon što je web sustav kreiran i testiran, navedene su korisničke upute o načinu korištenja i funkcionalnostima. U ovom poglavlju korisnici će saznati kako koristiti web sustav, kako mu pristupiti i koristit se njime. Također, bit će prikazani primjeri slučajeva koji pokazuju točnost rada web sustava te analiza rezultata koje korisnici mogu očekivati prilikom korištenja sustava. Sve ove informacije će pomoći korisnicima da maksimalno iskoriste prednosti i mogućnosti koje pruža naš web sustav.

5.1. Način rada web aplikacije

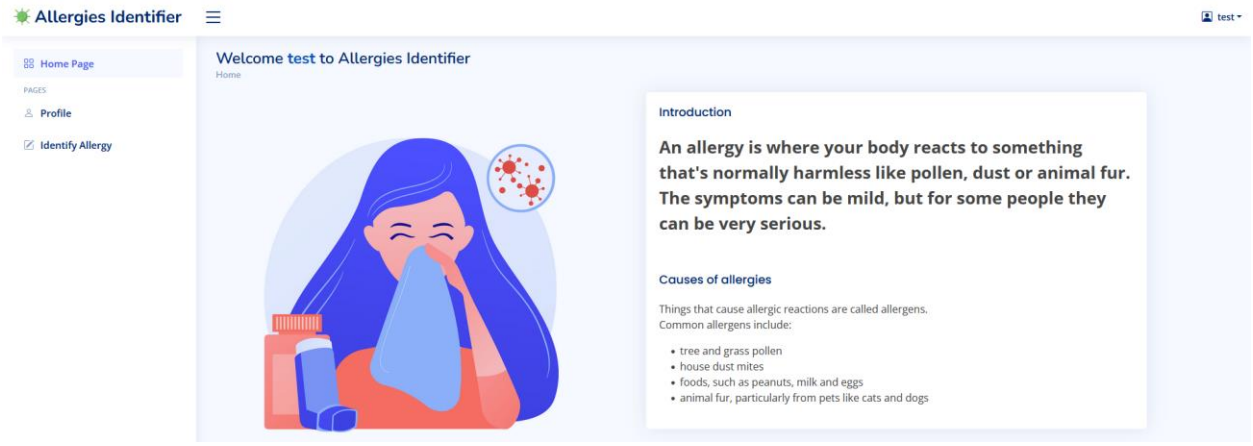
U prijašnjim poglavljima je detaljno opisano što je potrebno da bi se uspješno odradio zahtjev, a unutar ovoga dijela će biti kratak opis korištenja same aplikacije za nove korisnike. Dolaskom na naslovnu stranicu otvara se mjesto gdje se postojeći korisnici mogu prijaviti ili novi registrirati. Pritiskom na tipku prijave, korisnik je prebačen na početnu stranicu aplikacije prikazanu slikom 5.1.



Slika 5.1. *Prikaz naslovne stranice*

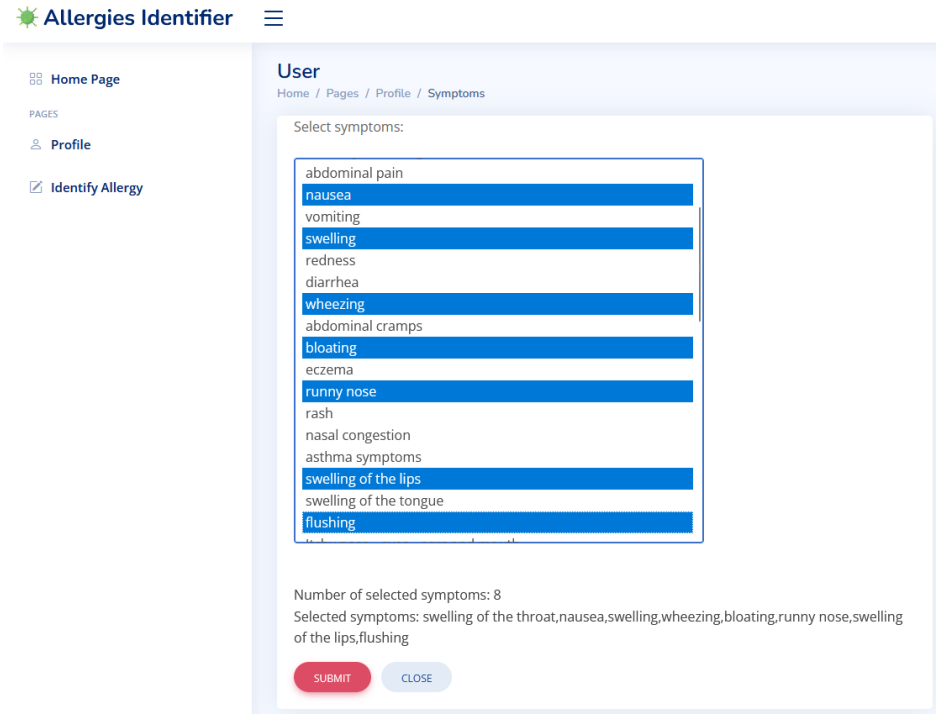
5.1.1. Pacijent kao korisnik

Kako bi se na strani pacijenta mogla odraditi preporuka alergije, pacijent najprije mora odabrati simptome koje trenutno ima. Slika 5.2 prikazuje početnu stranicu nakon autorizacije.



Slika 5.2. *Prikaz početne stranice*

Klikom na svoj profil, pojavljuje se gumb dodaj simptome gdje pacijent ima mogućnost odabira ponuđenih simptoma. Minimalan broj simptoma je tri, dok je maksimalan broj osam. Način odabira simptoma prikazan je slikom 5.3.



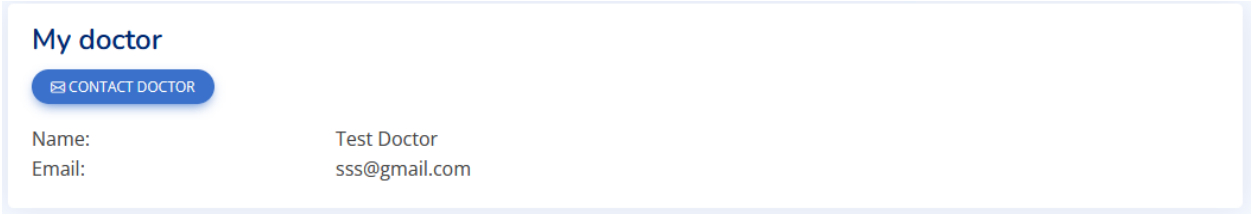
Slika 5.3. *Prikaz načina odabira simptoma*

Nakon odabira simptoma, korisnik ima mogućnost prelaska na izbornik dijagnosticiranja alergije koji je prikazan slikom 5.4. Ako prethodno nije odabrao simptome, neće imati mogućnost odabira

dijagnoze alergije. Prilikom detektiranih simptoma korisnika, klikom na gumb dijagnosticiraj alergiju, pokreće se postupak dijagnoze. Nakon toga pacijent dobiva odgovor o pronađenoj alergiji.

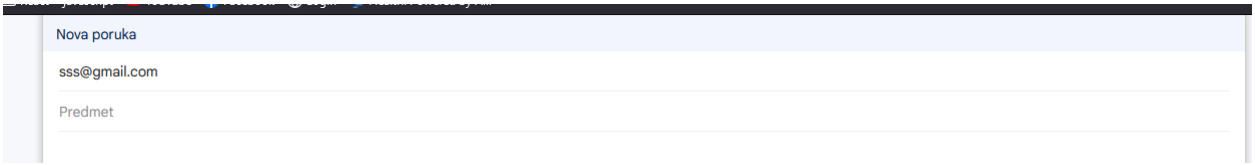
Slika 5.4. *Prikaz stranice za dijagnozu alergije*

Na svome profilu pacijent također ima mogućnost kontaktiranja liječnika ako mu informacija ispod gumba govori da ima propisanoga liječnika što je prikazano slikom 5.5.



