Razvoj i usporedba web sustava za potporu u dijagnosticiranju i liječenju bolesti korištenjem strojnog učenja i generativne umjetne inteligencije

Šušovček, Antonio

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

Sveučilišni studij

RAZVOJ I USPOREDBA WEB SUSTAVA ZA POTPORU U DIJAGNOSTICIRANJU I LIJEČENJU BOLESTI KORIŠTENJEM STROJNOG UČENJA I GENERATIVNE UMJETNE INTELIGENCIJE

Diplomski rad

Antonio Šušovček

Osijek, 2024.

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

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Antonio Šušovček
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D1332R, 07.10.2022.
JMBAG:	0165082573
Mentor:	prof. dr. sc. Goran Martinović
Sumentor:	
Sumentor iz tvrtke:	Davor Aleksić, mag.ing.comp.
Predsjednik Povjerenstva:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	prof. dr. sc. Goran Martinović
Član Povjerenstva 2:	izv. prof. dr. sc. Alfonzo Baumgartner
Naslov diplomskog rada:	Razvoj i usporedba web sustava za potporu u dijagnosticiranju i liječenju bolesti korištenjem strojnog učenja i generativne umjetne inteligencije
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U teorijskom dijelu diplomskog rada potrebno je opisati probleme i izazove pri dijagnosticiranju i liječenju kod bolesti, te mogućnosti korištenja strojnog učenja, generativne umjetne inteligencije i promptnog inženjeringa za potporu liječniku. Također, na temelju analize stanja u području i postojećih rješenja, definirati funkcionalne i nefunkcionalne zahtjeve na web sustav, predložiti model i arhitekturu web sustava na strani korisnika i na strani poslužitelja. U cilju potpore dijagnosticiranju i liječenju bolesti, treba prilagoditi skup podataka, definirati značajke modela
Datum ocjene pismenog dijela diplomskog rada od strane mentora:	09.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	12.09.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	13.09.2024.

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

IZJAVA O IZVORNOSTI RADA

Osijek, 13.09.2024.

Ime i prezime Pristupnika:	Antonio Šušovček
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D1332R, 07.10.2022.
Turnitin podudaranje [%]:	10

Ovom izjavom izjavljujem da je rad pod nazivom**: Razvoj i usporedba web sustava za potporu u dijagnosticiranju i liječenju bolesti korištenjem strojnog učenja i generativne umjetne inteligencije**
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 pristupnika:

Sadržaj

1. UVOD

U posljednjih nekoliko desetljeća, moguće je posvjedočiti značajnim tehnološkim napredcima koji su transformirali gotovo sve aspekte ljudskog života, uključujući i područje medicine. Razvoj novih tehnologija, poput strojnog učenja i generativne umjetne inteligencije, omogućuje stvaranje naprednih sustava koji mogu značajno unaprijediti proces dijagnosticiranja i liječenja različitih bolesti. Jedno od područja primjene ovih tehnologija je upravo dijabetes, kronična bolest koja pogađa milijune ljudi diljem svijeta i zahtjeva stalno praćenje i prilagodbu liječenja.

Ovaj diplomski rad bavi se razvojem i usporedbom web sustava za potporu u dijagnosticiranju i liječenju bolesti korištenjem strojnog učenja i generativne umjetne inteligencije. U radu se istražuju izazovi i mogućnosti primjene naprednih tehnologija u medicini, s posebnim naglaskom na dijabetes, kroničnu bolest koja pogađa milijune ljudi diljem svijeta. Opisana su dva najčešća tipa dijabetesa, tip 1 i tip 2, te čimbenici rizika povezani s njihovim razvojem. Također, prikazan je pregled postojećih rješenja koja koriste strojno učenje i generativnu umjetnu inteligenciju u biomedicinske svrhe. Cilj rada je razviti i analizirati web sustav koji može pomoći u pravovremenoj dijagnozi i praćenju bolesti, čime se značajno može poboljšati kvaliteta života pacijenata.

U drugom poglavlju prikazani su izazovi i pregled stanja u području korištenja strojnog učenja i generativne umjetne inteligencije u liječenju bolesti, s posebnim naglaskom na dijabetes, njegove tipove i čimbenike rizika. Treće poglavlje detaljno opisuje model i arhitekturu web sustava, uključujući funkcionalne i nefunkcionalne zahtjeve, građu i komponente sustava, te korištene algoritme strojnog učenja i pristupe generativne umjetne inteligencije. U četvrtom poglavlju objašnjava se programsko rješenje web sustava te razvoj i implementacija modela strojnog učenja i generativne umjetne inteligencije u sustav. Peto poglavlje posvećeno je prikazu rada web aplikacije, ispitivanju njezine funkcionalnosti i analizi rezultata.

2. IZAZOVI I PREGLED STANJA U PODRUČJU KORIŠTENJA STROJNOG UČENJA I GENERATIVNE UMJETNE INTELIGENCIJE U LIJEČENJU BOLESTI I DIJABETESA

U ovom poglavlju opisano je što je to dijabetes, koji tipovi dijabetesa postoje te čimbenici rizika koji mogu ukazivati na bolovanje od dijabetesa. Također, prikazana su postojeća rješenja koja koriste strojno učenje i generativnu umjetnu inteligenciju u biomedicinske svrhe.

2.1 Tipovi dijabetesa

Dijabetes je kronična bolest koja se pojavljuje kada tijelo ne može regulirati razinu glukoze u krvi. Zbog toga tijekom duljeg razdoblja postoji visoka razina šećera u krvi. Ostane li neliječen, može izazvati mnoge komplikacije poput bolesti srca, moždanog udara, zatajenja bubrega itd [1]. Postoji više vrsta dijabetesa, a najčešći su dijabetes tipa 1 i tipa 2.

2.1.1 Dijabetes tipa 1

Tip 1 je autoimuna bolest gdje imunološki sustav napada stanice gušterače koje su zadužene za proizvodnju inzulina. Razlog bolesti još uvijek nije jasno poznat, no znanstvenici smatraju da je njegova pojava uzrokovana genima i čimbenicima okoline. Bolest se uobičajeno pojavljuje u osobama mlađim od 20 godina, no moguće je pojavljivanje i kod odraslih osoba. Zbog činjenice da njihovo tijelo ne proizvodi inzulin, osobe koje boluju od tipa 1 trebaju uzimati inzulin svakodnevno putem injekcija ili inzulinskih pumpi.

2.1.2 Dijabetes tipa 2

Tip 2 je najčešći oblik dijabetesa. Moguće ga je razviti u bilo kojoj dobi, i uzrokuju ga različiti čimbenici rizika. Obično počinje s otpornosti na inzulin koji proizvodi gušterača tj. stanice tijela ne iskorištavaju inzulin kako bi trebale. Zbog toga gušterača konstatno povećava količinu inzulina koju proizvodi kako bi držala korak s potražnjom tijela, sve dok ne dođe do situacije gdje više ne može proizvoditi inzulin u dovoljnim količinama. Moguće je usporiti razvoj tipa 2 ili ga u potpunosti spriječiti tako da osoba kontrolira vlastitu prehranu, bavi se tjelovježbom i obraća pažnju na čimbenike rizika [2]. Ova vrsta dijabetesa obrađena je u radu.

2.2 Čimbenici rizika dijabetesa

Svi daljnje navedeni čimbenici rizika korišteni su unutar web sustava kako bi se pomoću njih analizirala mogućnost bolovanja pacijenta od dijabetesa.

2.2.1 HbA1c test

Glukoza je šećer koji je glavni izvor energije za ljudsko tijelo. Prenosi se krvlju do svih stanica u tijelu kako bi stanice imale energije za obavljanje vlastitih funkcionalnosti. Postoji nekoliko važnih čimbenika koji razinu glukoze održavaju na normalnoj razini, a među njima je najbitniji inzulin, koji se stvara u gušterači čovjeka. Gušterača osobe koja boluje od dijabetesa ne može proizvoditi inzulin ili ga ne proizvodi u dovoljnim količinama, što može dovesti do hiperglikemije tj. povećane razine šećera u krvi. Jedan od ključnih testova za praćenje kontrole šećera u krvi kod osoba sa dijabetesom je HbA1c test. Ovaj test mjeri prosječan nivo glukoze u krvi tijekom posljednjih 2-3 mjeseca, a ne trenutnu koncentraciju glukoze u krvi kao što je slučaj sa standardnim testovima glukoze. Iako stroga granica nije određena, smatra se da svaka osoba čiji HbA1c test pokazuje više od 6% može biti u riziku obolijevanja od dijabetesa [3].

2.2.2 Mjerenje razine kreatinina

Kreatinin je otpadni proizvod koji nastaje razgradnjom kreatina, molekule koja je važna za proizvodnju energije u mišićima. Kreatinin se otpušta u krvotok i filtrira kroz bubrege te izlučuje urinom. Važan je pokazatelj zdravlja bubrega jer je lako mjerljiv nusproizvod metabolizma koji se izlučuje nepromijenjen putem bubrega. Povećane razine kreatinina mogu biti znak da su bubrezi oštećeni ili ne funkcioniraju pravilno. Razina kreatinina se može testirati unutar krvi te unutar mokraće. Normalni rasponi kreatinina u krvi su od 0.8 do 1.4 mg/dL za muškarce te od 0.6 do 1.2 mg/dL za žene. Razina u mokraći se testira mjerenjem omjera albumina i kreatinina. Albumin je protein u krvi koji zdravi bubrezi zadržavaju i ne izlučuju ga u mokraći. Omjer opisuje koliko albumina ima u uzorku mokraće u odnosu na količinu kreatinina. Rezultati se izražavaju kao broj miligrama albumina na svaki gram kreatinina. Rezultati višlji od 17 mg/g za muškarce i 25 mg/g mogu ukazivati na probleme s bubrezima koji mogu biti uzrokovani od dijabetesa [4].

2.2.3 Indeks tjelesne mase

Indeks tjelesne mase (eng. *Body Mass Index* - BMI) je okvirni pokazatelj pretilosti osobe. Računa se na način da se tjelesna masa osobe u kilogramima podijeli s kvadratom visine u metrima. Prema Svjetskoj zdravstvenoj organizaciji, BMI se kategorizira na sljedeći način:

- Pothranjenost: BMI manji od 19.9
- Normalna tjelesna masa: BMI od 20 do 24.9
- Prekomjerna tjelesna masa: BMI od 25 do 29.9
- Pretilost (klasa I): BMI od 30 do 34.9
- Pretilost (klasa II): BMI od 35 do 39.9

- Teška pretilost (klasa III): BMI 40 ili više

Visok BMI povezan je s povećanim rizikom od razvoja dijabetesa tipa 2 zbog nakupljanja masnog tkiva koje može rezultirati povećanom otpornosti stanica na inzulin [5].

2.2.4 Urea

Urea je kemijski spoj koji nastaje kao produkt metabolizma proteina u jetri. Praćenje razine uree u krvi koristi se u medicini za procjenu funkcije bubrega i općeg metabolizma proteina, što je važno u dijagnosticiranju i praćenju različitih zdravstvenih stanja, uključujući dijabetes. Normalne vrijednosti uree mogu varirati ovisno o laboratoriju, ali općenito se kreću između 2.5 i 7.5 mmol/L (15 do 40 mg/dL) [6].

2.2.5 Kolesterol i trigliceridi

Kolesterol je važan lipid u tijelu koji se koristi za izgradnju staničnih membrana, proizvodnju hormona i vitamina D, te je ključan za mnoge biološke procese. Triglicerid je lipid koji se stvara kako bi se pohranio u masnim stanicama. Kada je tijelu potrebna energija, oni se otpuštaju iz masnih stanica u krvotok. Uz kolesterol i trigliceride su vezane sljedeće vrste lipoproteina:

- LDL: Poznat kao "loš" kolesterol, jer visoke razine LDL mogu doprinijeti razvoju ateroskleroze i srčanih bolesti. On nosi kolesterol koji se nakuplja na krvnim žilama.
- HDL: Poznat kao "dobar" kolesterol, jer visoke razine HDL pomažu uklanjati višak kolesterola iz krvi i smanjuju rizik od srčanih bolesti.
- VLDL: Također poznat kao “loš” kolesterol, VLDL prenosi trigliceride i kolesterol do tkiva.

Osobe koje boluju od dijabetesa često imaju niske razine HDL te povišene razine LDL i VLDL kolesterola [7].

2.3 Pregled stanja u području liječenja i prevencije dijabetesa

Liječenje i prevencija dijabetesa se može izvoditi na razne načine. Preporučeni načini su:

- Promjena prehrane: prehrana s niskim udjelom šećera te bogata vlaknima, voćem i povrćem pomaže u održavanju stabilne razine glukoze u krvi. Smanjenje unosa kalorija može pomoći smanjenju tjelesne težine, što smanjuje rizik od oboljenja.
- Redovita tjelovježba: pomaže tijelu da učinkovitije iskoristi inzulin i smanjuje tjelesnu težinu. Također može smanjiti potrebu za uzimanjem lijekova.

- Inzulin: manjak inzulina koji stvara gušterača se može nadomjestiti injekcijama ili inzulinskom pumpom. Postoji više vrsta inzulina ovisno o brzini djelovanja.
- Redovite kontrole: odsutnost simptoma se ne treba smatrati pokazateljem učinkovite kontrole. Učestalost praćenja simptoma ovise o vrstama liječenja i trebaju se odrediti u dogovoru s liječnikom. Metode kontrole uključuju: ispitivanje glukoze u mokraći, mjerenje glukoze u krvi te HbA1c testovi.
- Edukacija: ljudi koji boluju ili su u riziku od bolovanja od dijabetesa bi se trebali informirati o bolesti i kako pratiti njene simptome. Time bolje razumiju kako različiti čimbenici utječu na razvitak bolesti i kako se najbolje nositi s njome. Ovaj korak uključuje poznavanje svih prijašnje opisanih koraka [8].

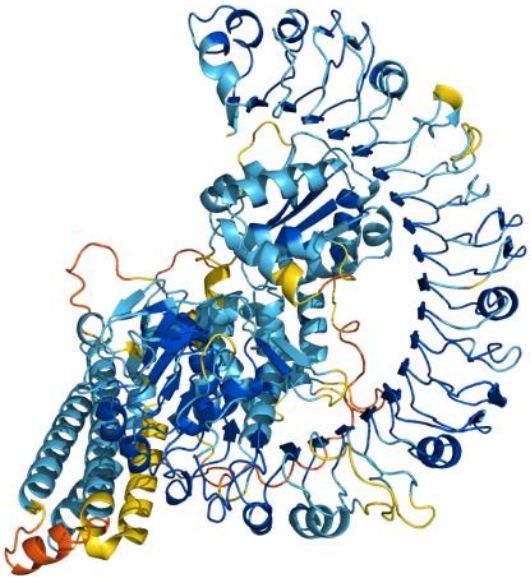
2.4 Pregled stanja u korištenju strojnog učenja i generativne umjetne inteligencije u biomedicini

Strojno učenje i generativna umjetna inteligencija može donijeti velike promjene u dijagnostici i liječenju bolesti. Mogućnost analize velikih skupova podataka koji su povezani s medicinskim terapijama može promijeniti medicinu u disciplinu koja se usmjerava na podatke i ishode koji rezultiraju u dijagnosticiranju i liječenju bolesti na način koji bi bio teže uočljiv tradicionalnim metodama. Podatci se već sada skupljaju na razne načine: trodimenzionalne slike organa, prikupljanje medicinskih nalaza pacijenata koji boluju od određenih bolesti, pametni satovi koji konstantno prate otkucaje srca itd. Automatizacija prepoznavanja uzoraka kroz strojno učenje je bitna zbog same veličine podataka. Ručna analiza tih podataka je vrlo neučinkovita i neodrživa, te strojnim učenjem je moguće sažeti te velike skupove podataka u smislene obrasce. Postoji sve veći broj istraživačkih timova s ciljem prikupljanja i organiziranja skupova podatka za treniranje specijaliziranih strojnih modela [9]. Neki od trenutnih primjera uporabe su:

- Algoritmi strojnog učenja mogu analizirati medicinske slike za otkrivanje abnormalnosti poput tumora ili ozljeda glave [10].
- Moguće je analizirati genetske podatke za rano otkrivanje nasljednih bolesti [11].
- Analiziranje povijesti bolesti i životnog stila za predviđanje rizika od razvoja određenih bolesti [12].
- Pametni sustavi mogu pratiti vitalne znakove i upozoriti liječnike na pogoršanje njihovog stanja [13].
- Generativna umjetna inteligencija može pružati virtualne asistente liječnicima i pacijentima, koji odgovaraju na postavljena pitanja [14].