Slika 5.5. *Prikaz dijaloga za kontakt liječnika*

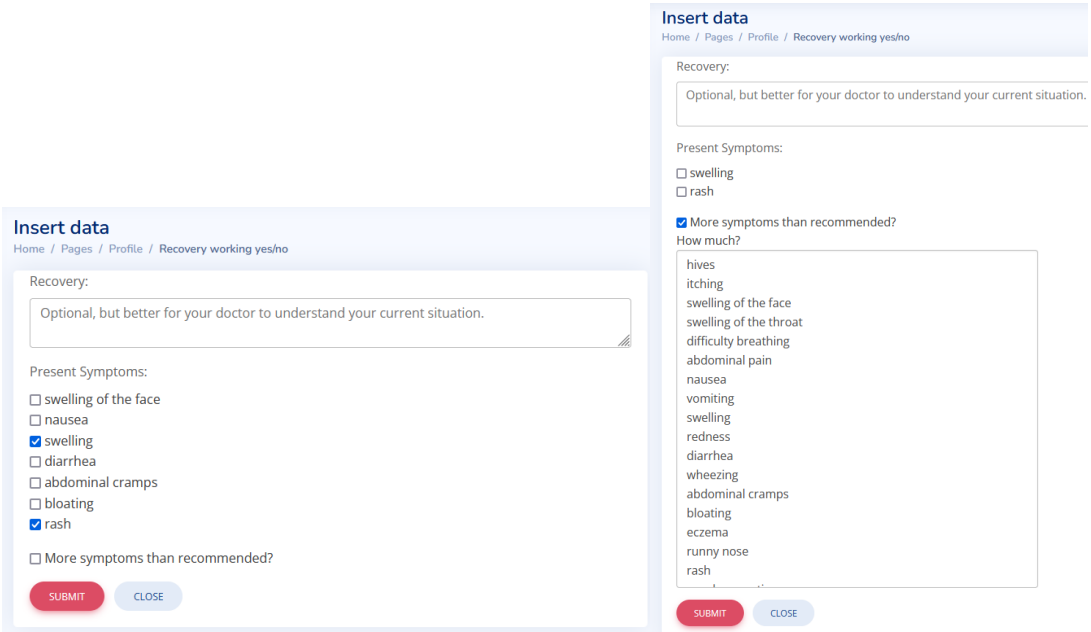
Klikom na gumb, otvara se elektronička pošta s već prethodno definiranom mail adresom liječnika. Prikaz otvorene elektorničke pošte s automatski ispunjenom adresom prikazano je slikom 5.6.



Slika 5.6. *Prikaz prozora elektroničke pošte*

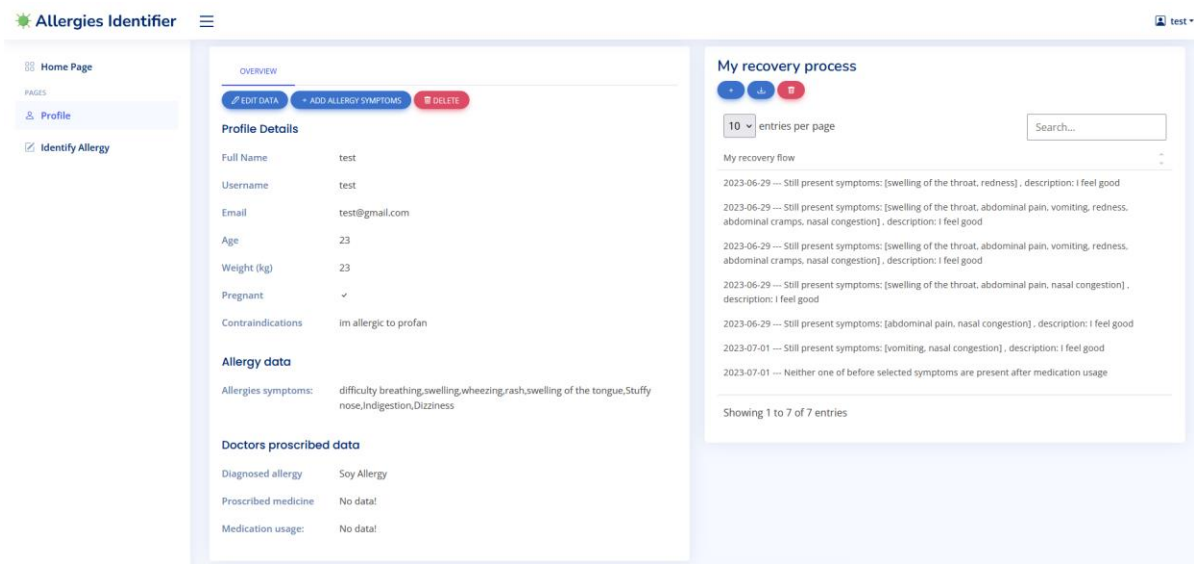
Također, pacijent u bilo kojemu trenutku može unijeti svoje trenutno stanje u dijelu procesa oporavka na njegovome profilu. Klikom na + gumb, otvara se prozor s poljima koje je poželjno ispuniti. Što više podataka je ispunjeno, to liječnik ima jasniji pregled u pacijentovo stanje. Prilikom prvog unosa

osjećanja pacijenta, simptomi koji se preporučuju su oni koje je on odabrao, ali nakon što se odaberu novi prisutni simptomi, u idućoj zabilješci stanja pacijenta, zadnji označeni simptomi se pojavljuju kao mogućnost odabira uz mogućnost odabira novih ne ponuđenih simptoma. Slika 5.7 prikazuje način unošenja tijeka oporavka pacijenta.



Slika 5.7. *Prikaz unosa podataka tijeka oporavka pacijenta*

Potvrdom unesenih podataka, podaci se pojavljuju na profilu pacijenta u tablici koju može pretraživati po bilo kojem tekstu što je prikazano slikom 5.8. Također, pacijent može preuzeti tijek oporavka u slučaju potrebe slanja liječniku ili nekoj osobi te ima mogućnost potpunog brisanja cijelog tijeka oporavka u slučaju želje za potpuno novim praćenjem tijeka oporavka. Unutar prozora podataka pacijenta, pacijent ima mogućnost brisanja svojih simptoma i svega što je liječnik preporučio. Ovu stavku je jedino dobro koristiti kada se želi krenuti s novim početkom tretiranja liječenja.



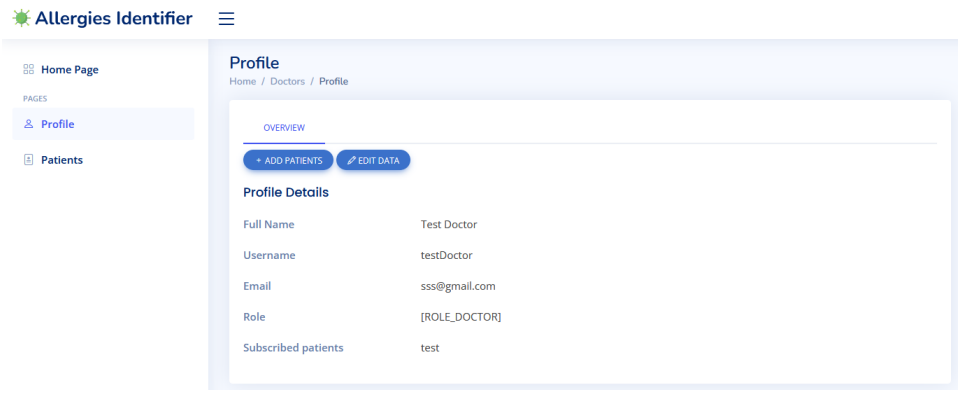
Slika 5.8. *Prikaz profila pacijenta*

5.1.2. Liječnik kao korisnik

Prijavom liječnika u sustav, dolazi na početnu stranicu aplikacije koja je prikazana slikom 5.9. Liječnik, za razliku od pacijenta, vidi druge izbornike pri ulasku u aplikaciju

Slika 5.9. *Prikaz početne stranice liječnika*

Odlaskom na profil, liječnik ima pregled osobnih podataka te mogućnost njihovog uređivanja što je vidljivo slikom 5.10. Klikom na gumb za uređivanje podataka, otvara se stranica s podacima i mogućnostima izmjene. Potvrdom, ti podaci se spremaju te se liječnik vraća na svoj profil. Također ima opciju odabira dodavanja pacijenata koje želi liječiti.



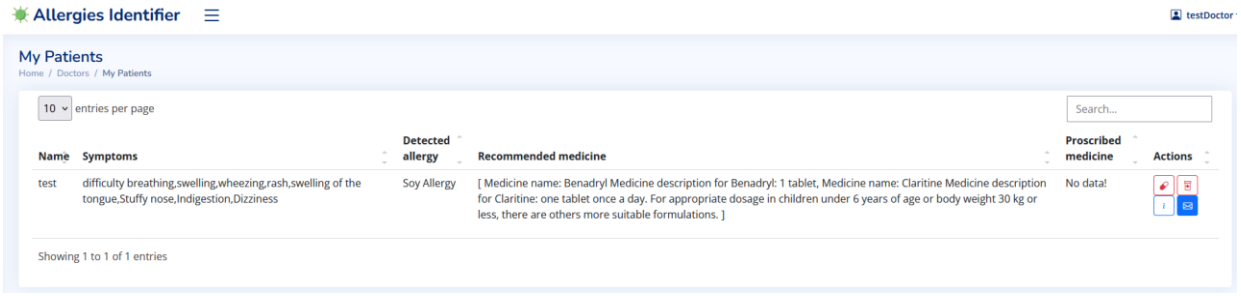
Slika 5.10. *Prikaz profila liječnika*

Klikom na gumb za dodavanje pacijenata, otvara se stranica koja mu to i omogućava. Odabirom i potvrdom pacijenti se dodaju i liječnik ima mogućnost pregleda tih pacijenata u izborniku. Slika 5.11 prikazuje način odabira pacijenata za praćenje.

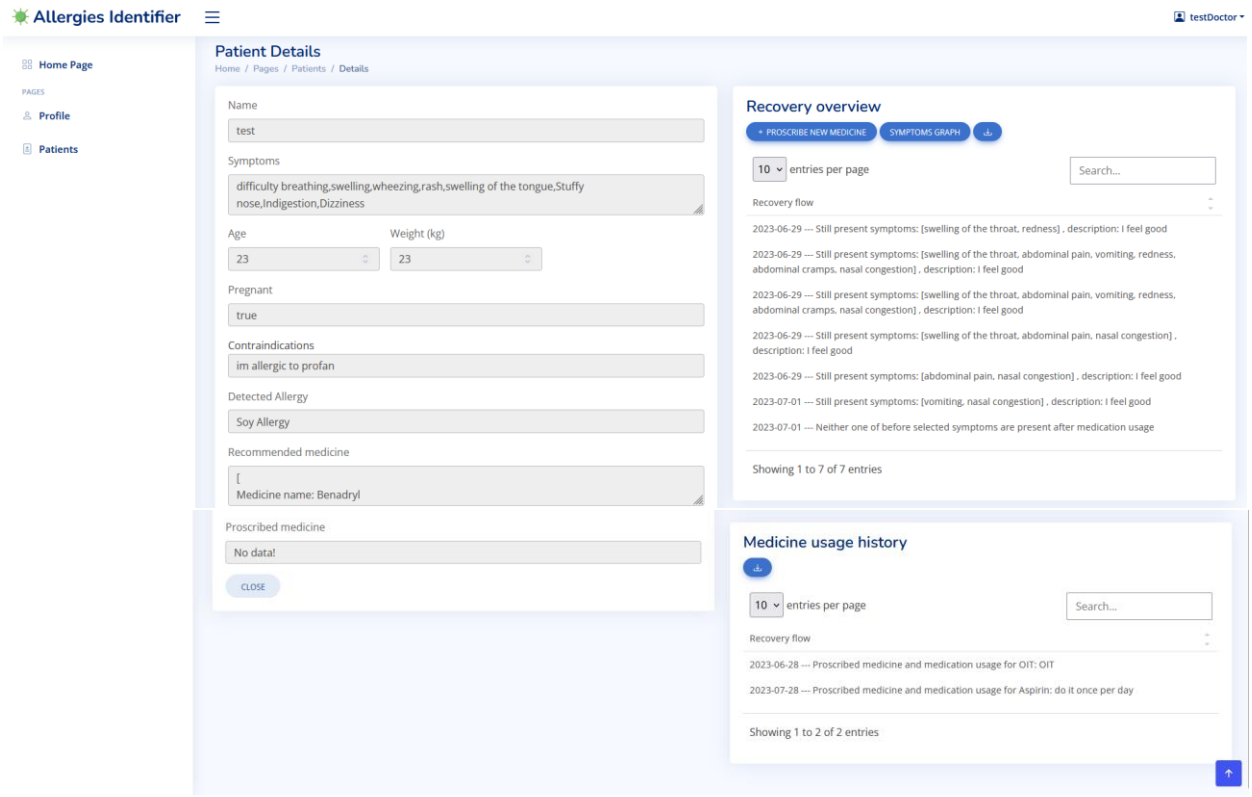
Slika 5.11. *Prikaz odabira pacijenata*

Klikom liječnika u izbornik pacijenata, otvara se stranica s njegovim odabranim pacijentima. Za svakog pacijenta liječnik ima mogućnost pregleda svih podataka koji su prikazani tablicom, te u koloni akcija, ima mogućnost preporuke lijeka, koja će na temelju prethodno navedenih parametara odlučiti koji lijek bi bio pogodan za pacijenta, zatim potvrdu preporučenog lijeka koja je prethodno prikazana slikom 4.47, gdje liječnik pregledava sav generirani sadržaj koji je aplikacija ponudila te na temelju svoga znanja donosi konačnu odluku te propisuje lijek. Klikom na potvrdu, unos korištenog lijeka se sprema te se generira sustav koji omogućava pregled korištenja lijekova tijekom liječenja pacijenta. Klikom na detalje o pacijentu što je prikazano slikom 5.13, liječnik pregledava sad sadržaj pacijenta gdje također grafički ima uvid u tijek samog liječenja pacijenta što je prikazano prethodno navedenom

slikom 4.51. Zadnja mogućnost koju liječnik ima je stupanje u kontakt s pacijentom putem elektroničke pošte.



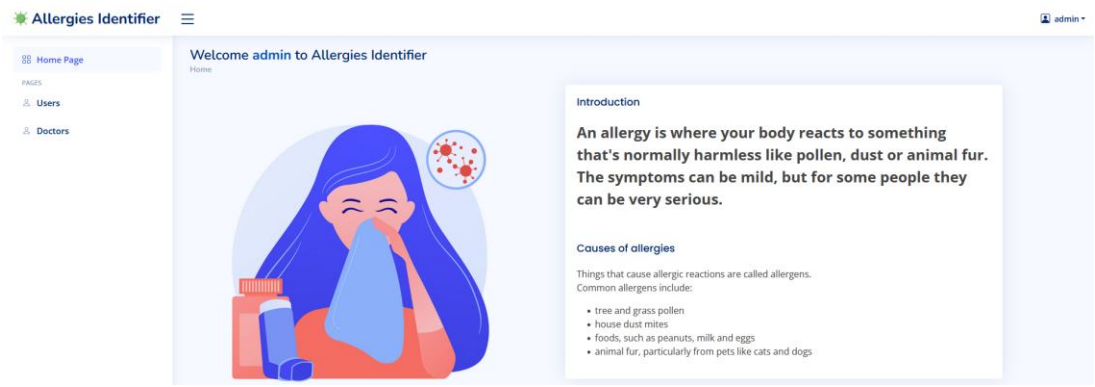
Slika 5.12. *Prikaz pacijenata liječnika*



Slika 5.13. *Prikaz detalja pacijenta*

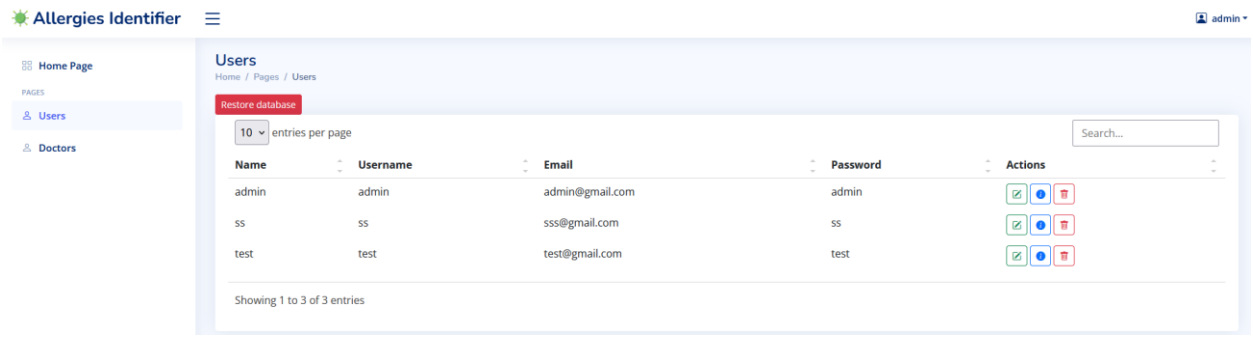
5.1.3. Admin kao korisnik

Ulaskom admin korisnika u aplikaciju, dolazi na početnu stranicu gdje ima pregled odabira korisnika i liječnika. Odabirom korisnika, otvara se pozor sa svim trenutnim korisnicima koji su registrirani u aplikaciji, te to isto pravilo vrijedi i za liječnike. Svi liječnici koji su definirani od strane admina se nalaze u izborniku liječnika. Slika 5.14 prikazuje početnu stranicu ADMIN korisnika.