2.4.1 Primjeri programskih rješenja koji koriste strojno učenje i generativnu umjetnu inteligenciju

AlphaFold je sustav umjetne inteligencije dubokog učenja razvijen od strane Google DeepMind tima. Koristi strojno učenje kako bi predvidio strukturu proteina. Protein je poput niza kuglica napravljenih od niza aminokiselina. Ti se nizovi sastavljaju prema uputama DNK, koji je sastavni dio svakog organizma. Privlačenje i odbijanje aminokiselina uzrokuje da se dogodi presavijanje, time tvoreći kovrče i petlje 3D strukture proteina. To je važno kako bi se razumjele mnoge bolesti i kako bi se pomoglo razvoju novih lijekova. AlphaFold baza podataka sadrži preko 200 milijuna predviđanja izgleda već poznatih proteina [15]. Na slici 2.1 preuzetoj s [15] je kao primjer prikazan protein Q8I3H7, koji bi mogao štititi parazite protiv napada imunološkog sustava organizma.



Slika 2.1 Predviđanje 3D modela proteina Q8I3H7 [15]

IBM Watson je programsko rješenje kao usluga koji koristi umjetnu inteligenciju za pružanje personaliziranih preporuka za liječenje raka. Pruža analizu značenja strukturiranih i nestrukturiranih podataka u medicinskim nalazima. Kombinira atribute iz nalaza s literaturom Sloan Kettering centra za rak te vanjskim istraživanjima kako bi identificirao potencijalne planove i opcije za liječenje pacijenta. Osmišljen je za pružanje informacija onkolozima temeljene na relevantnom kliničkom znanju. Njegove najnovije mogućnosti uključuju proširenu pokrivenost na rak dojke, pluća, debelog crijeva, želuca i grlića maternice [16]. Na slici 2.2 je prikazan isječak funkcionalnosti koje obavlja IBM Watson.

Slika 2.2 Prikaz rada alata IBM Watson

3. MODEL I ARHITEKTURA WEB SUSTAVA

U ovom poglavlju opisani su funkcionalni i nefunkcionalni zahtjevi na sustav. Objašnjena je građa i komponente sustava te su također opisani korišteni algoritmi vezani za strojno učenje te pristup korištenja generativne umjetne inteligencije.

3.1 Funkcionalni zahtjevi na web sustav

Funkcionalni zahtjevi opisuju specifične zadatke koje sustav treba obavljati te definiraju što sustav treba omogućiti korisnicima tj. definiraju što sustav treba raditi. Web sustav ostvaruje sljedeće funkcionalne zahtjeve:

- Mogućnost registracije i prijave korisnika
- Pohrana identifikacijskog broja korisnika nakon prijave
- Nakon prijave dohvaćaju se svi postojeći nalazi prijavljenog korisnika
- Korisniku je omogućeno kreiranje novog nalaza ili izmjena postojećeg nalaza
- Omogućen je unos podataka koji se pohranjuje u nalazu
- Omogućeno je spremanje nalaza
- Moguće je na temelju unesenih podataka dobiti povratan odgovor s Azure i OpenAI usluga o stanju korisnika
- Korisniku je omogućena odjava iz sustava

Zahtjevi su ispunjeni kroz programski kod poslužiteljskog dijela sustava opisanog u poglavlju 4.

3.2 Nefunkcionalni zahtjevi na web sustav

Nefunkcionalni zahtjevi opisuju karakteristike sustava koje nisu povezane s njegovim funkcionalnostima, ali su ključne za njegovu učinkovitu upotrebu. Oni definiraju kako sustav treba raditi. Web sustav ostvaruje sljedeće nefunkcionalne zahtjeve:

- Sigurnost
- Robusnost
- Upotrebljivost
- Pouzdanost [17]

Sigurnost ima najvažniju ulogu kako bi se sustav osigurao od neovlaštenog pristupa osjetljivim podacima. Unutar sustava implementirana je metoda usporedbe unesene adrese e-pošte i lozinke sa već postojećim vrijednostima u bazi podataka. Robusnost predstavlja mogućnost sustava da nastavi s radom u neočekivanim uvjetima. To je ostvareno kroz osiguravanje ispravnog unosa

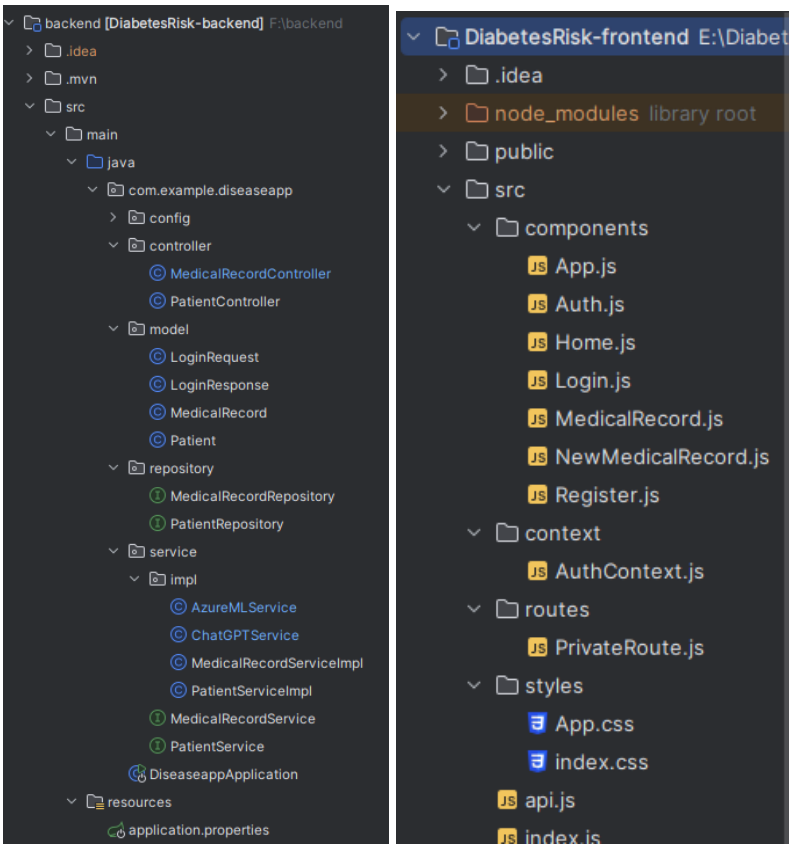
podataka vezanih uz prijavu pacijenta. Upotrebljivost se odnosi na intuitivnost sustava prilikom korištenja od strane korisnika. Ostvarena je putem jasno definiranih kontrola i tekstova koje se koriste unutar sustava. Pouzdanost je također ostvarena kako sustav ne bi neočekivano prestao s radom.

3.3 Građa i komponente web sustava

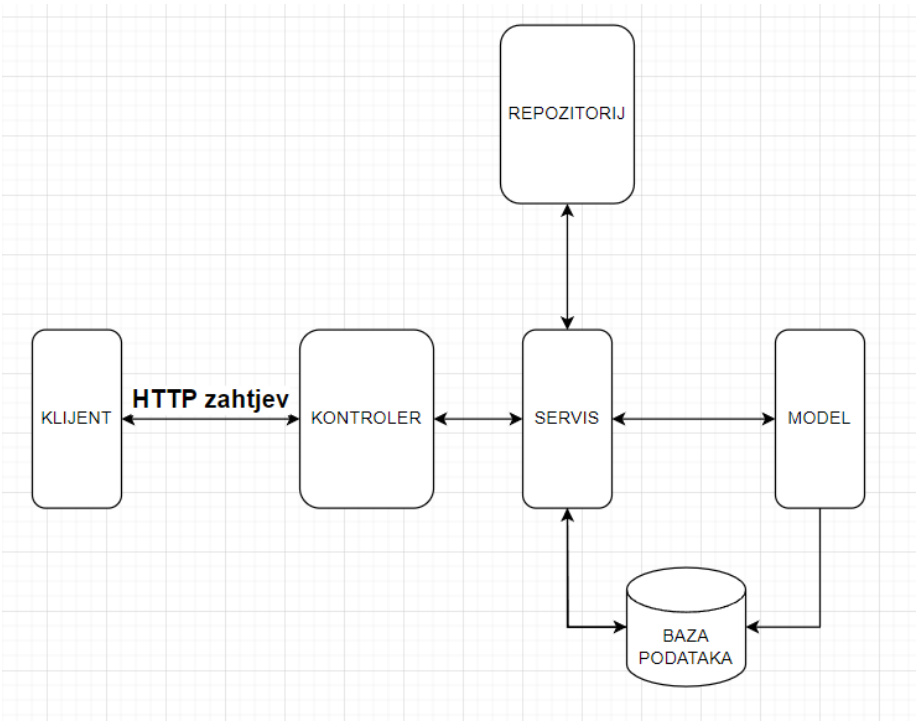
Web sustav je rađen na temelju MVC (engl. *Model View Controller*) arhitekture. Radi se o obrascu koji sustav dijeli na tri glavna dijela: *Model*, *View* i *Controller*. Model predstavlja poslovnu logiku sustava i upravlja podacima. U slučaju ovog rada, modeli su Java klase rađene unutar Spring Boota koje predstavljaju entitete u bazi podataka. Odgovoran je za pristup, ažuriranje i pravljenje operacija nad podacima. *View* predstavlja korisničko sučelje. To je dio koji korisnik vidi te se njime koristi. U ovom radu, *View* je rađen pomoću React biblioteke gdje su stranice sučelja podijeljene u komponente. *Controller* je dio sustava koji povezuje *Model* i *View*. On upravlja HTTP (engl. *Hypertext Transfer Protocol*) zahtjevima koje šalje korisnik, obrađuje ih te vraća odgovor korisniku. HTTP zahtjevi su glavna i najčešća metoda prijenosa informacija na internetu. *Controller* je također rađen unutar Spring Boota [18]. Slika 3.1 predstavlja strukturu projekta, gdje su prikazane klase koje predstavljaju funkcionalnosti vezane uz poslužiteljski i korisnički dio aplikacije.

Spring Boot arhitektura sastoji se od nekoliko slojeva koji su prikazani slikom 3.2. Funkcionira na način da kada klijent uputi HTTP zahtjev preko korisničkog sučelja sustava, zahtjev se proslijeđuje kontroleru na obradu . Nakon što kontroler primi zahtjev i obradi ga, poziva odgovarajuću metodu usluge i predaje mu odgovarajuće vrijednosti. Usluga obrađuje zahtjev pomoću modela i metoda unutar repozitorija. Nakon toga prosljeđuje odgovor nazad kontroleru, koji onda prosljeđuje odgovor nazad klijentu kroz korisničko sučenlje. Svi spomenuti dijelovi arhitekture su prikazani slikom 3.1. gdje su dijelovi prezentirani kao direktoriji unutar poslužiteljskog dijela sustava.

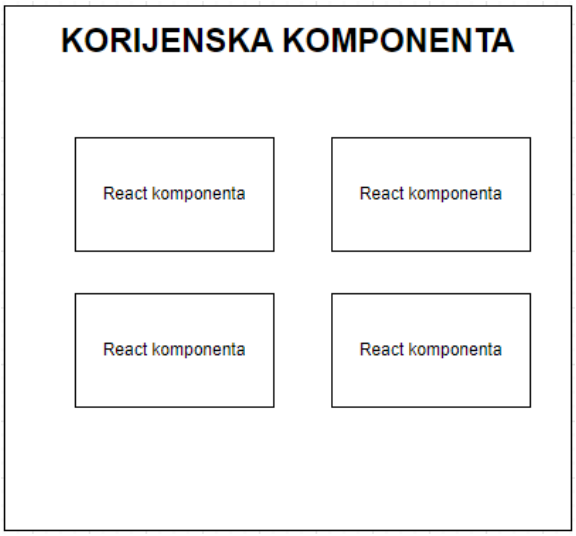
React arhitektura obično se temelji na komponentama. Radi se o pojedinačnim datotekama gdje svaka od komponenti sadrži dio korisničkog sučelja koji je moguće koristiti po potrebi više puta bilo gdje unutar sustava. Komponente se mogu ugrađivati jedna unutar druge, s time da korijenska komponenta sadrži sve ostale komponente unutar sebe. Ova hijerarhija je opisana slikom 3.3 .Komponente upravljaju svojim stanjem te su zadužene za postavljanje pojedinih dijelova korisničkog sučelja. Ovakav način arhitekture omogućuje lakše održavanje koda i brži pronalažak grešaka u kodu [19].



Slika 3.1 Struktura projekta vezana uz poslužiteljski i korisnički dio

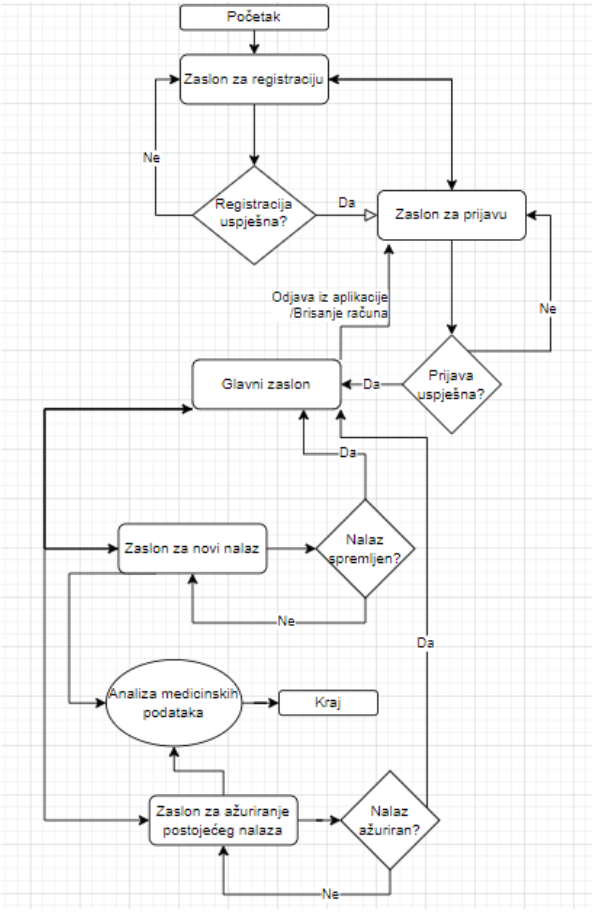


Slika 3.2 Prikaz arhitekture Spring Boota



Slika 3.3 Prikaz arhitekture React biblioteke

Na slici 3.4 prikazan je dijagram toka web sustava. Vidljivo je da se dijagram sastoji od komponenti koji predstavljaju zaslone sustava te od aktivnosti koji predstavljaju korisnikove zahtjeve nad sustavom. Slijed aktivnosti je detaljnije objašnjen u poglavlju 5 ovog rada.