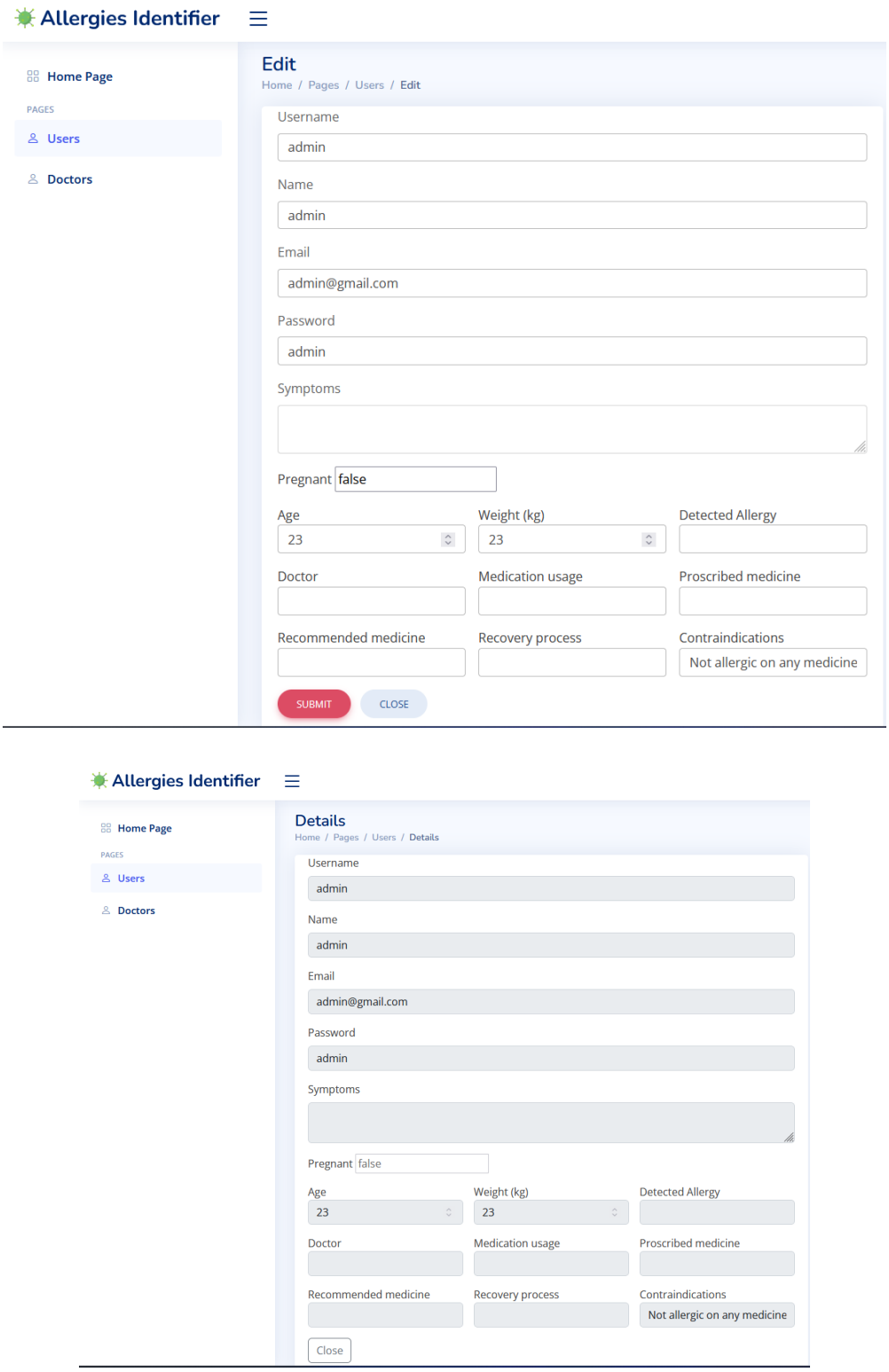


Slika 5.14. *Početna stranica ADMIN korisnika*

Klikom na izbornik korisnika, otvara se prozor svih korisnika u aplikaciji što je prikazano slikom 5.15. ADMIN ima mogućnost uređivanja gdje nakon uređivanja podataka klikom na potvrdu, podaci se spremaju te je korisnik ažuriran. Također ima mogućnost pogleda detalja korisnika te njegovo brisanje. Uređivanje i pregled detalja korisnika prikazano je slikom 5.16. Također samo admin ima mogućnost oporavka baze u slučaju greške unutar baze podataka. Klikom na gumb vraćanja baze , pokreće se skripta koja vraća stanje baze na trenutak kada je okinuta sigurnosna kopija baze.

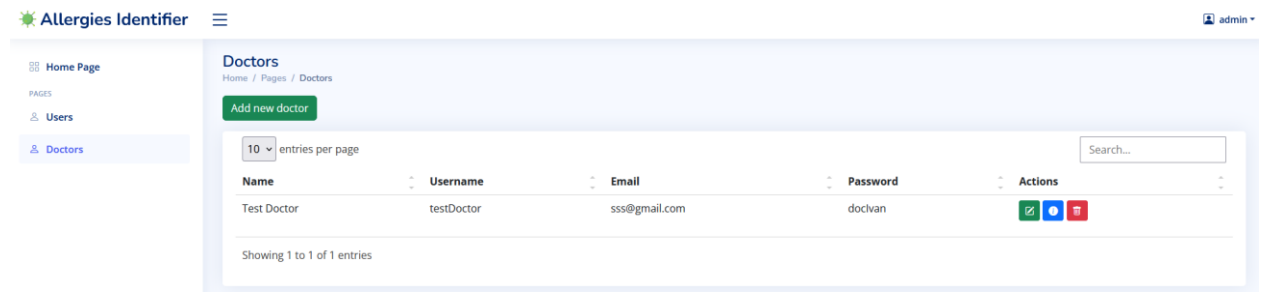


Slika 5.15. *Prikaz svih registriranih korisnika*



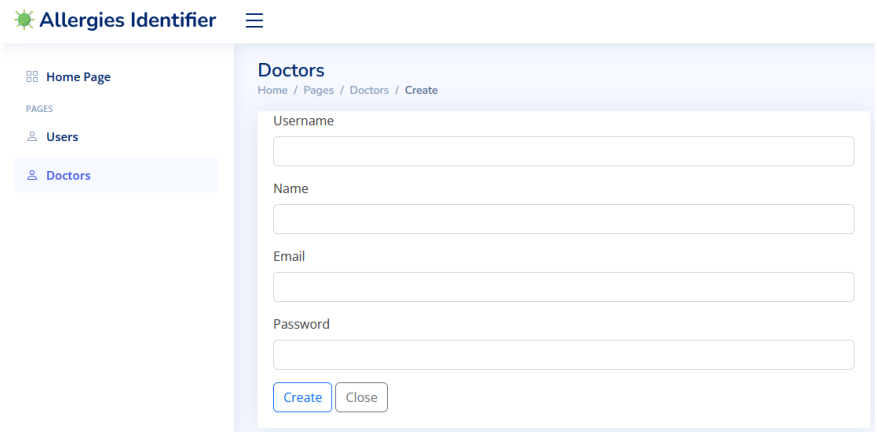
Slika 5.16. *Prikaz prozora uređivanja i pregleda detalja korisnika*

Samo admin korisnik ima mogućnost kreiranja liječnika. Klikom na izbornik liječnik, otvara se prozor koji omogućava pregled postojećih liječnika i mogućnost za dodavanjem novoga što je prikazano slikom 5.17.



Slika 5.17. *Prikaz tablice postojećih liječnika*

Akcije koje admin ima na liječniku su iste kao i za pacijenta. Klikom na gumb dodaj novog liječnika, otvara se stranica koja zahtjeva podatke o liječniku. Nakon uspješnog unošenja svakog od podataka, klikom na gumb kreiraj, liječnik se dodaje u bazu podataka i dodijeljena mu je uloga LIJEČNIK. Nakon toga liječnik ima mogućnost prijavljivanja u sustav i obavljanja svojih dužnosti. Slika 5.18 prikazuje način kreiranja liječnika.



Slika 5.18. *Prikaz načina dodavanja novoga liječnika*

5.2. Ispitivanje web aplikacije

Ispitivanje web aplikacije ključan je dio u razvoju. Ispitivanjem se zaključuje radi li sustav prema propisanim standardima. Ovakav način je također potreban da bi se provjerila dobra dijagnoza alergije te sama preporuka lijeka prema pacijentovim stanjima. Ovakav sustav bi se dao proširiti. Da bi se sustav proširio, potrebno je samo dodati nove alergije sa simptomima ili lijekove u bazu podataka te

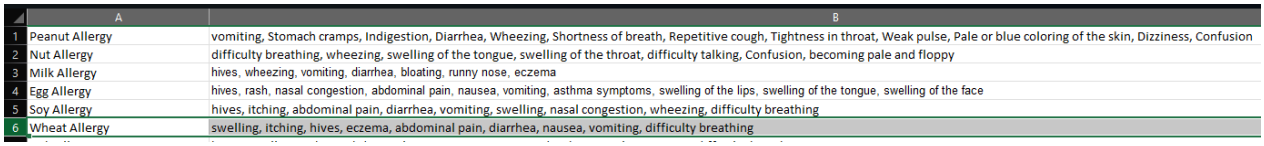
takve izmjene neće utjecati na glavne funkcionalnosti te veće potrebe za mijenjanjem sustava neće biti potrebne. U nastavku će biti prikaza tri slučaja testiranja aplikacije u kojemu će se testirati minimalan, maksimalan i srednji broj simptoma te očekivanje za točnom dijagnozom na temelju pacijentovi stanja koja će se također kroz testove promijeniti zbog testiranja točne dijagnoze lijekova.

5.2.1. Primjer korištenja 1

U prvi primjer testiranja spadaju 3 simptoma koje pacijent odabire, jer su tri simptoma minimalan broj koji aplikacija dopušta kao odabir, te na temelju tri simptoma želi se prikazati točna dijagnoza alergije. Pacijent će imati 23 godine, 55 kg težine i neće biti trudan te od kontraindikacije neće imati ništa navedeno. S tim atributima će se ići u preporuku lijeka pacijenta. Prikaz podataka prvog primjera prikazan je slikom 5.19.