Slika 3.4 Dijagram toka web sustava

3.4 Korišteni algoritmi strojnog učenja

Na temelju skupa podataka koji se koristi, korišten je binarni pristup gdje je cilj klasificirati pacijente u dvije kategorije: prisutnost ili odsutnost dijabetesa. Upravo zbog podjele na dvije kategorije, klasifikacijski algoritmi su najprikladniji za ovu vrstu analize. Radi se o algoritmima čiji je cilj predviđanje pripadnosti komponente jednoj od unaprijed definiranih kategorija na temelju njenih značajki. Model se trenira na skupu podataka koji već sadrži pridružene kategorije za pojedinu komponentu. Nakon obučavanja modela, moguće je klasificirati i dotad neviđene komponente u te iste kategorije. Kako bi se model trenirao, korištena je platforma Azure ML Studio opisana u poglavlju 4. Pomoću nje moguKako bi se utvrdio algoritam s najboljim rezultatima, skup je treniran na četiri različita klasifikacijska algoritma koja su dostupna preko Azure platforme: stablo odluke, slučajna šuma, logistička regresija, neuronska mreža [20]. Metrike koje su korištene za vrednovanje točnosti modela su opisane u poglavlju 4.

3.4.1 Stablo odluke

Stablo odluke je algoritam strojnog učenja koji se koristi za klasifikaciju i regresiju. Započinje s cijelim skupom podataka i postepeno ga dijeli na manje podskupove na temelju značajki podataka. Svaki čvor u stablu predstavlja odluku ili test na nekoj značajki, a grane predstavljaju ishod testa [20].

3.4.2 Slučajna šuma

Slučajne šume su kombinacija većeg broja stabala odluke s ciljem postizanja što točnijih klasifikacijskih rezultata. Stabla odluke se generiraju slučajno, a nakon što svako stablo izračuna svoju izlaznu vrijednost, algoritam slučajne šume kombinira te rezultate i odabire najbolji rezultat. Time se smanjuje rizik od prenaučenosti i poboljšava stabilnost modela [20].

3.4.3 Logistička regresija

Logistička regresija je statistički model koji se koristi za predviđanje vjerojatnosti pripadnosti razredima. Koristi se za klasifikacijske probleme s dvije ili više klasa. Logistička regresija koristi logističku funkciju, poznatu kao sigmoidna funkcija, za mapiranje predikcija i njihovih vjerojatnosti. Sigmoidna funkcija je S-krivulja koja bilo koju realnu vrijednost pretvara u raspon između 0 i 1. Funkcija je prikazana izrazom 4-1 [20].

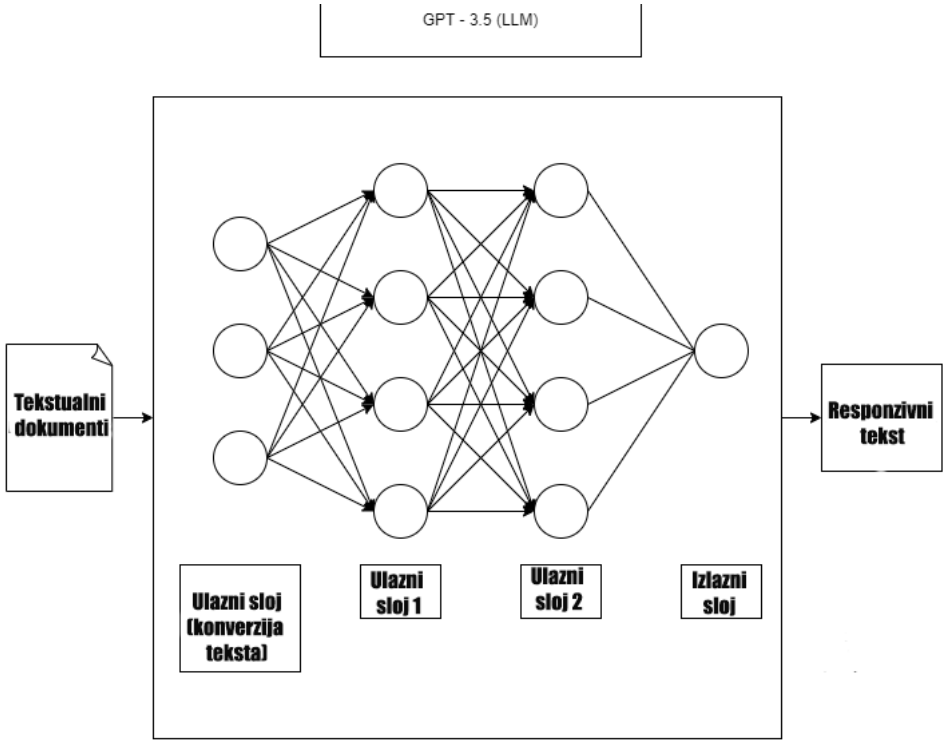
$$f(x) = \frac{1}{1+e^{-x}} \tag{4-1}$$

3.4.4 Neuronska mreža

Neuronska mreža je skup povezanih neurona tj. jedinica organiziranih u slojeve. Svaki neuron obrađuje ulazne podatke te prosljeđuje rezultat sljedećem sloju. Koriste se za razne zadatke, uključujući klasifikaciju, regresiju, prepoznavanje uzoraka, obradu prirodnog jezika i još mnogo toga [20].

3.5 Korišteni pristupi generativne umjetne inteligencije

Za potrebe ovog rada korištena je ChatGPT usluga, čiji modeli spadaju pod LLM (eng. *Large Language Model*). LLM modeli su vrsta jezičnih modela koji se treniraju na ogromnim količinama podataka. Ti podatci su uobičajeno prikupljeni s interneta te mogu sadržavati milijune gigabajta teksta za obradu. Cilj modela je da s dovoljno primjera uspijeva razumjeti i interpretirati ljudski jezik, i da daje tekstualne odgovore. GPT modeli su izgrađeni na transformer arhitekturi, koji se sastoji od enkodera i dekodera. GPT koristi samo dekodere kako bi generirao tekstualne odgovore. Rad ChatGPT-a uključuje nekoliko koraka. Prvi korak je da korisnik unese upit ili upit u sustav, koji se zatim obrađuje u modelu. Model koristi svoje razumijevanje jezičnih obrazaca i asocijacija za generiranje odgovora, koji se zatim vraća korisniku. Korisnik može dalje nastaviti razgovor ili postaviti dodatna pitanja. Ovaj pristup je u potpunosti uvježban kroz podržano učenje, pri čemu je ljudska povratna informacija glavni izvor poboljšanja [21]. U ovom radu korišten je model *GPT 3.5 turbo*. Njegov način rada prikazan je na slici 3.5.



Slika 3.5 Način rada modela GPT 3.5 [22]

Prema slici 3.5, tekstualni dokument predstavlja početnu točku procesa. To je izvorni tekst koji se želi obraditi ili analizirati. Ulazni sloj odgovoran je za pretvorbu tekstualnog dokumenta u format koji je razumljiv modelu. U slučaju GPT-3.5, ovo uključuje tokenizaciju, gdje se tekst dijeli na manje jedinice poput riječi ili dijelova riječi. Ostali numerirani ulazni slojevi predstavljaju dodatnu obradu i transformaciju podataka. Izlazni sloj je sloj za generiranje odgovora na temelju obrade ulaznih podataka, dok je responzivni tekst rezultat obrade koji se šalje korisniku.

4. PROGRAMSKO RJEŠENJE WEB SUSTAVA

U ovom poglavlju objašnjene su sve programske tehnologije, jezici i razvojne okoline koje su korištene za razvoj programskog rješenja. Također je objašnjen način implementacije modela strojnog učenja i generativne umjetne inteligencije u web sustav. Na kraju je prikazan programski kod poslužiteljskog te korisničkog dijela sustava. Sve funkcionalnosti sustava su pojedinačno opisane.

4.1 Korištene programske tehnologije, jezici i razvojne okoline

4.1.1 Spring Boot

Spring Boot je programski okvir otvorenog koda kojeg je kreirao Pivotal Team Inc. Kreiran je kao odgovor na nekoliko problema s kojima su se programeri suočavali koristeći originalni Spring Framework te se može smatrati njegovim proširenjem. Spring Boot omogućuje smanjenje takozvanog „boilerplate“ koda. To je kod koji se često ponavlja unutar aplikacije, ali se ne mijenja. Spring Boot koristi starter zavisnosti sa unaprijed definiranim konfiguracijama kako bi se pojednostavila izgradnja aplikacija. Time se omogućuje izbjegavanje navođenja svake zavisnosti pojedinačno. Spring Boot se automatski brine o konfiguriranju što omogućuje programerima da se samo brinu o poslovnoj logici aplikacije [23].

4.1.2 PostgreSQL

PostgreSQL je sustav baze podataka koji podržava SQL standarde. Od rujna 2023. godine, Postgre je u skladu sa 170 od 179 obaveznih značajki SQL:2023 standardom. PostgreSQL je stekao snažnu reputaciju zahvaljujući svojoj provjerenoj arhitekturi, pouzdanosti, integritetu podataka te proširivosti. Dolazi s mnogim značajkama usmjerenim na pomoć programerima u izradi aplikacija te omogućuje upravljanje podacima bez obzira na njihovu veličinu. Podržan je širok raspon izvornih tipova podataka poput: *boolean*, binarni, znakovni, datum/vrijeme itd. Također je moguće kreirati vlastite tipove podataka koji se obično mogu u potpunosti indeksirati putem PostgreSQL-ove infrastrukture za indeksiranje [24].

4.1.3 Maven

Maven je alat za upravljanje projektima i alat za razumijevanje projekata, koji se koristi prvenstveno za Java projekte. Omogućuje automatizaciju procesa izgradnje, upravljanje zavisnostima i generiranje dokumentacije projekta. Mavenova primarna svrha je omogućiti programeru da u najkraćem mogućem roku u potpunosti shvati stanje projekta. Pojednostavljuje proces izgradnje projekta definiranjem jasnih standarda i konvencija za upravljanje izgradnjom. Potiče stvaranje detaljnih metapodataka o projektu, kao što su verzije, ovisnosti i struktura

projekta, kako bi korisnicima pružio jasnoću i razumijevanje projekta. Promiče korištenje najboljih praksi u razvoju programskih rješenja, uključujući organizaciju projektnih datoteka, upravljanje ovisnostima, testiranje i dokumentiranje [25].

4.1.4 ChatGPT

ChatGPT je jezični model razvijen od strane OpenAI-a koji koristi umjetnu inteligenciju za razgovor s korisnicima putem teksta, pružajući odgovore i pomoć u realnom vremenu na različite teme. Smatra se početkom ubrzanog rasta korištenja umjetne inteligencije u aplikacijama. Do siječnja 2023., postala je najbrže rastuća aplikacija za potrošače u povijesti, stekavši više od 100 milijuna korisnika. GPT je skraćenica za "Generative Pre-trained Transformer". To je vrsta modela umjetne inteligencije koja koristi transformere za generiranje teksta na temelju prethodno naučenih uzoraka iz velikih količina podataka. Model se prvo trenira na velikim skupovima podataka, a zatim se može podesiti za specifične zadatke, kao što su odgovaranje na pitanja ili vođenje razgovora. ChatGPT je baziran na osnovnim modelima GPT-a, posebno GPT-3.5, GPT-4 i GPT-4o, koji su podešeni za razgovornu upotrebu. Proces podešavanja koristi nadzirano učenje i učenje putem povratnih informacija od korisnika. Za potrebe rada korišten je API kojemu je proslijeđen upit, te on daje povratnu informaciju korisniku o njegovom riziku oboljenja od dijabetesa [26].

4.1.5 Azure

Azure je platforma računarstva u oblaku i usluga od Microsofta, koja nudi širok spektar usluga uključujući hosting aplikacija, skladištenje podataka, analitiku i mnoge druge. Microsoft Azure podržava brojne programske jezike, alate i okvire, uključujući programska rješenja i sustave specifične za Microsoft, kao i rješenja trećih strana. Za potrebe rada korišten je Azure ML Studio, integrirano razvojno okruženje koje omogućuje korisnicima dizajniranje, izgradnju, testiranje i implementaciju modela strojnog učenja na Microsoft Azure platformi. Kreirani model pozivan je pomoću API-ja kako bi dao povratnu informaciju korisniku o njegovom riziku oboljenja od dijabetesa [27].

4.1.6 React

React je besplatna JavaScript biblioteka koja omogućuje programerima izradu brzih i pouzdanih aplikacija. Nudi širok spektar alata, API-ja i biblioteka za pojednostavljenje i optimizaciju razvojnog procesa. React pruža čvrstu platformu koja omogućuje izradu aplikacija koje su brze, pouzdane i skalabilne, bez obzira na veličinu tima ili koda. U ovom radu React je korišten za slanje zahtjeva prema poslužitelju te kao grafičko sučelje za korisnika [28].

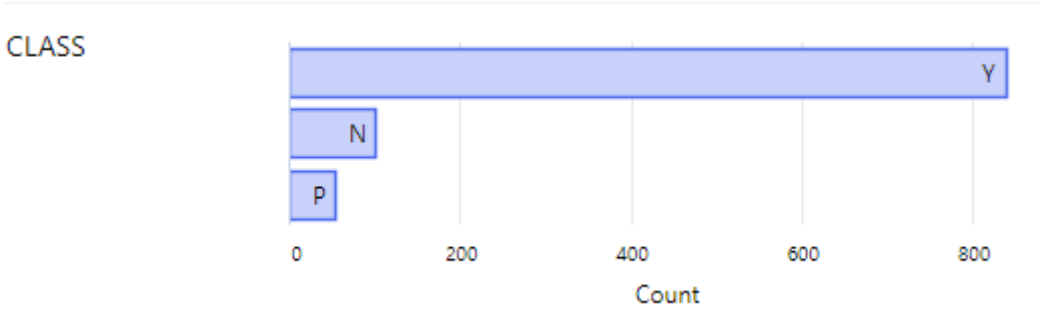
4.1.7 IntelliJ IDEA

IntelliJ IDEA je jedan od najpopularnijih integriranih razvojnih okruženja namijenjenih programiranju u različitim jezicima, posebno za Java aplikacije. Razvila ga je kompanija JetBrains i nudi širok spektar funkcionalnosti koje olakšavaju razvoj programskih rješenja. Podržava različite tehnologije i programske okvire kao što su Spring, Hibernate, Maven, Gradle, Android, Git, i druge. U radu je korišten kako bi se ostvarilo programsko rješenje na strani poslužitelja [29].

4.2 Razvoj i implementacija modela strojnog učenja

4.2.1 Skup podataka za treniranje modela

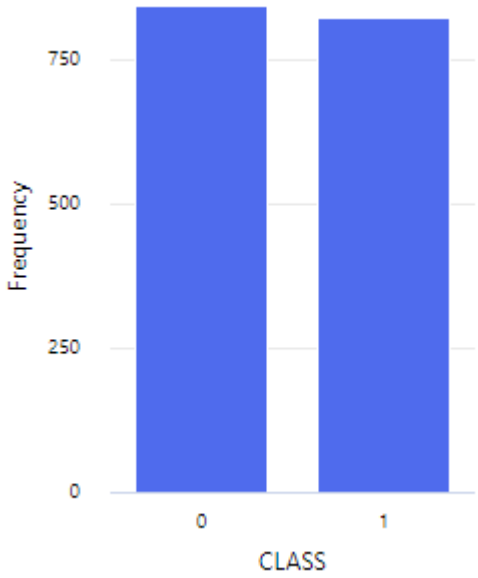
Skup podataka *Diabetes Dataset* je preuzet sa stranice Kaggle [30]. Odabrani skup se sastojao od 14 stupaca: ID, No_Pation, Gender, AGE, Urea, Cr, HbA1c, Chol, TG, HDL, LDL, VLDL, BMI, CLASS. Izbačeni su stupci ID i No_Pation zato što ne predstavljaju rizične čimbenike opisane u poglavlju 2. tj. ne doprinose analizi podataka kako bi se odredilo postojanje dijabetesa kod pacijenta. Stupac CLASS se sastojao od tri vrijednosti prikazanih na slici 4.1: osoba ima dijabetes (vrijednost Y), osoba nema dijabetes (vrijednost N) te preddijabetes (P). Daljnjom analizom skupa utvrđeno je da su podaci vrlo neuravnoteženi: Y vrijednosti je 840, N 102 te P 53.