Slika 5.19. *Prikaz podataka pacijenta*

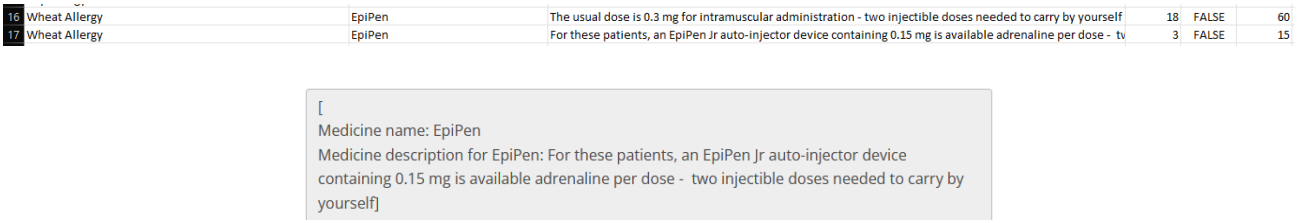
Na temelju slike 5.19 dijagnosticirana je alergija na pšenicu. Prilikom pokretanja dijagnoze u bazi alergija na pšenicu ima simptome: *swelling, itching, hives, eczema, abdominal pain, diarrhea, nausea, vomiting, difficulty breathing*. Pacijent je odabrao tri simptoma koja su: nausea,diarrhea,eczema. Ako se u listi simptoma prebroji koliko ima simptoma te alergije koji odgovaraju pacijentovom unosu, vidi se da je program dobro izračunao postotak. Tri simptoma se dijele sa maksimalnim brojem simptoma te alergije, što je devet, ispada 0.33*100 = 33.33%. To se odradi za svaku alergiju te se dobiva lista u programu ([0.0, 0.0, 28.57, 10.0, 11.11, **33.33**, 22.22, 30.0, 12.5, 12.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]). Pronalazi se najveći indeks u listi te pomoću njega unutar baze Excela gdje su alergije definirane se potrebno pozicionirati na pronađeni indeks što je prema slici 5.20 alergija na pšenicu, čime se zaključuje da je dijagnoza alergija točna za 3 odabrana simptoma.



	A	B
1	Peanut Allergy	vomiting, Stomach cramps, Indigestion, Diarrhea, Wheezing, Shortness of breath, Repetitive cough, Tightness in throat, Weak pulse, Pale or blue coloring of the skin, Dizziness, Confusion
2	Nut Allergy	difficulty breathing, wheezing, swelling of the tongue, swelling of the throat, difficulty talking, Confusion, becoming pale and floppy
3	Milk Allergy	hives, wheezing, vomiting, diarrhea, bloating, runny nose, eczema
4	Egg Allergy	hives, rash, nasal congestion, abdominal pain, nausea, vomiting, asthma symptoms, swelling of the lips, swelling of the tongue, swelling of the face
5	Soy Allergy	hives, itching, abdominal pain, diarrhea, vomiting, swelling, nasal congestion, wheezing, difficulty breathing
6	Wheat Allergy	swelling, itching, hives, eczema, abdominal pain, diarrhea, nausea, vomiting, difficulty breathing

Slika 5.20. *Prikaz dijagnosticirane alergije*

Nakon uspješne dijagnoze alergije, dolazi se do preporuke lijeka za pacijenta na temelju njegovih unesenih podataka. Gore opisanim atributima pacijenta, pokreće se preporuka lijeka. Unutar baze lijeka prikazana je dobna granica 18 godina, što znači da lijek nije prikladan za trudnice i da lijek smiju koristiti osobe 0.3mg dozu stariji od 60 godina i 0.15mg dozu stariji od 15 godina. Slikom 5.21 prikazan je dobro preporučen lijek na temelju pacijentovih podataka. 0.3mg nije preporučeno iz razloga što pacijent ima 55 godina, a granica je 60, čime je eliminirana doza tog lijeka za pacijenta. S ovim primjerom prikazan je ispravan rad sustava dijagnoze alergije i preporuke lijeka.



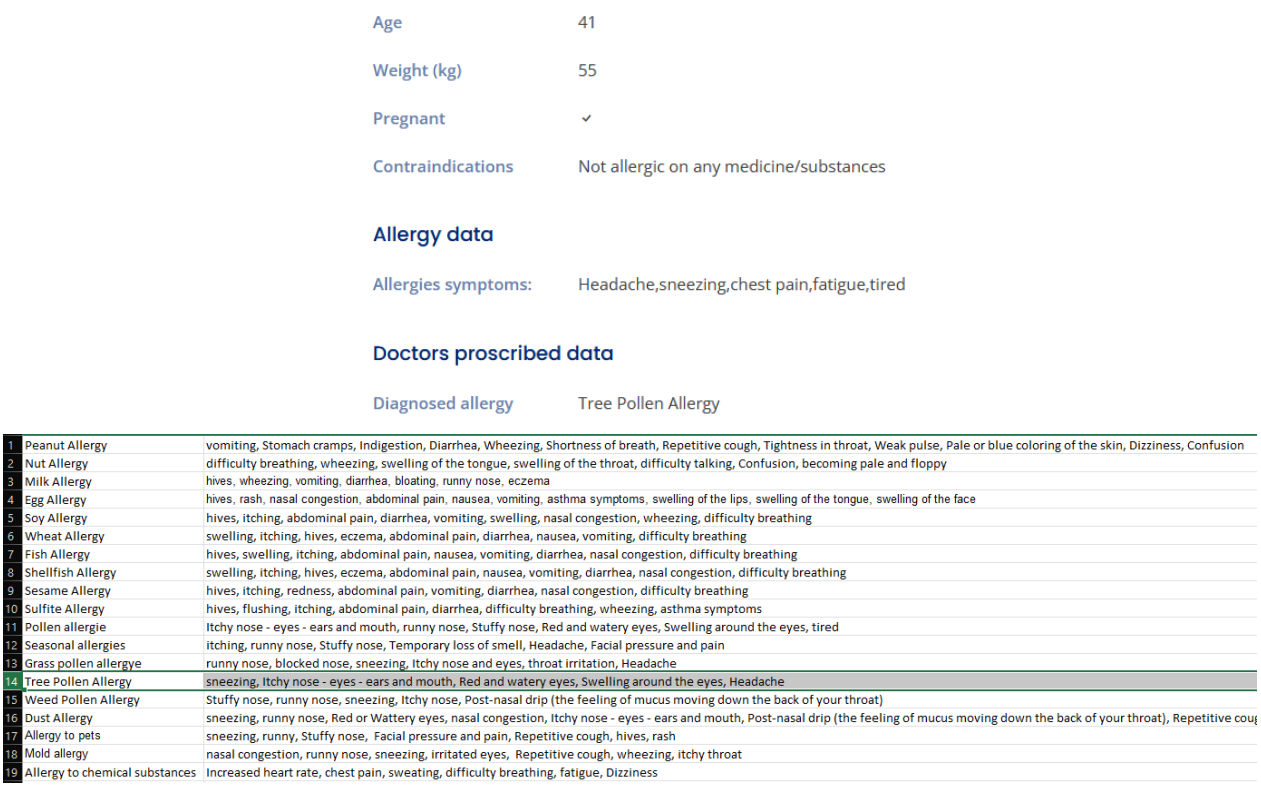
16	Wheat Allergy	EpiPen	The usual dose is 0.3 mg for intramuscular administration - two injectible doses needed to carry by yourself	18	FALSE	60
17	Wheat Allergy	EpiPen	For these patients, an EpiPen Jr auto-injector device containing 0.15 mg is available adrenaline per dose - tv	3	FALSE	15

[
Medicine name: EpiPen
Medicine description for EpiPen: For these patients, an EpiPen Jr auto-injector device containing 0.15 mg is available adrenaline per dose - two injectible doses needed to carry by yourself]

Slika 5.21. *Prikaz preporučenog lijeka*

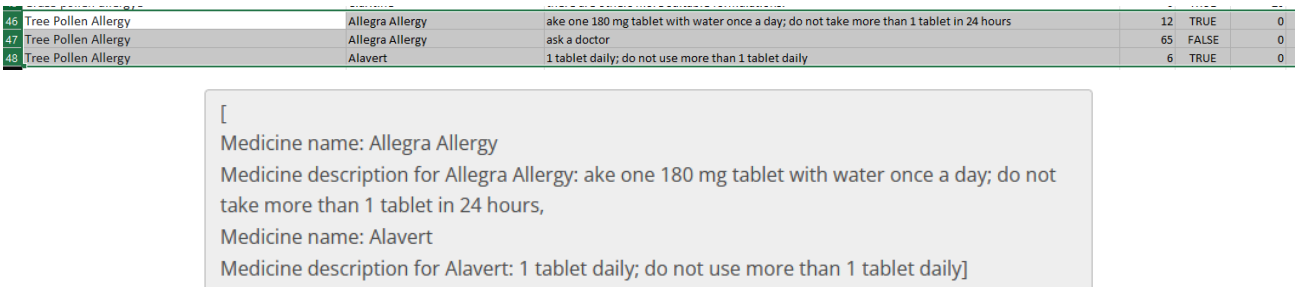
5.2.2. Primjer korištenja 2

U idućem primjeru pacijent ima 41 godinu i težinu 55 kg, te različit broj simptoma što je prikazano slikom 5.22. Ovoga puta odabrano je 5 simptoma i pacijent je trudan. Dijagnozom alergije se dolazi do rezultata da je pacijent alergičan na pelud drveća. Istim principom je određena alergija kao što je opisano u pod-poglavlju 4.2.1. Dobiva se lista postotaka podudaranja simptoma ([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 16.67, 16.67, 33.33, **40.0**, 20.0, 12.5, 14.29, 14.29, 33.33]) iz koje se uzima indeks najvećeg broja te se taj indeks traži u bazi.



Slika 5.22. *Prikaz pronađene alergije*

Nakon što je alergija pronađena, potrebno je obaviti preporuku lijeka. Pacijent je trudan, te je samim time očekivanje da 47 red na slici 5.23 ne bude preporučen zbog trudnoće. Pogledom na sliku 5.23, taj lijek nije preporučio te da su 2 lijeka koja odgovaraju pacijentovim unosima preporučeni. Zaključno tome, vidi se da preporuka lijeka radi ovisno o trudnoći pacijenta.

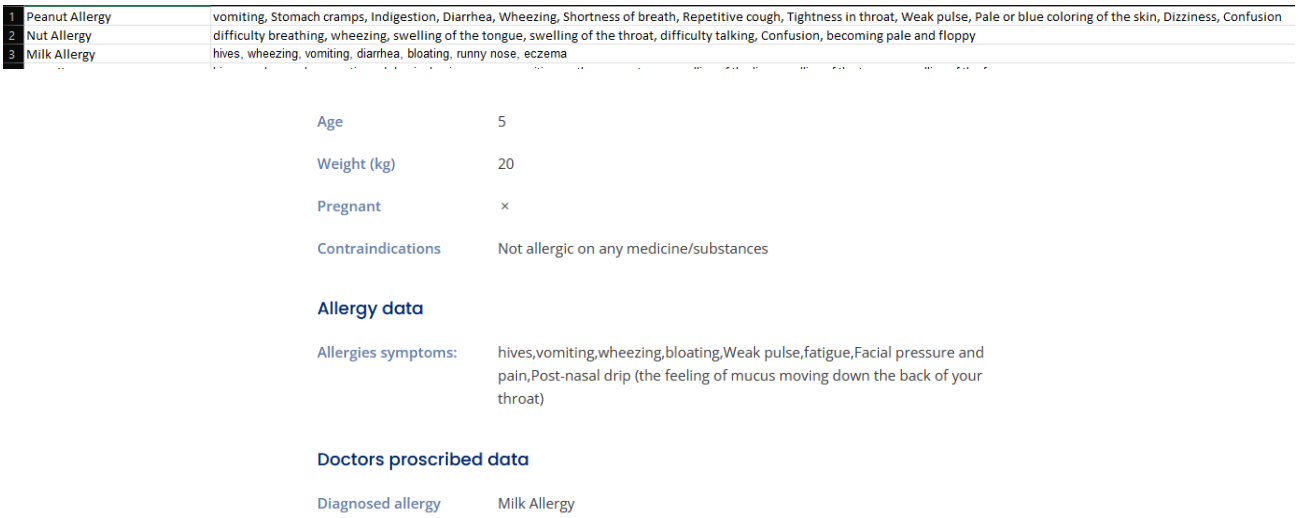


Slika 5.23. *Prikaz preporučenih lijekova*

5.2.3. Primjer korištenja 3

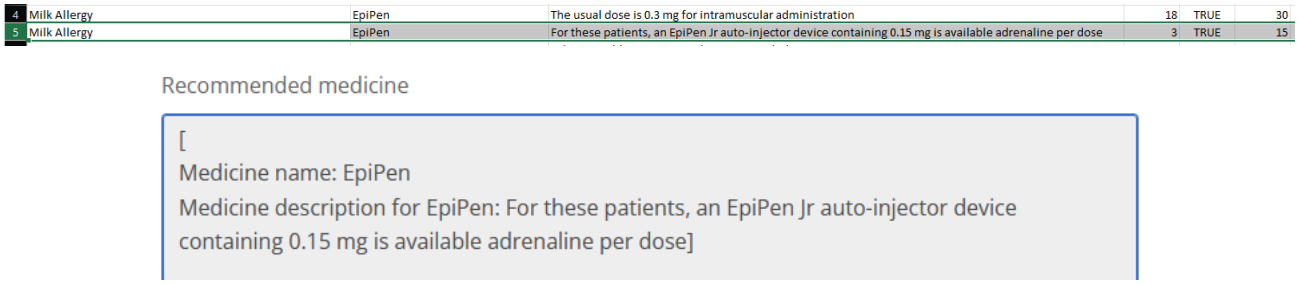
U ovome primjeru pacijent će se prikazati kao dijete. Pacijent će imati dob 5 godina, težinu 20kg, nije trudan i nema niti jednu prijašnje definiranu alergiju te će biti odabran maksimalan broj simptoma (8)

što je prikazano slikom 5.24. Odabirom simptoma već opisanom postupkom dolazi se do dijagnoze alergije koja je u ovome slučaju alergija na mlijeko. Postoci kojima je određena alergija su: [16.67, 14.29, **57.14**, 20.0, 33.33, 22.22, 22.22, 20.0, 25.0, 25.0, 0.0, 16.67, 0.0, 0.0, 20.0, 25.0, 28.57, 14.29, 16.67] te na drugom indeksu predstavlja najveći postotak koji u bazi prikazuje alergiju na mlijeko.



Slika 5.24. *Prikaz baze i profila pacijenta s dijagnosticiranom alergijom*

Nakon preporuke lijeka, unutar baze podataka lijekova, prema slici 5.25 postoje dva unosa za takav tip alergije. S obzirom na dob, izostavljen je prvi unos na slici zbog nedovoljnog broja godina pacijenta, te je preporučen drugi koji pokazuje točan rad algoritma.



Slika 5.25. *Prikaz baze i preporučenog lijeka*

Iz ovih triju opisanih testnih slučajeva zaključuje se da postupak dijagnosticiranja alergije i preporuke lijeka funkcionira u skladu s definiranim specifikacijama. Važno je napomenuti da nije obuhvaćena analiza procesa oporavka pacijenta nakon uporabe propisanog lijeka niti pohrana podataka o korištenom lijeku, budući da su ti koraci već prethodno opisani u programskom opisu rada i smjernicama za korištenje aplikacije.

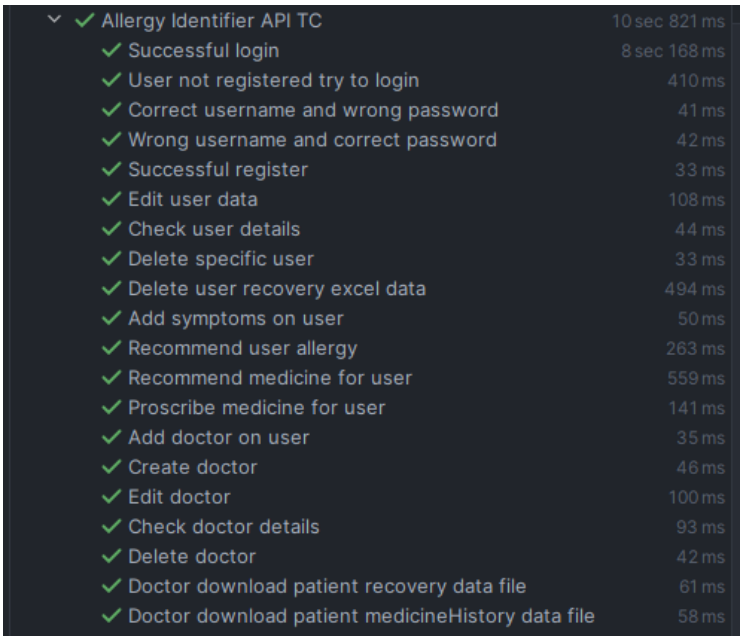
5.3. Prikaz rješenja API testiranjem

Unutar aplikacije postoji obilje API-ja koji se koriste kako bi se osiguralo pravilno i učinkovito funkcioniranje svih njezinih aspekata. U sklopu temeljitog pristupa testiranju, pažljivo su odabrani API koji su ključni za simuliranje ključnih funkcija aplikacije. Ovi API-ji su pažljivo testirani kako bismo osigurali da aplikacija pruža izvanredno korisničko iskustvo. Među testiranim API-ima su oni koji omogućavaju simulaciju procesa kao što su dijagnoza alergije, preporuka lijekova, uređivanje postavki za pacijente i liječnike, te preuzimanje Excel datoteka koje sadrže informacije o tijeku oporavka pacijenata i tijeku njihovog liječenja. Ovi API-ji su odabrani zbog njihovog ključnog doprinosa u optimiziranju funkcionalnosti aplikacije te pružanju sigurnog i korisnog okruženja za pacijente i medicinske profesionalce. Važno je napomenuti da, dok su neki drugi API-ji, poput onih koji se odnose na kontaktiranje između pacijenata i liječnika, testirani ručno, ovi API-ji nisu uključeni u ovu fazu testiranja jer se kroz ručno testiranje već osigurala visoka kvaliteta i funkcionalnost tih aspekata aplikacije. Cilj je osigurati da svaki aspekt aplikacije radi besprijekorno i pruža pouzdano iskustvo korisnicima. Stoga se usredotočilo na ključne API-je koji najviše doprinose tom cilju, kako bi se osigurala najbolja moguća kvaliteta proizvoda. Popis API-a unutar aplikacije i API koji su obuhvaćeni u API testiranju prikazani su tablicom 5.1.