Slika 4.1 Vrijednosti stupca CLASS

Iz tog razloga odlučeno je primijeniti SMOTE tehniku. Radi se o tehnici naduzorkovanja koja se koristi za rješavanje problema neuravnoteženih podataka u strojnom učenju. Identificira se razred koji ima manje uzoraka, te SMOTE generira nove sintetičke uzorke. To se postiže odabirom uzorka iz manjinskog razreda i pronalaženjem njegovih k-najbližih susjeda [31]. Ova metoda omogućava stvaranje novih uzoraka koji su slični, ali ne identični postojećim uzorcima. Rezultat je obogaćivanje skupa podataka s novim varijantama koji pomažu u boljem učenju modela. Generiranjem sintetičkih uzoraka, SMOTE pomaže u postizanju ravnoteže između razreda, što omogućava modelima strojnog učenja da bolje nauče karakteristike manjinskog razreda. Ovo može poboljšati performanse modela u prepoznavanju i klasificiranju uzoraka iz manjinskog

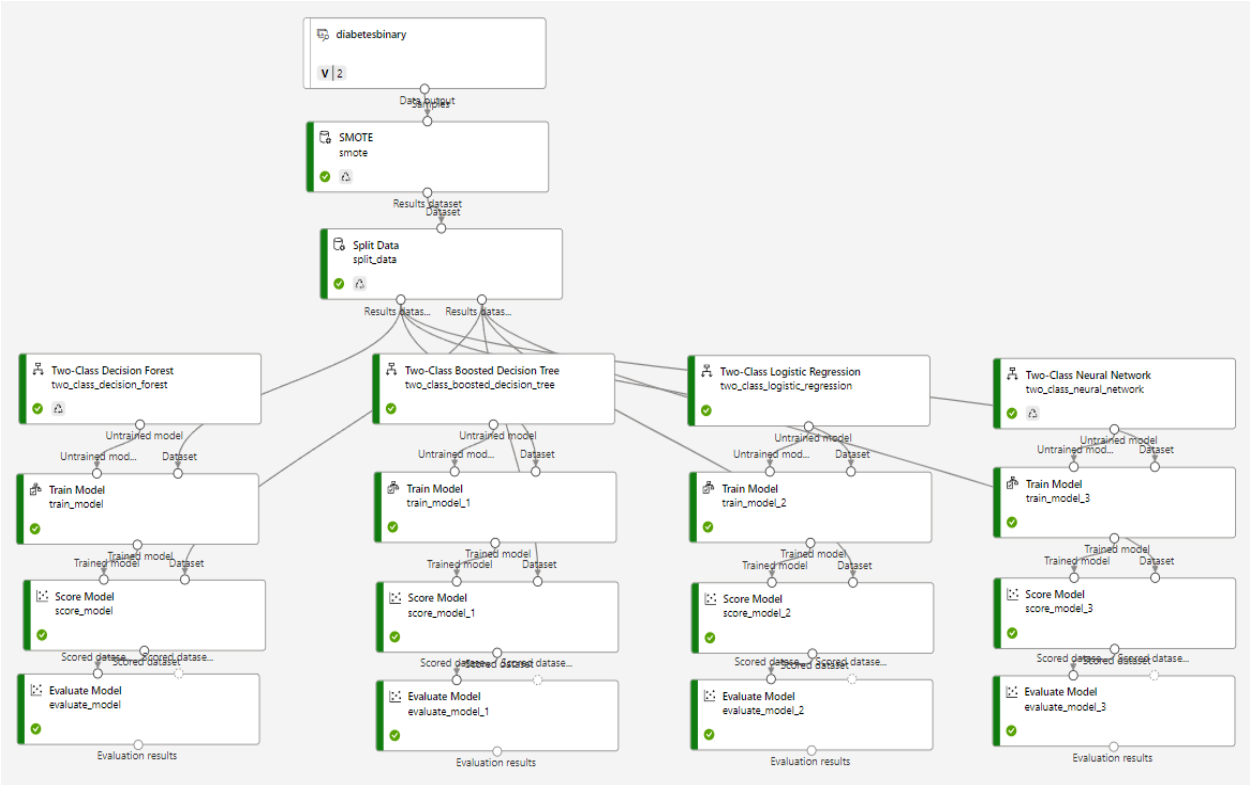
razreda, čime se smanjuje pristranost prema većinskom razredu. Zbog toga što SMOTE može raditi samo u binarnoj klasifikaciji, odlučeno je izbaciti skup podataka P jer ima najmanje podataka. Rezultat tehnike je prikazan slikom 4.2. Skup podataka je nakon toga razdvojen u omjeru 75:25, gdje je 75 % skupa korišteno za treniranje modela, a 25 % za njegovo testiranje.



Slika 4.2 Vrijednosti stupca CLASS primjenom tehnike SMOTE

4.2.2 Odabir klasifikacijskog algoritma

Kako bi se odredio najučinkovitiji algoritam, kreiran je model strojnog učenja prikazan slikom 4.3. Skup podataka je prvotno proveden kroz SMOTE, te nakon toga razdijeljen u omjeru 75:25. Svaki od algoritama se odvojeno koristio za treniranje svojeg modela, te je nakon toga ocijenjena njihova učinkovitost.



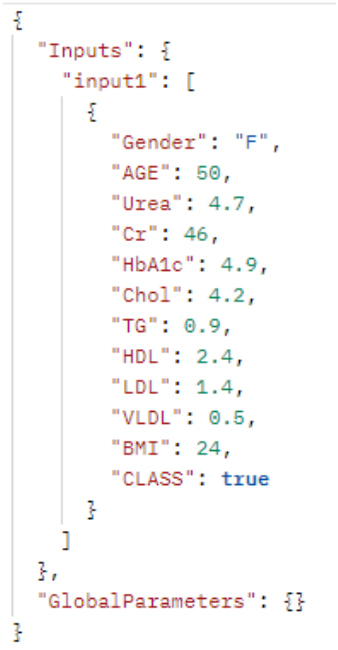
Slika 4.3 Model za testiranje najboljeg algoritma klasifikacije

Ocjene su se temeljile na metrikama prikazanim slikama 4.4, 4.5, 4.6 te 4.7. Točnost (engl. *Accuracy*) mjeri postotak točnih predviđanja u usporedbi sa svim predviđanjima koje model napravi. AUC mjeri sposobnost modela da razlikuje između klasa. F1 se koristi kako bi se uravnotežile vrijednosti preciznosti i odziva. Preciznost (engl. *Precision*) mjeri mogućnost modela da izbjegne predviđanje negativnih primjera kao pozitivnih, a odziv (engl. *Recall*) mjeri proporciju točno predviđenih pozitivnih primjera sa svim stvarnim pozitivnim primjerima [32]. Usporedbom svih rezultata prikazanih slikom 4.4 pokazalo se da je stablo odluke ostvarilo najbolje vrijednosti te je on odabran kao algoritam klasifikacije koji je korišten za model strojnog učenja.

Slika 4.4 Metrike modela klasifikacijskih algoritama

4.2.3 Postavljanje web usluge

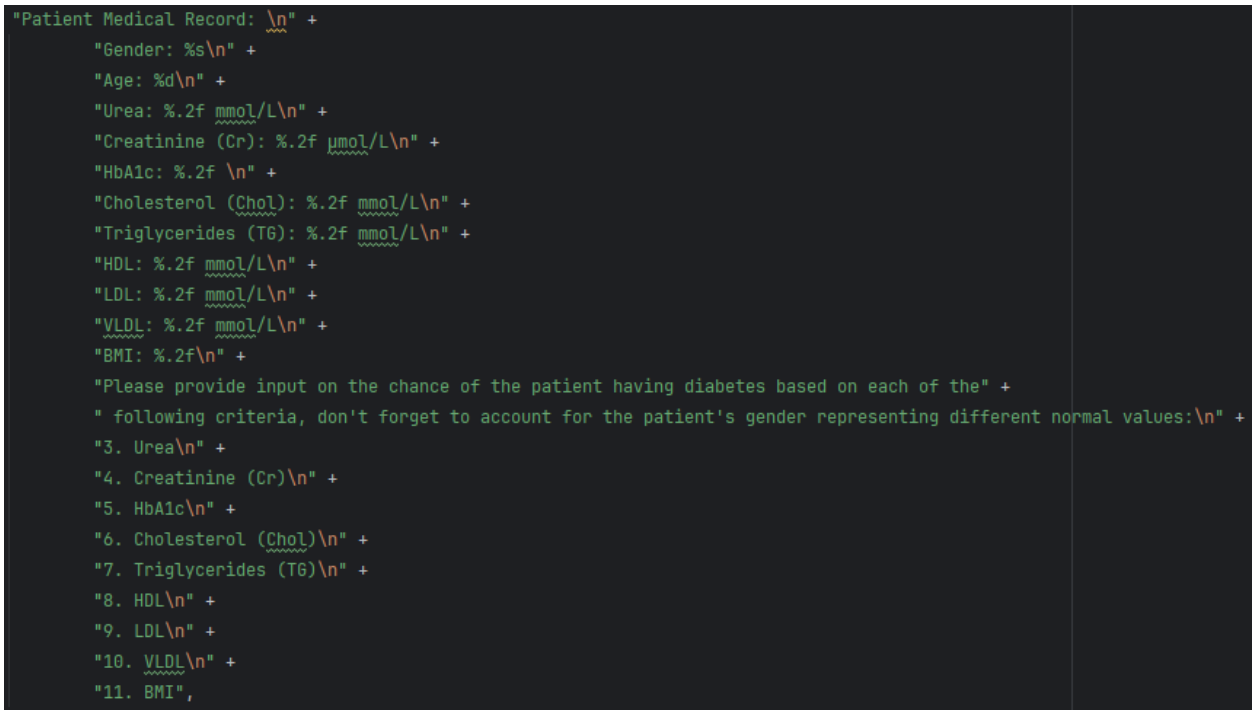
Nakon što je odabran algoritam za model strojnog učenja, kreira se cjevovod. To je niz koraka koji se koriste za primjenu modela na nove podatke kako bi mogao generirati predviđanja na temelju neviđenih podataka. Nakon što se kreira i postavi cjevovod, slijedi postavljanje modela kao web usluge. Nakon uspješnog postavljanja usluge, sada je moguće slati zahtjeve modelu preko kreiranog URL-a u formatu JSON prikazanom na slici 4.5. Unutar tog JSON-a šalju se vrijednosti rizičnih čimbenika koji su objašnjeni u poglavlju 2. Iako usluga traži da se pošalje i atribut CLASS koji je predviđen kada usluga vrati odgovor nazad, on nema nikakvog utjecaja na predviđanje modela, te je moguće poslati proizvoljnu vrijednost za njegov atribut.



Slika 4.5 Zahtjev u formatu JSON za slanje podataka na uslugu Azure

4.3 Implementacija generativne umjetne inteligencije u web sustav

Kako bi se implementirala generativna umjetna inteligencija u web sustav. Korišten je OpenAI API [33]. On pruža jednostavan način za pristup modelima koji se mogu koristiti u razne svrhe, a koji je unutar ovog rada korišten za analizu unesenih podataka o pacijentu. Kako bi se API mogao koristiti, potrebno je kreirati API ključ koji se koristi kako bi se sigurno pristupalo API-ju. Nakon toga se unutar sustava postavljaju podatci o adresi na kojoj se nalazi API te i sam API ključ koji je kasnije korišten za slanje poruka API-ju. OpenAI modeli stvaraju tekst na temelju primljene poruke od korisnika i šalju ga nazad korisniku. Kvaliteta odgovara je proporcionalna kvaliteti poruke. Korisnik koji šalje poruku treba jasno definirati što traži od modela. U suprotnom, ako je poruka nejasna, može dovesti do nepotpunih ili generičkih odgovora. U svrhu poboljšanja kvalitete poruke kreirana je nova disciplina koja se naziva promptni inženjering. Obuhvaća širok raspon vještina i tehnika koje su korisne za interakciju i razvoj modela generativne umjetne inteligencije [34]. Zahtjev koji se šalje modelu traži od modela da analizira svaku pojedinu značajku unesenu od strane korisnika te vrati u kakvom su stanju te značajke u odnosu na normalne vrijednosti. Također i na kraju treba vratiti je li korisnik pod rizikom bolovanja od dijabetesa ili ne. U skladu s navedenim, odabrana je poruka koja je prikazana na slici 4.6. U njoj je navedeno da se radi o nalazu pacijenta, navedena je svaka značajka zajedno sa svakom od unesenih vrijednosti, te se dodatno naglašava uporaba spola kako bi se prilagodile normalne vrijednosti čimbenika koje model odabire za analizu, zbog različitosti prosječnih vrijednosti između muškaraca i žena.



Slika 4.6 Tekst poruke koja se šalje OpenAI API-ju

4.4 Programsko rješenje na strani poslužitelja

Programsko rješenje je kreirano pomoću Spring Boot programskog okvira. Funkcionalnosti sustava na strani poslužitelja su raspoređene prema tome odnose li se na model *Patient* ili model *MedicalRecord*.

4.4.1 Funkcionalnosti vezane uz model *Patient*

U ovom dijelu surađuju tri klase: *PatientController*, *PatientServiceImpl* te *PatientRepository*. Metoda *savePatient* sprema objekt *Patient* u bazu podataka. Metoda *login* pomoću metode *authenticate* provjerava da li korisnik koji se prijavljuje već postoji u bazi podataka, tako da usporedi predanu adresu e-pošte i lozinku. Ako je provjera uspješna, vraćen je objekt *Patient*, a ako nije, vraćen je *null*, te na temelju toga je poslan odgovarajući odgovor nazad. Opisani programski kod prikazan je na slikama 4.7 te 4.8.

```java
@RestController  Antonio Šušovček
@RequestMapping("/api/patient")
public class PatientController {

    @Autowired
    private PatientService patientService;

    @PostMapping("/save")  Antonio Šušovček
    public String savePatient(@RequestBody Patient patient) {
        patientService.savePatient(patient);
        return "Patient saved successfully";
    }

    @PostMapping("/login")  Antonio Šušovček
    public ResponseEntity<?> login(@RequestBody LoginRequest loginRequest) {
        Patient patient = patientService.authenticate(loginRequest.getEmail(), loginRequest.getPassword());

        if (patient != null) {
            return ResponseEntity.ok(new LoginResponse(patient.getId(), message: "Login successful"));
        } else {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid email or password");
        }
    }
}
```

Slika 4.7 Klasa *PatientController*

```java
@Service  Antonio Šušovček
public class PatientServiceImpl implements PatientService {

    @Autowired
    private PatientRepository patientRepository;

    @Override  1 usage  Antonio Šušovček
    public Patient savePatient(Patient patient) { return patientRepository.save(patient); }

    @Override  4 usages  Antonio Šušovček
    public Patient findById(Long id) {
        Optional<Patient> patient = patientRepository.findById(id);
        return patient.orElse( other: null);
    }

    @Override  1 usage  Antonio Šušovček
    public Patient authenticate(String email, String password) {
        return patientRepository.findByEmail(email)
                .filter(patient -> patient.getPassword().equals(password))
                .orElse( other: null);
    }
}
```

Slika 4.8 Klasa *PatientService*

4.4.2 Funkcionalnosti vezane uz model *MedicalRecord*

U ovom dijelu koda surađuje pet različitih klasa: *MedicalRecordController*, *MedicalRecordServiceImpl*, *MedicalRecordRepository*, *ChatGPTService* te *AzureService*. Funkcionalnosti su objašnjene pomoću *MedicalRecordController* klase.