Način pisanja cucumber API testova opisan je u potpoglavlju 3.2.6. tako da način pisanja ovdje ponovno neće biti objašnjavan. Pokretanjem svih testova dobiva se dokaz o uspješnome izvođenju svakog od napisanih API testova što je prikazano slikom 5.26. Svaki od API testova je napisan da ili sam ili unutar pokretanja skupine testova (regresija) može odraditi zadatak. S ovim rezultatom, zaključuje se da aplikacija radi kako je zamišljena. Jedan od ključnih razloga korisnosti testnih slučajeva za API-je je njihova sposobnost otkrivanja problema u ranim fazama razvoja. Testirajući API izolirano, moguće je identificirati i ispraviti probleme prije nego što se integrira u složeni sustav. To sprječava potencijalno ozbiljne probleme koji bi se mogli pojaviti u kasnijim fazama razvoja ili čak u produkcijskom okruženju. Automatizacija testiranja API-ja dodatno povećava njihovu vrijednost. Automatizirani testovi omogućuju brzo i dosljedno izvođenje testova bez potrebe za ručnim intervencijama. To ubrzava proces testiranja i omogućuje često ponavljanje testova kako bi se brzo identificirali i riješili problemi. Na slici 5.26.a također se vidi izvještaj koji se generira nakon izvođenja testova unutar alata InteliJ te sliku 5.26.b koja prikazuje cucumber web prikaz rezultata testiranja koji olakšavaju pregled izvršenih testova.

Tablica 5.1. *Prikaz korištenih API-ja tijekom testiranja*

API korišteni u testiranju	Opis
/doctors-patients/confirmRecommendedMedicine/{id}	Potvrda preporučenog lijeka za pacijenta sa određenim ID-em
/addSymptomsOnUser/add	Potvrda dodanih simptoma za korisnika
/attachPatient	Povezivanje pacijenta sa liječnikom
/checkDoctorParams	Provjera parametara liječnika
/checkUserParams	Provjera parametara korisnika
/deleteExcelFile	Brisanje Excel datoteke
/deleteMedicineHistory/{id}	Brisanje povijesti lijekova za pacijenta sa određenim ID-em
/doctors-patients/doctorAccessPatientDetails/{id}	Pregled detalja pacijenta za liječnika sa određenim ID-em
/doctors-patients/proscribeMedicine	Propisivanje lijeka pacijentu
/download/{id}	Preuzimanje datoteke za pacijenta sa određenim ID-em
/downloadMedicineHistory/{id}	Preuzimanje povijesti lijekova za pacijenta sa određenim ID-em
/identifyUserAllergy	Identifikacija alergije za korisnika
/pages-all_doctors/delete/{id}	Brisanje liječnika sa određenim ID-em
/pages-all_doctors/details/{id}	Pregled detalja liječnika sa određenim ID-em
/pages-all_doctors/edit/{id}	Uređivanje podataka liječnika sa određenim ID-em
/pages-all_doctors/saveDoctor	Spremanje novog liječnika
/pages-all_doctors/update/{id}	Ažuriranje podataka liječnika sa određenim ID-em
/pages-all_users/delete	Brisanje korisnika
/pages-all_users/details/{id}	Pregled detalja korisnika sa određenim ID-em
/pages-all_users/edit/{id}	Uređivanje podataka korisnika sa određenim ID-em
/pages-login	Prijavljivanje u sustav
/pages-register	Registracija novog korisnika
/proscribeMedicine	Propisivanje lijeka



a)

Scenario: Recommend medicine for user

- **Given** I register as user with data: name: "test", username: "test", password: "test", email: "test@gmail.com", age: "23", pregnant: "true", weight: "20"
- **When** the user logs in with username: "test" and password "test"
- **And** I add "swelling of the throat,swelling,Confusion,becoming pale and floppy,sneezing,Facial pressure and pain" symptoms on "test" user
- **Then** I check if user "test" contains property "symptoms" with data "swelling of the throat,swelling,Confusion,becoming pale and floppy,sneezing,Facial pressure and pain"
- **And** I recommend allergy for "test"
- **Then** I check if user "test" contains property "detectedAllergy" with data "Nut Allergy"
- **And** I recommend medicine for user "test"

Scenario: Proscribe medicine for user

- **Given** I register as user with data: name: "test", username: "test", password: "test", email: "test@gmail.com", age: "23", pregnant: "true", weight: "20"
- **When** the user logs in with username: "test" and password "test"
- **And** I add "swelling of the throat,swelling,Confusion,becoming pale and floppy,sneezing,Facial pressure and pain" symptoms on "test" user
- **Then** I check if user "test" contains property "symptoms" with data "swelling of the throat,swelling,Confusion,becoming pale and floppy,sneezing,Facial pressure and pain"
- **And** I recommend allergy for "test"
- **Then** I check if user "test" contains property "detectedAllergy" with data "Nut Allergy"
- **And** I recommend medicine for user "test"
- **And** I prescribe medicine "Aspirin" and medication usage "once per day" for user "test"
- **Then** I check if user "test" contains property "proscribedMedicine" with data "Aspirin"

b)

Slika 5.26. *Prikaz uspješnog izvođenja API testova unutar alata InteliJ (a) i cucumber web prikaza rezultata (b)*

6. ZAKLJUČAK

Web aplikacije izrađena u sklopu ovog rada predstavlja korak naprijed u olakšavanju dijagnosticiranja i upravljanja alergijskim bolestima. Ova aplikacija je rezultat istraživanja i implementacije, s jasnim ciljem poboljšanja kvalitete života pacijenata te smanjenja utjecaja alergijskih bolesti na njihovo opće zdravlje. Putem analize različitih vrsta alergija, uključujući alergije na pelud, hranu i inhalirane alergene, aplikacija omogućuje korisnicima dublje razumijevanje njihovih simptoma i potencijalnih uzroka. Pomoću implementiranog modela preporuka za dijagnosticiranje alergija na temelju simptoma, korisnicima se pružaju korisne smjernice koje olakšavaju identifikaciju mogućih alergena. Uz to, omogućena je analiza primjene odgovarajućih lijekova za već dijagnosticirane alergije što omogućuje da korisnici budu informirani o različitim terapijskim mogućnostima. Također, aplikacija omogućuje korisne informacije korisnicima i medicinskim stručnjacima.

Web aplikacija programski je ostvarena koristeći MVC predložak programske arhitekture koji se svojom strukturom uklopio u programsko rješenje i olakšao njegovu implementaciju, a ponajviše prikaz podataka na web stranici zbog jednostavne komunikacije između kontrolera i prikaza. Analiza ostvarene web aplikacije s ciljem utvrđivanja njenih mogućnosti provedena je automatskim testiranjem koje je osiguralo lakše izvođenje testova, te ručnim testiranjem, gdje su odabrana tri različita slučaja korištenja na kojima se temeljito testiralo i prikazalo očekivane rezultate. Rezultati analize pokazuju da predloženo programsko rješenje radi prema propisanim specifikacijama. Razvoj i implementacija web aplikacije temeljila se i na sveobuhvatnom testiranju. Skup pažljivo osmišljenih API testova doprinio je ranom otkrivanju potencijalnih problema koji su se najviše pojavljivali pri spremanju podataka korisnika tijekom cijelog procesa dijagnoze alergije i preporuke lijeka, a automatiziranje testiranja provedeno je za testove koji simuliraju cijeli proces dijagnoze alergije i preporuke lijeka. Testovi koji zahtjevaju osobnu procjenu i ideju obavljeni u ručno. Ključno je istaknuti da ova web aplikacije može poduprijeti, ali ne može zamijeniti stručno medicinsko mišljenje i vođenje liječenja.

7. LITERATURA

[1] Alergije na pelud, ZZJZDNZ, Zdravlje i okoliš, dostupno na: https://www.zzjzdnz.hr/zdravlje/okolis-i-zdravlje/480s, pristup: 24.6.2023.

[2] Pollen Allergy, AAFA - Asthma and Allergy Foundation of America, dostupno na: https://www.aafa.org/allergies/types-of-allergies/pollen-allergy/, pristup 22.6.2023.

[3] R. Pawankar, G.W. Canonica, S.T. Holgate, R.F. Lockey, White Book on Allergy, WAO, Wisconsin, 2013, dostupno na: https://www.worldallergy.org/UserFiles/file/WhiteBook2-2013-v8.pdf , pristup: 27.6.2023.

[4] H. Steinman, Tree pollens TFS, Thermo, Sweden, 2008, dostupno na: http://www.immunocapexplorer.com/uploads/cms/asset_brick/asset/10414/52-5107-91_02-TreePollens.pdf,pristup:27.6.2023.

[5] R. Ogino, Y Chinuki , T Yokooji , D. Takizawa , H. Matsuo, E. Morita, Identification of Peroxidase-1 and Beta-Glucosidase as Cross-Reactive wheat Allergens in Grass Pollen-Related Wheat Allergy , Allergology International, Vol. 70, Issue 2, April 2021, pp. 215-222, dostupno na: https://www.sciencedirect.com/science/article/pii/S1323893020301313, pristup: 27.6.2023.

[6] Allergopharma. (n.d.). Patient Information Download: Pollen Allergy. https://www.allergopharma.com/fileadmin/user_upload/allergopharma-com/Patients_information_download_Pollen_allergy.pdf, pristup: 27.6.2023.

[7] A. Muraro, Managing Food Allergy: GA2LEN guideline 2022, World Allergy Organization Journal, Vol. 15, Issue 9, 100687, 2022, dostupno na: https://www.worldallergyorganizationjournal.org/article/S1939-4551(22)00063-1/fulltext , pristup: 27.6.2023.

[8] National Health Service, Food Allergy, dostupno na: https://www.nhs.uk/conditions/food-allergy/, pristup: 23.6.2023.

[9] J.W. Mims, M.C. Veling, Inhalant Allergies in Children, Otolaryngologic Clinics of North America, Vol. 44, Issue 3, June 2011, pp. 797-814, dostupno na: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7172761/pdf/main.pdf , pristup: 27.6.2023.

[10] German Center for Research and Innovation - DWIH New York. ADA - Asthma, Dust and Allergy. https://www.dwih-newyork.org/en/2021/04/06/ada/ , pristup: 23.7.2023

[11] MedIately. (n.d.). Home. https://mediately.co/hr, pristup: 23.7.2023.

[12] MedIately. (n.d.). EpiPen 0.3 mg otopina za injekciju u napunjenoj brizgalici - Doziranje. https://mediately.co/hr/drugs/MY0GTGiw0CMPrFbCDAQGgpFOOh9/epipen-0-3-mg-otopina-za-injekciju-u-napunjenoj-brizgalici#dosing, pristup: 23.7.2023.

[13] Medscape. (n.d.). Drugs, Diseases, & Procedures. https://reference.medscape.com/, pristup: 23.7.2023.

[14] Medscape. (n.d.). Epipen Jr., epinephrine. https://reference.medscape.com/drug/epipen-jr-epinephrine-342437, pristup: 23.7.2023.

[15] AltexSoft. (n.d.). Functional and Non-Functional Requirements: Specification and Types. https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/, pristup: 23.7.2023.

[16] A. Deluka-Tibljaš, B. Karleuša, N. Dragičević, Pregled primjene metoda višekriterijske analize pri donošenju odluka o prometnoj infrastrukturi. Građevinar, 65(7), str. 619-631, 2013, dostupno na: https://hrcak.srce.hr/file/156408, pristup: 23.7.2023.

[17] J. Deacon, Model-View-Controller (MVC) Architecture, dostupno na: http://www.johndeacon.net/john-deacon/articles/model-view-controller-architecture/, pristup: 23.7.2023.

[18] M.V. Zlatić, R. Kaurić, I. Radonjić, MVC Architecture: A Comparative Study Between Laravel Framework and Slim Framework in Freelancer Project Monitoring System Web Based, Procedia Computer Science, Vol. 157, 2019, pp. 134-141, 2019, dostupno na: https://www.sciencedirect.com/science/article/pii/S1877050919310683 pristup: 25.7.2023.

[19] Mozilla Developer Network, MVC (Model-View-Controller). https://developer.mozilla.org/en-US/docs/Glossary/MVC, pristup: 25.7.2023.

[20] Oracle, What is Java? https://www.java.com/en/download/help/whatis_java.html, pristup: 26.7.2023.

[21] E. Stević, Z. Jatić, E. Salihefendić, A. Kadić, Dobra praksa propisivanja i izdavanja lijekova. Sarajevo: Ministarstvo zdravstva Kantona Sarajevo, Institut za naučno-istraživački rad i razvoj Kliničkog centra Univerziteta u Sarajevu, 2011, dostupno na: https://mz.ks.gov.ba/sites/mz.ks.gov.ba/files/Dobra%20praksa%20propisivanja%20i%20izdavanja%20lijekova.pdf , pristup: 19.8.2023.

[22] L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos, Non-Functional Requirements in Software Engineering in Conceptual Modeling: Foundations and Applications, Springer US, pp 363–379,

2000, dostupno na: https://link.springer.com/chapter/10.1007/978-3-642-02463-4_19, pristup: 19.8.2023.

[23] P. Rieck, Test Thymeleaf Controller Endpoints with Spring Boot and MockMvc, dostupno na: https://rieckpil.de/test-thymeleaf-controller-endpoints-with-spring-boot-and-mockmvc/, pristup: 19.8.2023.

[24] E. Triantaphyllou, Multi-Criteria Decision Making Methods, Multi-criteria Decision Making Methods: A Comparative Study, Springer US, Vol. 44, pp 5–21, 2000, dostupno na: https://link.springer.com/chapter/10.1007/978-1-4757-3157-6_2, pristup: 19.8.2023.

[25] J. Smith, The Journal of Allergy and Clinical Immunology: In Practice 2017 Year in Review, The Journal of Allergy and Clinical Immunology: In Practice, Vol. 6, Issue 2, March–April 2018, pp 328-352, dostupno na: https://www.sciencedirect.com/science/article/pii/S2213219817310437, pristup: 19.8.2023.

[26] Y. Skaf, R. Laubenbacher, Topological data analysis in biomedicine: A review , Journal of Biomedical Informatics, Vol. 130, June 2022., 104082, dostupno na: https://www.sciencedirect.com/science/article/pii/S1532046422000983, pristup: 19.8.2023.

[27] Dust Allergies, ACAAI, dostupno na: https://acaai.org/allergies/allergic-conditions/dust-allergies/, pristup: 19.8.2023.

[28] D. Westerveld, API Testing and Development with Postman, Packt, Mumbai, 2021. dostupno na: https://download.bibis.ir/Books/other/2021/API-Testing-and-Development-with-Postman-A-practical-guide-to-creating,-testing,-and-managing-APIs-for-automated-software-testing-by-Dave-Westerveld-(bibis.ir).pdf , pristup: 17.8.2023.

SAŽETAK

Web aplikacija ostvarena u ovom radu predstavlja sveobuhvatno rješenje za olakšavanje dijagnosticiranja i upravljanja alergijskim bolestima. Aplikacija se temelji na modernoj arhitekturi MVC. Pruža korisnicima relevantne informacije i smjernice za bolje razumijevanje i suočavanje s alergijskim reakcijama. Za razvoj korisničkog sučelja korišteni su HTML, CSS i Thymeleaf za dinamički prikaz informacija. JavaScript i Bootstrap omogućuju interaktivnost i poboljšano korisničko iskustvo. Java je izabrana za implementaciju poslovne logike i obrade podataka na poslužiteljskoj strani. Za komunikaciju s bazom podataka, aplikacija koristi sustav PostgreSQL, koja omogućuje pohranu, rukovanje i dohvat podataka relevantnih za alergijske reakcije i njihove uzroke. Aplikacija omogućuje korisnicima unos simptoma. Na temelju unesenih podataka, aplikacija stvara preporuke za dijagnosticiranje alergija na temelju simptoma. Također, aplikacija omogućuje analizu uspješnosti primjere odgovarajućih lijekova za već dijagnosticirane alergije. To korisnicima pruža informirane smjernice za postizanje najboljeg mogućeg ishoda u liječenju i omogućuje bolje razumijevanje, dijagnosticiranje i upravljanje alergijskim reakcijama.

Ključne riječi: alergije, API testiranje, praćenje tijeka liječenja, preporuka lijeka, web aplikacija.

ABSTRACT

Development and Testing of a Web Application with a Recommendation Generation System to Support the Treatment of Allergic Diseases

The web application developed in this project serves as a comprehensive solution to simplify the diagnosis and management of allergic conditions. It is constructed using a contemporary MVC architecture. The application furnishes users with pertinent information and guidance to enhance their comprehension of and coping with allergic responses. To craft the user interface, HTML, CSS, and Thymeleaf are utilized to dynamically present information. JavaScript and Bootstrap are harnessed to facilitate interactivity and elevate the user experience. Java is selected for implementing the business logic and data processing on the server side. In terms of database communication, the application employs the PostgreSQL system, enabling storage, manipulation, and retrieval of data pertaining to allergic reactions and their triggers. Users are empowered to input their symptoms into the application. Using the provided data, the application generates suggestions for diagnosing allergies based on the symptoms entered. Moreover, the application allows for an assessment of the efficacy of utilizing suitable medications for allergies that have already been diagnosed. This equips users with informed recommendations to achieve optimal outcomes in treatment and enriches the comprehension, diagnosis, and management of allergic reactions.

Keywords: allergies, API testing, treatment progress tracking, medication recommendation, web application.

ŽIVOTOPIS

Ivan Staković rođen 8.11.1999. godine u Nizozemskoj u gradu Rotterdamu. Osnovnu školu pohađao je u Sesvetama pod nazivom Osnovna škola Sesvetska Sopnica. Nakon toga upisuje elektrotehničku srednju školu u Jelkovcu koju završava 2018. godine i upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer preddiplomski studiji Računarstvo**.** Godine 2020/2021. upisuje prvu godinu diplomskog sveučilišnog studija računarstvo smjera Informacijske i podatkovne znanosti. Krajem 2020. godine počinje raditi kao *tester* i *front-end-developer* u kompaniji Ericsson Nikola Tesla d.d**.**

PRILOZI

Prilog 1. Diplomski rad u DOCX obliku

Prilog 2. Diplomski rad u PDF obliku

Prilog 3. Programski kod web aplikacije za dijagnozu alergije i preporuku prikladnih lijekova