4.4.2.1 Metoda *saveMedicalRecord*

Ova metoda omogućuje spremanje nalaza u bazu podataka. To se ostvaruje pomoću klase *MedicalRecordRepository* koja proširuje *JPARepository*. *JPARepository* je sučelje koje pruža skup metoda za rad s entitetima u bazi podataka. Programski kod metode *saveMedicalRecord* i klase *MedicalRecordRepository* je prikazan slikama 4.9, 4.10. te 4.11.

```
@PostMapping("/{id}/medicalrecord/save")  Antonio Šušovček *
public ResponseEntity<String> saveMedicalRecord(@PathVariable Long id, @RequestBody MedicalRecord medicalRecord) {
    Patient patient = patientService.findById(id);
    if (patient == null) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Patient not found");
    }


    medicalRecord.setPatient(patient);
    medicalRecordService.saveMedicalRecord(medicalRecord);

    return ResponseEntity.status(HttpStatus.CREATED).body("Medical record saved successfully");
}
```

Slika 4.9 Metoda *saveMedicalRecord* unutar klase *MedicalRecordController*

```
@Override  1 usage  Antonio Šušovček
public MedicalRecord saveMedicalRecord(MedicalRecord medicalRecord) {
    return medicalRecordRepository.save(medicalRecord);
}
```

Slika 4.10 Metoda *saveMedicalRecord* unutar klase *MedicalRecordServiceImpl*

```
@Repository  2 usages  Antonio Šušovček
public interface MedicalRecordRepository extends JpaRepository<MedicalRecord, Long> {
    List<MedicalRecord> findByPatientId(Long patientId);  1 usage  Antonio Šušovček
}
```

Slika 4.11 Klasa *MedicalRecordRepository*

4.4.2.2 Metoda *getMedicalRecordById*

Metoda *getMedicalRecordById* omogućuje pronalazak postojećeg nalaza unutar baze podataka pomoću predanog ID-a. Metoda *findById* se koristi kako bi pronašao ID nalaza koji ima istu vrijednost kao i predani ID. To se ostvaruje pomoću klase *MedicalRecordRepository* koja proširuje *JPARepository*. Programski kod metode *getMedicalRecordById* je prikazan slikom 4.12.

```
@GetMapping("/medical-record/{recordId}")  Antonio Šušovček *
public ResponseEntity<MedicalRecord> getMedicalRecordById(@PathVariable Long recordId) {
    MedicalRecord medicalRecord = medicalRecordService.findById(recordId);
    if (medicalRecord == null) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(null);
    }
    return ResponseEntity.ok(medicalRecord);
}
```

Slika 4.12 Metoda *getMedicalRecordById*

4.4.2.3 Metoda *getMedicalRecordsByPatientId*

Ova metoda omogućuje dohvaćanje svih nalaza pacijenta pomoću predanog ID-a. Predani ID se koristi kako bi se pacijent potražio u bazi podataka. Ako on postoji, dohvaćaju se svi nalazi vezani uz tog pacijenta i šalju nazad kao odgovor. Programski kod metode *getMedicalRecordByPatientID* je prikazan slikom 4.13.

```
@GetMapping("/{id}/medicalrecord")  Antonio Šušovček *
public ResponseEntity<List<MedicalRecord>> getMedicalRecordsByPatientId(@PathVariable Long id) {
    Patient patient = patientService.findById(id);
    if (patient == null) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(null);
    }

    List<MedicalRecord> medicalRecords = medicalRecordService.getMedicalRecordsByPatientId(id);
    if (medicalRecords.isEmpty()) {
        return ResponseEntity.noContent().build();
    }

    return ResponseEntity.ok(medicalRecords);
}
```

Slika 4.13 Metoda *getMedicalRecordsByPatientID*

4.4.2.4 Metoda *getChatGPTResponse*

Ova metoda se koristi kako bi se atributi nalaza poslali na OpenAI API i dobio povratan odgovor koji sadrži analizu predanih atributa. Nalaz se predaje na obradu metodi unutar *ChatGPTService* klase. Programski kod metode je prikazan slikom 4.14.

```
@PostMapping("/{id}/medicalrecord/gpt")  Antonio Šušovček *
public String getChatGPTResponse(@PathVariable Long id, @RequestBody MedicalRecord medicalRecord) {
    Patient patient = patientService.findById(id);
    if (patient == null) {
        return "Patient not found";
    }
    medicalRecord.setPatient(patient);

    return chatGPTService.getChatGPTResponse(medicalRecord);
}
```

Slika 4.14 Metoda *getChatGPTResponse* unutar klase *MedicalRecordController*

Metoda *getChatGPTResponse* prvo postavlja HTTP zaglavlje u JSON format te postavlja autentifikaciju pomoću OpenAI API ključa. Zatim se poziva metoda koja postavlja poruku koja se šalje na API pomoću predanih podataka iz nalaza. U *requestBody* se postavlja vrsta modela koja je korištena, njegova uloga te prethodno spomenuta poruka. On se šalje pomoću *REST* obrasca te se rezultat vraća kao JSON *string*. Nakon toga se poziva metoda *extractContentFromResponse* kako bi se iz JSON-a izvukao *string* koji sadrži odgovor modela. Programski kod metoda *getChatGPTResponse* te *extractContentFromResponse* je prikazan slikama 4.15 i 4.16.

4.4.2.1 Metoda *getAzureResponse*

Ova metoda se koristi kako bi se atributi nalaza poslali na Azure API i dobio povratan odgovor koji vraća vrijednost *boolean* koja može biti *true* ili *false*. Nalaz se predaje na obradu metodi unutar klase *AzureMLService*. Programski kod metode *getAzureMLResponse* je prikazan slikom 4.17.

Unutar metode *predict* postavlja se HTTP zaglavlje u JSON format te se postavlja autentifikacija pomoću Azure API ključa. Programski kod metode *predict* je prikazan slikom 4.18.

Metoda *constructRequestBody* gradi tijelo zahtjeva za slanje podataka o nalazu na Azure API. Prvo se stvara mapa *inputs* koja sadrži podatke iz nalaza, te se zatim ti podaci dodaju u mapu *input1* koja se s globalnim parametrima pohranjuje u mapu *requestBody*. *RequestBody* se na kraju pretvara u JSON *string*. Programski kod metode *constructRequestBody* je prikazan slikom 4.19.

```java
public String getChatGPTResponse(MedicalRecord medicalRecord) {  1 usage  Antonio Šušovček *
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    headers.setBearerAuth(apiKey);

    String prompt = constructPromptFromMedicalRecord(medicalRecord);

    Map<String, Object> requestBody = new HashMap<>();
    requestBody.put("model", "gpt-3.5-turbo");
    requestBody.put("messages", new Object[]{new HashMap<String, String>() {{  Antonio Šušovček
        put("role", "user");
        put("content", prompt);
    }}});
    requestBody.put("max_tokens", 850);

    HttpEntity<Map<String, Object>> request = new HttpEntity<>(requestBody, headers);

    ResponseEntity<String> response = restTemplate.postForEntity(apiUrl, request, String.class);

    return extractContentFromResponse(response.getBody());
}
```

Slika 4.15 Metoda *getChatGPTResponse* unutar klase *ChatGPTService*

```java
private String extractContentFromResponse(String responseBody) {  1 usage  Antonio Šušovček
    try {
        JsonNode root = objectMapper.readTree(responseBody);
        return root.path( s: "choices").get(0).path( s: "message").path( s: "content").asText();
    } catch (Exception e) {
        throw new RuntimeException("Failed to extract content from response", e);
    }
}
```

Slika 4.16 Metoda *extractContentFromResponse* unutar klase *ChatGPTService*

```java
@PostMapping("/{id}/medicalrecord/azure")  Antonio Šušovček *
public String getAzureMLResponse(@PathVariable Long id, @RequestBody MedicalRecord medicalRecord) {
    Patient patient = patientService.findById(id);
    if (patient == null) {
        return "Patient not found";
    }
    medicalRecord.setPatient(patient);

    return azureMLService.predict(medicalRecord);
}
```

Slika 4.17 Metoda *getAzureMLResponse* unutar klase *MedicalRecordController*

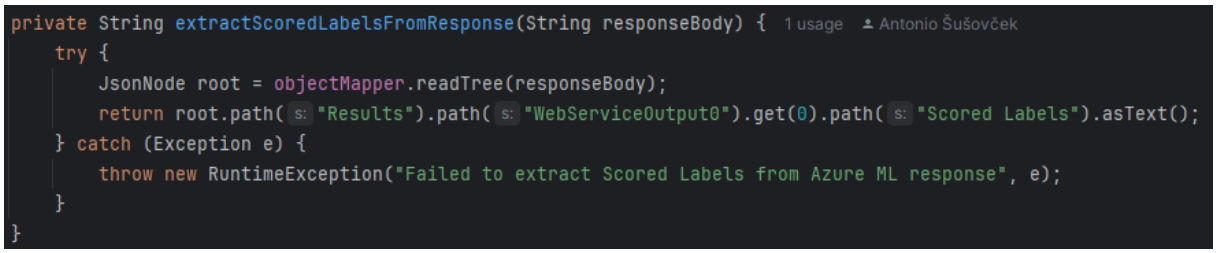
```
public String predict(MedicalRecord medicalRecord) {  1 usage   Antonio Šušovček *
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    headers.set("Authorization", "Bearer " + azureMlApiKey);

    String requestBody = constructRequestBody(medicalRecord);

    HttpEntity<String> entity = new HttpEntity<>(requestBody, headers);

    RestTemplate restTemplate = new RestTemplate();
    ResponseEntity<String> response = restTemplate.exchange(azureMlUrl, HttpMethod.POST, entity, String.class);

    String scoredLabel = extractScoredLabelsFromResponse(response.getBody());

    return generateEvaluationMessage(medicalRecord, scoredLabel);
}
```

Slika 4.18 Metoda *predict* unutar klase *AzureMLService*

```
private String constructRequestBody(MedicalRecord medicalRecord) {  1 usage   Antonio Šušovček
    Map<String, Object> inputs = new HashMap<>();
    inputs.put("Gender", medicalRecord.getGender());
    inputs.put("AGE", medicalRecord.getAge());
    inputs.put("Urea", medicalRecord.getUrea());
    inputs.put("Cr", medicalRecord.getCr());
    inputs.put("HbA1c", medicalRecord.getHba1c());
    inputs.put("Chol", medicalRecord.getChol());
    inputs.put("TG", medicalRecord.getTg());
    inputs.put("HDL", medicalRecord.getHdl());
    inputs.put("LDL", medicalRecord.getLdl());
    inputs.put("VLDL", medicalRecord.getVldl());
    inputs.put("BMI", medicalRecord.getBmi());
    inputs.put("CLASS", false);

    Map<String, Object> input1 = new HashMap<>();
    input1.put("input1", List.of(inputs));

    Map<String, Object> requestBody = new HashMap<>();
    requestBody.put("Inputs", input1);
    requestBody.put("GlobalParameters", new HashMap<>());

    try {
        return objectMapper.writeValueAsString(requestBody);
    } catch (Exception e) {
        throw new RuntimeException("Failed to construct Azure ML request body", e);
    }
}
```

Slika 4.19 Metoda *constructRequestBody* unutar klase *AzureMLService*

Nakon izvršenja metode *constructRequestBody*, metoda *extractScoredLabelsFromResponse* iz dobivenog odgovora, od strane Azure API-ja, uzima *string* koji predstavlja vrijednost *boolean*. Programski kod metode *extractScoredLabelsFromResponse* je prikazan slikom 4.20.



Slika 4.20 Metoda *extractScoredLabelsFromResponse* unutar klase *AzureMLService*

Metoda *generateEvaluationMessage* koristi metodu *evaluate* kako bi svaki parametar usporedila s unaprijed definiranim granicama za vrijednosti atributa. Atributi mogu imati ispodprosječnu i iznadprosječnu vrijednost. Na kraju, dodaje se poruka o mogućnosti dijabetesa na temelju rezultata dobivenog iz Azure API-ja. Programski kod metode *generateEvaluationMessage* je prikazan slikom 4.21.

Slika 4.21 Metoda *generateEvaluationMessage* unutar klase *AzureMLService*

4.5 Programsko rješenje na strani korisnika

Rješenje je kreirano pomoću React biblioteke. Korisničko sučelje je podijeljeno na više različith direktorija, a najbitniji je direktorij components. Direktorij *components* koji je prikazan slikom 3.1 sadrži sve komponente korisničkog sučelja. Komponente su samostalni dijelovi korisničkog sučelja koji u sebi sadrže vlastitu strukturu, stil i ponašanje, te se mogu iskoristiti više puta kroz različite dijelove web sustava.

4.5.1 Komponenta *App*

App je glavna komponenta koja sadrži osnovnu strukturu sučelja. Ona je korijenska komponenta iz koje se sve ostale komponente postavljaju. Njen programski kod prikazan je slikom 4.22.

```
function App() {  Show usages   Antonio Šušovček *
    return (
        <div className="App">
            <Routes>
                <Route path="/auth" element={<Auth />} />
                <Route path="/home" element={<PrivateRoute element={<Home />} />} />
                <Route path="/medical-record/:patientId" element={<PrivateRoute element={<MedicalRecord />} />} />
                <Route path="/medical-record/new" element={<NewMedicalRecord />} />
                <Route path="/" element={<Navigate to="/auth" />} />
            </Routes>
        </div>
    );
}
```

Slika 4.22 Komponenta *App*

4.5.2 Komponenta *Auth*

Ova komponenta upravlja prikazom komponenti *Login* i *Register*. Pomoću vrijednosti *isRegistering* upravlja se koja komponenta je prikazana korisniku. Pomoću tipki *Login* i *Register* ta vrijednost se izmjenjuje. Programski kod komponente *Auth* prikazan je slikom 4.23.

```
function Auth() {  Show usages   Antonio Šušovček
    const [isRegistering : boolean , setIsRegistering] = useState( initialState: true);

    const toggleForm = () : void  => {  Show usages   Antonio Šušovček
        setIsRegistering(!isRegistering);
    };

    return (
        <div className="auth-container">
            <div className="auth-form">
                <h2>{isRegistering ? 'Register an account' : 'Login'}</h2>
                {isRegistering ? (
                    <>
                        <Register />
                        <p>
                            Already have an account?
                            <button className="toggle-button" onClick={toggleForm}>Login</button>
                        </p>
                    </>
                ) : (
                    <>
                        <Login />
                        <p>
                            Don't have an account?
                            <button className="toggle-button" onClick={toggleForm}>Register</button>
                        </p>
                    </>
                )}
            </div>
        </div>
    );
}
```

Slika 4.23 Komponenta *Auth*

Login je komponenta koja omogućuje korisnicima da unesu adresu e-pošte i lozinku te se pokušaju prijaviti u sustav. Kada korisnik klikne na gumb *Login*, poziva se funkcija *handleSubmit*. Ona pokušava prijaviti korisnika pozivanjem funkcije *login* pomoću unesene adrese e-pošte i lozinke. *Login* je funkcija koja se poziva iz komponente *AuthContext* kako bi poslao HTTP zahtjev na poslužiteljski dio web sustava. Ako je prijava uspješna, korisnik se preusmjerava na stranicu *Home* pomoću funkcije *navigate*, te se njegov ID sprema za daljnje korištenje. Ako prijava ne uspije, postavlja se odgovarajuća poruka o pogrešci. Programski kod funkcije *login* i komponente *Login* je prikazan slikama 4.24 i 4.25.

```
function Login() {  Show usages  Antonio Šušovček *
    const [email : string , setEmail] = useState( initialState: '');
    const [password : string , setPassword] = useState( initialState: '');
    const [errorMessage : string , setErrorMessage] = useState( initialState: '');
    const { login } = useAuth();
    const navigate : NavigateFunction  = useNavigate();

    const handleSubmit = async (e) : Promise<void> => {  Show usages  Antonio Šušovček *
        e.preventDefault();
        setErrorMessage( value: '');

        try {
            await login(email, password);
            navigate('/home');
        } catch (error) {
            setErrorMessage( value: 'Login failed: Invalid email or password');
        }
    };

    return (
        <div>
            <form onSubmit={handleSubmit}>
                <div>
                    <label>Email:</label>
                    <input
                        type="email"
                        value={email}
                        onChange={(e : ChangeEvent<HTMLInputElement> ) : void  => setEmail(e.target.value)}
                        required
                    />
                </div>
                <div>
                    <label>Password:</label>
                    <input
                        type="password"
                        value={password}
                        onChange={(e : ChangeEvent<HTMLInputElement> ) : void  => setPassword(e.target.value)}
                        required
                    />
                </div>
                <button type="submit">Login</button>
            </form>
            {errorMessage && (
                <div style={{ color: 'red', marginTop: '10px' }}>
                    {errorMessage}
                </div>
            )}
        </div>
    );
```

Slika 4.24 Komponenta *Login*

```
const login = async (email, password) : Promise<void> => { Show usages  Antonio Šušovček *
    try {
        const response : AxiosResponse<any> = await axios.post( url: 'http://localhost:8080/api/patient/login', data: { email, password }, config: {
            headers: { 'Content-Type': 'application/json' },
        });


        if (response.data && response.data.id) {
            setIsAuthenticated( value: true);
            setPatientId(response.data.id);
            localStorage.setItem('patientId', response.data.id);
            navigate('/home');
        } else {
            throw new Error('Invalid response format');
        }
    } catch (error) {
        throw error;
    }
};
```

Slika 4.25 Funkcija *login* unutar komponente *AuthContext*

Komponenta *Register* omogućuje korisnicima da unesu svoje podatke i registriraju se kao pacijenti unutar sustava. Korisnik unosi svoje ime i prezime, adresu e-pošte i lozinku. *HandleChange* se poziva svaki put kada korisnik promijeni nešto u unosu kako bi ažurirao odgovarajuću vrijednost patient objekta. Kada se klikne tipka *Register*, poziva se funkcija *handleSubmit* koja objekt *patient* predaje na obradu funkciji *registerPatient* koji se nalazi unutar komponente *api*. Ta funkcija šalje HTTP zahtjev na poslužiteljski dio kako bi se registracija obradila. Ovisno o uspješnosti ili neuspješnosti registracije, postavlja se odgovarajuća poruka. Programski kod funkcije *registerPatient* i komponente *Register* je prikazan slikama 4.26 i 4.27.

```
export const registerPatient = async (patient) : Promise<...> => { Show usages  Antonio Šušovček
    try {
        const response : AxiosResponse<any> = await axios.post( url: `${API_BASE_URL}/patient/save`, patient);
        return response.data;
    } catch (error) {
        console.error('Error registering patient:', error);
        throw error;
    }
};
```

Slika 4.26 Funkcija *registerPatient* unutar komponente *api*

```
function Register() {  Show usages  Antonio Šušovček
    const [patient : {…} , setPatient] = useState( initialState: {
        firstName: '',
        lastName: '',
        email: '',
        password: '',
    });

    const navigate : NavigateFunction  = useNavigate();

    const handleChange = (e) : void  => {  Show usages  Antonio Šušovček
        const { name, value } = e.target;
        setPatient( value: { ...patient, [name]: value });
    };

    const handleSubmit = async (e) : Promise<void>  => {  Show usages  Antonio Šušovček
        e.preventDefault();
        try {
            await registerPatient(patient);
            alert('Patient registered successfully');
            navigate('/');
        } catch (error) {
            alert('Failed to register patient');
        }
    };

    return (
        <div>
            <form onSubmit={handleSubmit}>
                <div>
                    <label>First Name:</label>
                    <input type="text" name="firstName" value={patient.firstName} onChange={handleChange} required />
                </div>
                <div>
                    <label>Last Name:</label>
                    <input type="text" name="lastName" value={patient.lastName} onChange={handleChange} required />
                </div>
                <div>
                    <label>Email:</label>
                    <input type="email" name="email" value={patient.email} onChange={handleChange} required />
                </div>
                <div>
                    <label>Password:</label>
                    <input type="password" name="password" value={patient.password} onChange={handleChange} required />
                </div>
                <button type="submit">Register</button>
            </form>
        </div>
    );
}
```

Slika 4.27 Komponenta *Register*

4.5.3 Komponente *MedicalRecord* i *NewMedicalRecord*

Obje komponente pokrivaju unos i spremanje nalaza pacijenta, no na različite načine. *NewMedicalRecord* se koristi za unos novih medicinskih podataka o pacijentu, dok se *MedicalRecord* koristi kako bi se dohvatili postojeći podaci i ažurirali. Komponente koriste funkciju *handleChange* za ažuriranje stanja podataka na temelju korisnikovog unosa. Koriste funkcije *handleAzure* i *handleGPT* kako bi preko poslužitelja komunicirali s istoimenim API uslugama te dobili povratne odgovore na temelju unesenih podataka. Komuniciraju s poslužiteljem kako bi spremili novi tj. ažurirali postojeći nalaz u bazi podataka. Komponenta *MedicalRecord* ima dodatnu funkciju *fetchMedicalRecord* kako bi dohvatila već spremljene podatke preko ID-a nalaza. Programski kod komponente *MedicalRecord* je prikazan na slici 4.28.

```
function MedicalRecord() {
    const { patientId } = useParams();
    const [medicalRecord, setMedicalRecord] = useState({
        gender: '',
        age: '',
        urea: '',
        cr: '',
        hba1c: '',
        chol: '',
        tg: '',
        hdl: '',
        ldl: '',
        vldl: '',
        bmi: '',
        hasDiabetes: false
    });
    const [gptResponse, setGptResponse] = useState('');
    const [azureResponse, setAzureResponse] = useState('');
    const navigate = useNavigate();

    useEffect(() => {
        const fetchMedicalRecord = async () => {
            try {
                const response = await axios.get(`http://localhost:8080/api/patient/medical-record/${patientId}`);
                setMedicalRecord(response.data);
            } catch (error) {
                console.error('Error fetching medical record:', error);
            }
        };

        fetchMedicalRecord();
    }, [patientId]);

    const handleChange = (e) => {
        const { name, value, type, checked } = e.target;
        const val = type === 'checkbox' ? checked : value;
        setMedicalRecord({ ...medicalRecord, [name]: val });
    };

    const handleSave = async () => {
        try {
            await saveMedicalRecord(localStorage.getItem('patientId'), medicalRecord);
            alert('Medical record saved successfully');
        } catch (error) {
            alert('Failed to save medical record');
        }
    };

    const handleGPT = async () => {
        try {
            const response = await getGPTResponse(localStorage.getItem('patientId'), medicalRecord);
            setGptResponse(response);
        } catch (error) {
            alert('Failed to get GPT response');
        }
    };
```

```
    const handleAzure = async () => {
        try {
            const response = await getAzureResponse(localStorage.getItem('patientId'), medicalRecord);
            setAzureResponse(response);
        } catch (error) {
            alert('Failed to get Azure response');
        }
    };

    const handleBack = () => {
        navigate('/home');
    };

    return (
        <div className="container">
            <div className="form-container">
                <h2>Edit Medical Data</h2>
                <form>
                    <div>
                        <label>Gender:</label>
                        <input type="text" name="gender" value={medicalRecord.gender} onChange={handleChange}
required />
                    </div>
                    <div>
                        <label>Age:</label>
                        <input type="number" name="age" value={medicalRecord.age} onChange={handleChange}
required />
                    </div>
                    <div>
                        <label>Urea:</label>
                        <input type="number" name="urea" value={medicalRecord.urea} onChange={handleChange}
required />
                    </div>
                    <div>
                        <label>Creatinine (Cr):</label>
                        <input type="number" name="cr" value={medicalRecord.cr} onChange={handleChange}
required />
                    </div>
                    <div>
                        <label>HbA1c:</label>
                        <input type="number" step="0.1" name="hba1c" value={medicalRecord.hba1c}
onChange={handleChange} required />
                    </div>
                    <div>
                        <label>Cholesterol (Chol):</label>
                        <input type="number" name="chol" value={medicalRecord.chol} onChange={handleChange}
required />
                    </div>
                    <div>
                        <label>Triglycerides (TG):</label>
                        <input type="number" name="tg" value={medicalRecord.tg} onChange={handleChange}
required />
                    </div>
                    <div>
                        <label>HDL:</label>
                        <input type="number" name="hdl" value={medicalRecord.hdl} onChange={handleChange}
required />
                    </div>
                    <div>
                        <label>LDL:</label>
                        <input type="number" name="ldl" value={medicalRecord.ldl} onChange={handleChange}
required />
                    </div>
                    <div>
                        <label>VLDL:</label>
                        <input type="number" name="vldl" value={medicalRecord.vldl} onChange={handleChange}
required />
                    </div>
                    <div>
                        <label>BMI:</label>
                        <input type="number" step="0.1" name="bmi" value={medicalRecord.bmi}
onChange={handleChange} required />
                    </div>
                </form>
            </div>
            <div className="response-container">
                <div className="button-group">
                    <button type="button" onClick={handleGPT}>Analize data through GPT</button>
                    <div className="response-section">
                        {gptResponse && (
                            <div>
                                <h3>GPT Response:</h3>
                                <p>{gptResponse}</p>
                            </div>
                        )}
                    </div>
                    <button type="button" onClick={handleAzure}>Analize data through Azure</button>
```

```
                    <div className="response-section">
                        {azureResponse && (
                            <div>
                                <h3>Azure Response:</h3>
                                <p>{azureResponse}</p>
                            </div>
                        )}
                    </div>
                </div>
                <button type="button" onClick={handleSave}>Update Medical Record</button>
                <button type="button" className="back-button" onClick={handleBack}>Back to Home</button>
            </div>
        </div>
    );
}
```

Slika 4.28 Komponenta *MedicalRecord*

4.5.4 Komponenta *Home*

Komponenta *Home* služi kao početna stranica korisniku nakon prijave. Prikazuje sve medicinske nalaze pacijenta koji su spremljeni u bazi i omogućuje upravljanje njima. Funkcija *fetchMedicalRecords* omogućuje dohvaćanje tih nalaza iz baze pomoću poslužitelja, koristeći patientId spremljen tijekom prijave. Funkcija *handleRecordClick* omogućuje korisniku da otvori odabrani nalaz pomoću funkcionalnosti navigate. Funkcija *handleDeleteRecord* omogućuje korisniku da obriše odabrani nalaz iz baze podataka. Tipka *Logout* omogućuje korisniku da se odjavi, te je potrebno opet se prijaviti ako želi pristupiti svojim nalazima. Za razliku od nje, tipka *handleDeletePatient* trajno briše pacijentov račun iz baze podataka zajedno s njegovim nalazima, te je potrebno registrirati novi račun ako želi opet pristupiti aplikaciji. Programski kod komponente *Home* je prikazan na slikama 4.29 i 4.30.

```
function Home() {  Show usages  Antonio Šušovček *
    const [medicalRecords : any[] , setMedicalRecords] = useState( initialState: []);
    const { patientId, logout } = useAuth();
    const navigate : NavigateFunction  = useNavigate();

    useEffect( effect: () : void  => {
        const fetchMedicalRecords = async () : Promise<void>  => {  Show usages  Antonio Šušovček
            if (!patientId) {
                console.error("No patient ID found");
                return;
            }

            try {
                const response : AxiosResponse<any>  = await axios.get( url: `http://localhost:8080/api/patient/${patientId}/medicalrecord`);
                setMedicalRecords(response.data);
            } catch (error) {
                console.error("Error fetching medical records:", error);
            }
        };

        fetchMedicalRecords();
    }, deps: [patientId]);

    const handleRecordClick = (recordId) : void  => {  Show usages  Antonio Šušovček
        navigate(`/medical-record/${recordId}`);
    };

    const handleAddNewRecord = () : void  => {  Show usages  Antonio Šušovček
        navigate('/medical-record/new');
    };

    const handleDeleteRecord = async (recordId) : Promise<void>  => {  Show usages  new *
        try {
            await axios.delete( url: `http://localhost:8080/api/patient/medicalrecord/${recordId}`);
            setMedicalRecords(medicalRecords.filter(record => record.id !== recordId));
        } catch (error) {
            console.error("Error deleting medical record:", error);
        }
    };
```

Slika 4.29 Komponenta *Home* (a)

```
    const handleDeletePatient = async () : Promise<void>  => {  Show usages  new *
        try {
            await axios.delete( url: `http://localhost:8080/api/patient/${patientId}`);
            logout();
            navigate('/auth');
        } catch (error) {
            console.error("Error deleting patient:", error);
        }
    };

    return (
        <div className="home-container">
            <h2>Your Medical Records</h2>
            <button onClick={handleAddNewRecord}>Add New Medical Record</button>
            <button onClick={handleDeletePatient}>Delete Your Account</button>
            <button onClick={logout}>Logout</button>
            <div className="records-list">
                {medicalRecords.length > 0 ? (
                    medicalRecords.map(record => (
                        <div key={record.id} className="record-box">
                            {`Record ID: ${record.id}`}
                            <button onClick={() : void  => handleRecordClick(record.id)}>View</button>
                            <button onClick={() : Promise<void>  => handleDeleteRecord(record.id)}>Delete</button>
                        </div>
                    ))
                ) : (
                    <p>No medical records found.</p>
                )}
            </div>
        </div>
    );
}
```

Slika 4.30 Komponenta *Home* (b)

5. PRIKAZ RADA WEB APLIKACIJE, ISPITIVANJE I ANALIZA REZULTATA

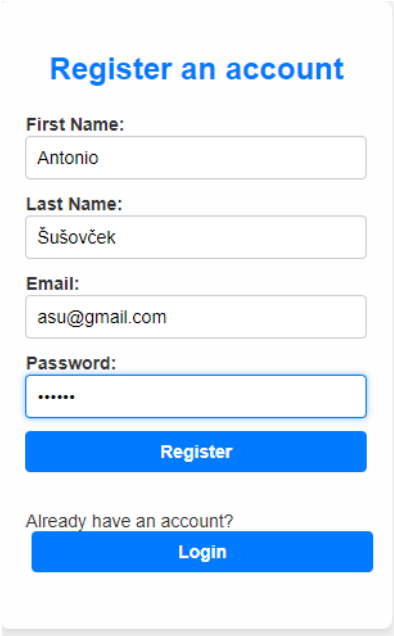
U ovom poglavlju su prikazane upute za korištenje aplikacije, te ispitivanje rad aplikacije na nekoliko korisničkih slučajeva.

5.1 Prikaz rada web aplikacije

Kako bi aplikacija radila, potrebno je pokrenuti Postgres lokalnu bazu podataka, Spring Boot poslužiteljsku aplikaciju te React klijentsku aplikaciju. Aplikaciji se tada pristupa preko URL-ahttp://localhost:3000/.

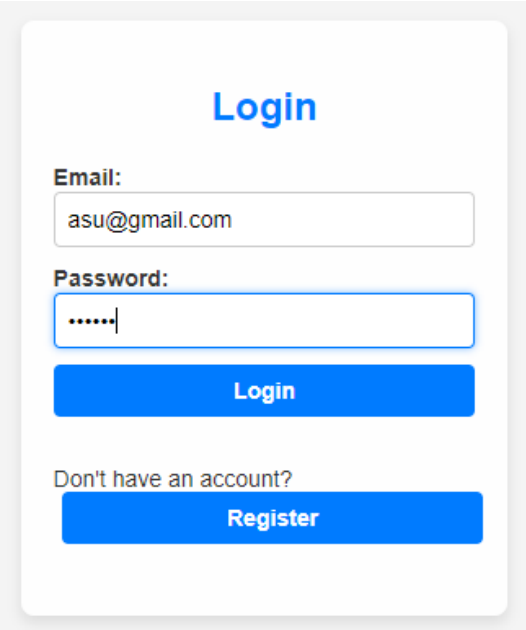
5.1.1 Registracija i prijava korisnika

Na slici 5.1 prikazan je zaslon za registraciju korisnika. Korisnik treba popuniti sve podatke kako bi se mogao registrirati. Nakon što korisnik unese sve podatke, klikne na tipku *Register* kako bi se registrirao.



Slika 5.1 Zaslon za registraciju

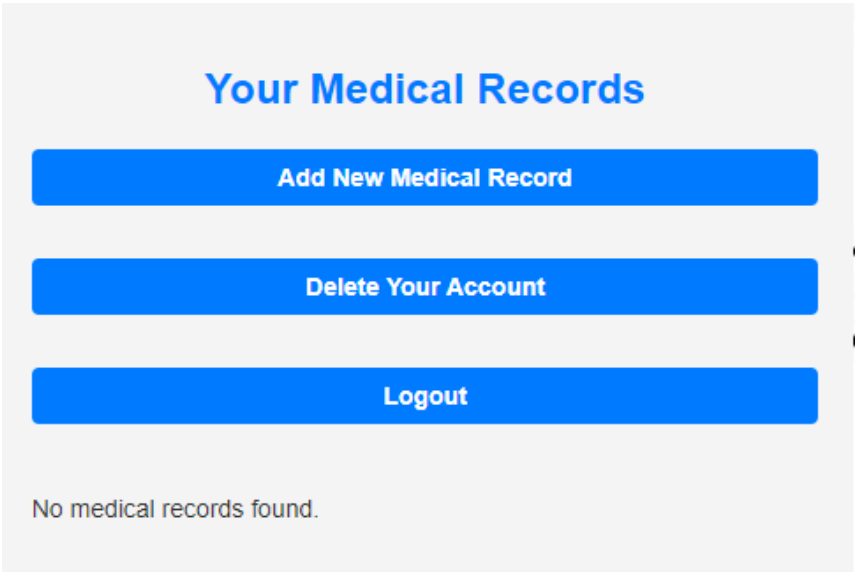
Nakon toga, ako je registracija uspješna, sustav korisnika prebacuje na zaslon za prijavu prikazan slikom 5.2. Ovdje korisnik treba unijeti podatke pomoću kojih se prethodno registrirao te kliknuti na tipku *Login*. Ako podaci nisu ispravno uneseni, sustav javlja grešku. Ako je korisnik ispravno unio podatke, on je prijavljen te može pristupiti glavnom zaslonu aplikacije.



Slika 5.2 Zaslon za prijavu

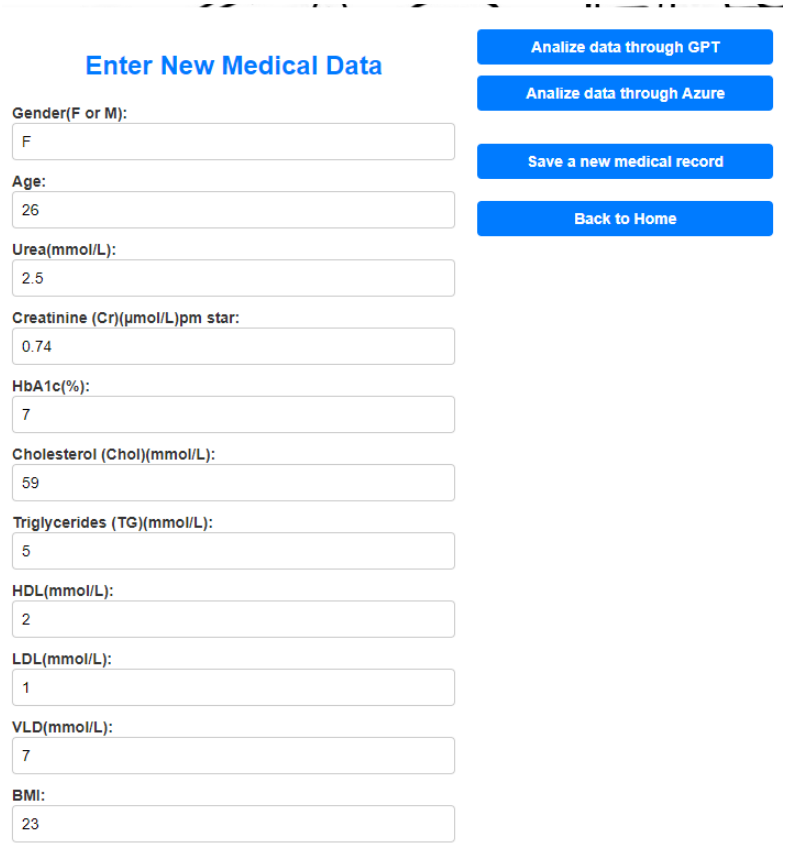
5.1.2 Dodavanje novog nalaza

Kada korisnik pristupi glavnom zaslonu, vidljiv je prikaz na slici 5.3. Tipka *Delete Your Account* se koristi kada korisnik želi potpuno obrisati svoj račun zajedno sa svim nalazima koje je kreirao. Tipka *Logout* se koristi kako bi se korisnik odjavio iz aplikacije. Obje tipke korisnika vraćaju na zaslon za prijavu. Pošto se korisnik registrirao i prijavio po prvi put, njegova lista nalaza je trenutno prazna. Kako bi dodao novi nalaz, korisnik odabire tipku *Add New Medical Record.*



Slika 5.3 Zaslon Home

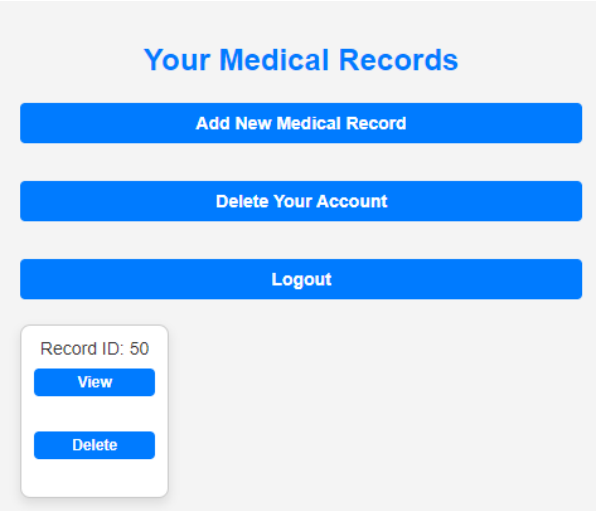
Korisniku je nakon pritiska tipke *Add New Medical Record* predstavljen zaslon prikazan slikom 5.4. Na njemu je omogućen unos medicinskih podataka kako bi se moglo spremiti tipkom *Save a new medical record*. Također, korisnik može zatražiti analizu unesenih podataka pomoću tipki *Analize data through GPT* te *Analize data through Azure*. Klikom na obje tipke korisnik dobiva dvije različite analize te ih može usporediti. Tipkom *Back to Home* korisnik se može vratiti na glavni zaslon.



Slika 5.4 Zaslon za stvaranje novog nalaza

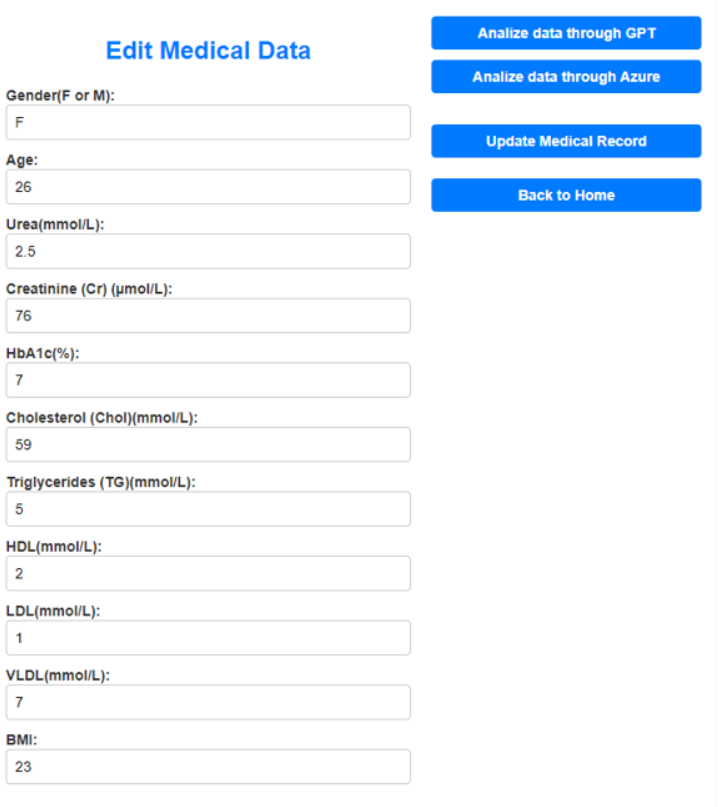
5.1.3 Uređivanje postojećeg nalaza

Nakon što je spremio nalaz, korisnikov glavni zaslon se sada promijenio kao na slici 5.5. Može vidjeti da se sada na njemu nalazi prethodni nalaz koji je unio u sustav. Nalaz može pregledati i uređivati klikom na tipku *View* ili ga može obrisati pomoću tipke *Delete*.



Slika 5.5 Zaslon Home s unesenim nalazom

Ako korisnik klikne na tipku *View* pristupa zaslonu koji je prikazan slikom 5.6. Na njemu su podaci već uneseni jer se radi o postojećem nalazu kojeg je korisnik prethodno kreirao. Korisnik te postojeće podatke može ažurirati novim vrijednostima. Ovdje je također omogućena analiza medicinskih podataka pomoću GPT i Azure usluga. Klikom na *Update Medical Record* korisnik ažurira nalaz te je vraćen na glavni zaslon.



Slika 5.6 Zaslon za ažuriranje postojećeg nalaza

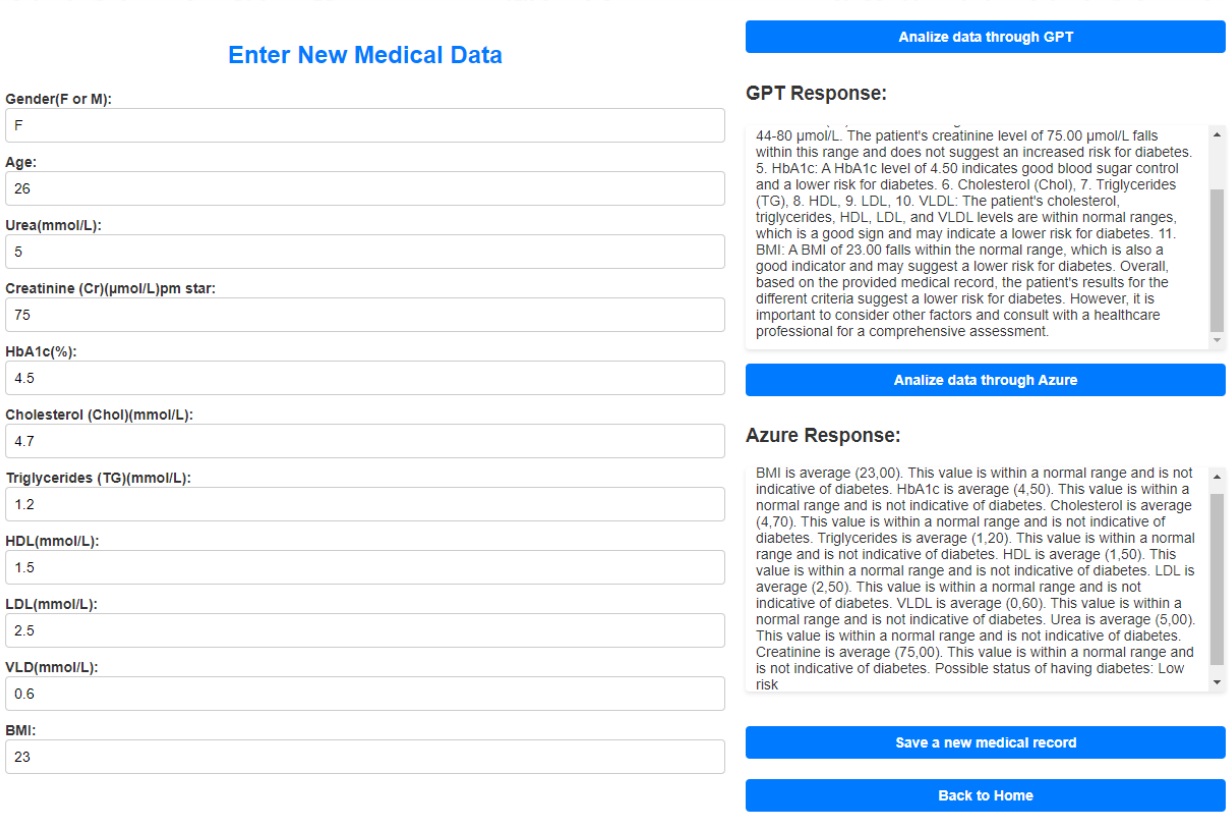
5.2 Ispitivanje rada web aplikacije

Rad je ispitan na tri korisnička slučaja i to:

- kada korisnik ima nizak rizik bolovanja od dijabetesa, odnosno podatci korisnika su unutar granica normalnih vrijednosti
- kada korisnik ima visok rizik bolovanja od dijabetesa, odnosno podatci korisnika su izvan granica normalnih vrijednosti
- kada su neki od podataka korisnika unutar normalnog raspona vrijednosti, a neki izvan njega.

5.2.1 Korisnik s malim rizikom obolijevanja od dijabetesa

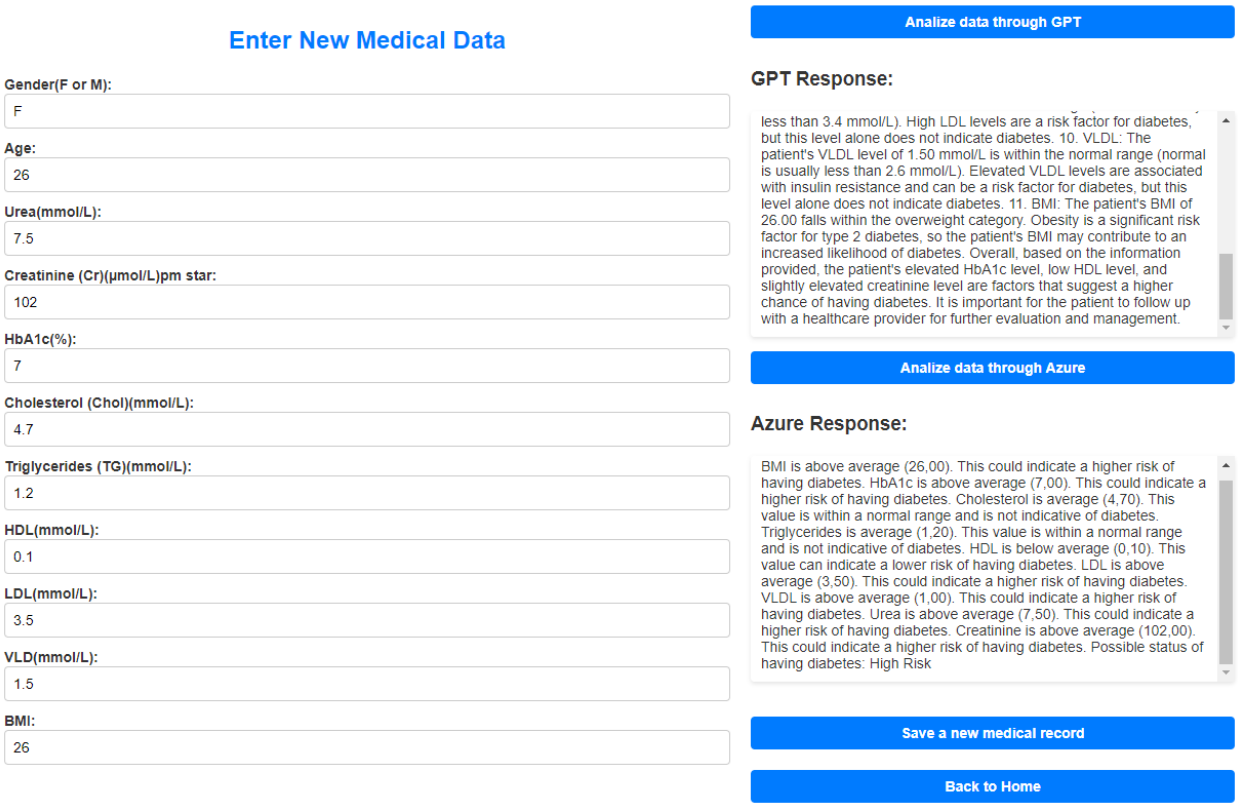
U ovom slučaju korisnik kreira novi nalaz te unosi vrijednosti koje se sve nalaze u normalnim razinama. Nakon toga korisnik pritišće tipke povezane uz Azure i GPT usluge kako bi se njegovi podaci analizirali. Na slici 5.7 mogu se vidjeti uneseni podatci te se mogu vidjeti i analize usluga. Može se vidjeti da su obje usluge zaključile da korisnik najvjerojatnije ne boluje od dijabetesa.



Slika 5.7 Vrijednosti parametara korisnika s malom vjerojatnošću obolijevanja od dijabetesa

5.2.2 Korisnik s velikim rizikom bolovanja od dijabetesa

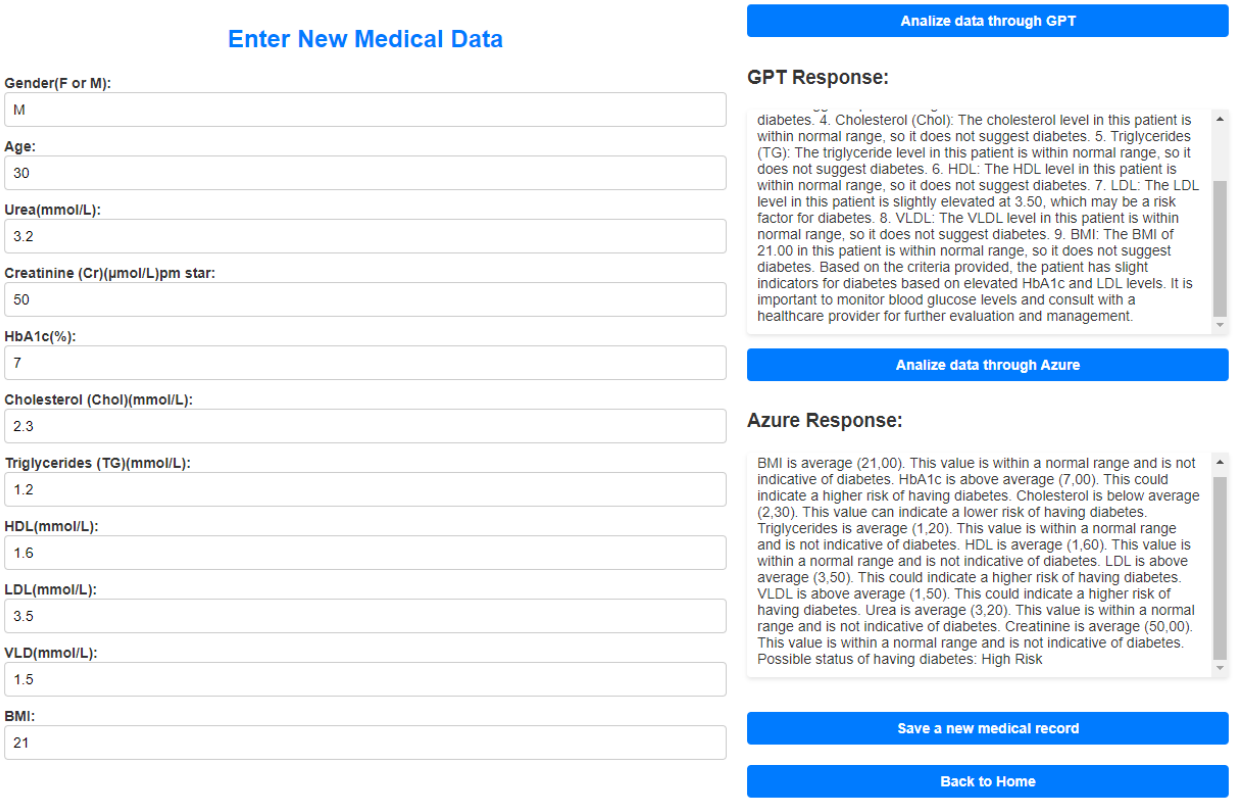
Ovaj slučaj se izvršava na sličan način kao i slučaj u poglavlju 5.2.1, no ovaj put korisnikove vrijednosti su iznad dozvoljenih vrijednosti. Nakon što unese podatke i klikne na obje tipke, na slici 5.8 može se vidjeti da su i ovaj put obje usluge došle do istog zaključka, a to je da korisnik ima povećane šanse bolovanja od dijabetesa te da bi trebao obaviti posjet doktoru.



Slika 5.8 Vrijednosti parametara korisnika s velikom vjerojatnošću obolijevanja od dijabetesa

5.2.3 Korisnik s neizvjesnim rizikom bolovanja od dijabetesa

Ovaj slučaj je izvršen na identičan način kao prethodna dva, no ovaj put podatci su izmiješani tako da su neki od njih povišeni, a neki u normalnim rasponima vrijednosti. Rezultat tog je prikazan slikom 5.9. Vidljivo je da su se usluge složile da su vrijednosti HbA1c i LDL povišene te da zbog toga predlažu odlazak doktoru na pregled.



Slika 5.9 Vrijednosti parametara korisnika s neizvjesnom vjerojatnošću obolijevanja od dijabetesa

5.2.4 Analiza dobivenih rezultata

Moguće je vidjeti da su usluge u sva tri slučaja uskladili svoje rezultate. Odstupanja potencijalno mogu nastati kod određenih podataka zbog toga što se na Azure usluzi granice normalnih vrijednosti trebaju postaviti unutar poslužiteljskog dijela sustava, osim za konačni rezultat rizika bolovanja od dijabetesa. Za razliku od njega GPT usluzi se samo šalju podaci te on na temelju vlastitih kriterija određuje koliko iznose granice normalnih vrijednosti.

GPT model koristi napredne tehnike dubokog učenja za analizu i interpretaciju podataka. Kada prima podatke, model koristi svoje prethodno stečeno znanje i algoritme za procjenu granica normalnih vrijednosti. Ovo omogućava modelu da pruža fleksibilnije i preciznije rezultate u odnosu na dinamične podatke koje korisnik unosi. Međutim, model ima kao cilj stvoriti nove podatke na temelju unosa, te zbog toga se može dogoditi da model daje netočne informacije koje nisu stvarne, nego ih je on stvorio u tom trenutku.

Za razliku od njega, model strojnog učenja stvoren na platformi Azure ima za cilj da predvidi i kategorizira podatke na temelju već postojećih podataka pomoću kojih je treniran, što znači da ne

stvara nove podatke poput GPT modela. Ovaj pristup može biti manje fleksibilan u odnosu na GPT model, ali omogućava dosljednost i stabilnost u predviđanjima na temelju poznatih obrazaca.

Korištenjem oba modela unutar sustava korisniku je omogućeno da na dva različita načina analizira svoje medicinske podatke. GPT model može pružiti dublje, personalizirane uvide i preporuke, dok Azure model može pružitii dosljednu i standardiziranu procjenu rizika. Primjenom oba modela, korisnik može imati bolje razumijevanje svojih zdravstvenih podataka, što može dovesti do učinkovitijih preventivnih mjera kad se radi o riziku obolijevanja od dijabetesa.

6. ZAKLJUČAK

U ovom radu osmišljen je, modeliran i programski ostvaren web sustav za potporu u dijagnosticiranju i liječenju dijabetesa korištenjem strojnog učenja i generativne umjetne inteligencije. Sustav korisnicima omogućuje analizu medicinskih podataka i procjenu rizika od dijabetesa, pružajući povratne informacije i preporuke za daljnje postupke. Web aplikacija izrađena je korištenjem tehnologija poput Spring Boota, Reacta i PostgreSQL-a, dok su rješenje zasnovano na strojnom učenju i na generativnoj umjetnoj inteligenciji implementirani pomoću algoritama kao što su stablo odluke i modeli GPT-a. Obje usluge unutar sustava se pozivaju odgovarajućim funkcijama te vraćaju tekstualne odgovore opisujući dane podatke i procjenjuju rizik bolovanja od dijabetesa.

Rad web aplikacije ispitan je kroz nekoliko korisničkih slučajeva, pri čemu su rezultati pokazali da sustav pouzdano analizira podatke i daje relevantne preporuke bez pogrešaka. Model strojnog učenja pokazao se učinkovitim u klasifikaciji korisnika prema riziku od dijabetesa, dok generativna umjetna inteligencija pruža dodatne upute na temelju unesenih podataka. Iako sustav pokazuje značajan potencijal u podršci dijagnosticiranja i liječenja dijabetesa, važno je napomenuti da se rezultati dobiveni korištenjem ovog rješenja ne mogu smatrati zamjenom za stručne medicinske savjete liječnika. Nadalje, postoji prostor za daljnja poboljšanja, kao što su povećanje točnosti modela, proširenje funkcionalnosti za praćenje i liječenje drugih bolesti, te optimizacija rada aplikacije. Razvoj takvih sustava predstavlja korak naprijed prema personaliziranoj medicini i poboljšanju zdravstvene skrbi za pacijente s kroničnim bolestima poput dijabetesa.

LITERATURA

[1] World Health Organization, Classification of Diabetes Mellitus, str. 6, 2019.

[2] M. Krause i G. De Vito, Type 1 and Type 2 Diabetes Mellitus: Commonalities, Differences and the Importance of Exercise and Nutrition, Nutrients, sv. 15, str. 1-2, 2023.

[3] World Health Organization, Use of Glycated Haemoglobin (HbA1c) in the Diagnosis of Diabetes Mellitus, str. 6-9, 2011.

[4] P. K. Dabla, Renal Function in Diabetic Nephropathy, World Journal of Diabetes, sv. 1, str. 50-51, 2010.

[5] F. Q. Nutall, Obesity, BMI, and Health: A Critical Review, Nutrition Today, sv. 50, str. 117-120, 2015.

[6] S. B. Pathan, P. Jawade i P. Lalla, Correlation of Serum Urea and Serum Creatinine in Diabetics Patients and Normal Individuals, International Journal of Clinical Biochemistry and Research, sv. 11, str. 45-48, 2024.

[7] M. Laakso, Lipid Disorders in Type 2 Diabetes, Endocrinol Nutr, sv. 56, str. 43-45 2009.

[8] World Health Organization, Management of Diabetes Mellitus, str. 9-16, 1994.

[9] J. Goecks, V. Jalili, L. M. Heiser i J. W. Gray, How Machine Learning Will Transform Biomedicine, Cell, sv. 181, str. 93-97, 2020.

[10] Ardila, D., Kiraly, A.P., Bharadwaj, End-To-End Lung Cancer Screening with Three-Dimensional Deep Learning on Low-Dose Chest Computed Tomography. Natural Medicine, sv. 25, str. 954–961, 2019.

[11] L.M. García-Marín, P. Reyess-Pérez, S. Diaz Torres, A. Medina-Rivera, N. Martin, B. Mitchell, M. E. Renteria, Shared Molecular Genetic Factors Influence Subcortical Brain Morphometry and Parkinson's Disease Risk, npj Parkinson s Disease, sv. 75, str. 1-7, 2023.

[12] Electronic Medical and Health Records, Ignite Healthwise, [Online]. Dostupno na: https://healthy.kaiserpermanente.org/health-wellness/health-encyclopedia/he.electronic-medical-and-health-records.abo4950 [04.09.2024.]

[13] M. Helfand Christensen, J. Anderson, EarlySense for Monitoring Vital Signs in Hospitalized Patients, Portland VA Healthcare System, str. 1-6, 2016.

[14] Better Healthcare Services with AI Chatbots, IBM, [Online]. Dostupno na: https://www.ibm.com/products/watsonx-assistant/healthcare [04.09.2024.]

[15] AlphaFold, Google, [Online]. Dostupno na: https://alphafold.ebi.ac.uk/about. [04. 09. 2024.].

[16] 5725-W51 IBM Watson for Oncology, IBM, [Online]. Dostupno na: https://www.ibm.com/docs/en/announcements/watson-oncology?region=CAN. [04. 09. 2024.].

[17] Functional vs Non Functional Requirements, GeeksForGeeks, [Online]. Dostupno na: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/ [04. 09. 2024.]

[18] MVC Framework Introduction, GeeksForGeeks, [Online]. Dostupno na: https://www.geeksforgeeks.org/mvc-framework-introduction/ [04. 09. 2024.]

[19] React Architecture Pattern and Best Practices in 2024, GeeksForGeeks, [Online]. Dostupno na: https://www.geeksforgeeks.org/react-architecture-pattern-and-best-practices/ [04.09.2024.]

[20] A.F. Alnuaimi i T. H. Albaldawi, An Overview of Machine Learning Classification Techniques, BIO Web of Conferences, sv. 97, str. 4-19, 2024.

[21] Sami, I. Uddin, N. Fayyaz, M. Bilal, M. Shahid i I. Ali, Getting to Know ChatGPT: An Introduction to Implementation and Working, Proceedings of 1st International Conference on Computing Technologies, Tools and Applications, sv.1, str. 273-276, Peshawar, 2023.

[22] Bud-D: Enabling Bidirectional Communication with ChatGPT by Adding Listening and Speaking Capabilities, FMDB , sv. 1, str. 52, 2023.

[23] What is Java Spring Boot, IBM, [Online]. Dostupno na: https://www.ibm.com/topics/java-spring-boot. [04. 09. 2024.].

[24] About, The PostgreSQL Global Development Group, [Online]. Dostupno na: https://www.postgresql.org/about/. [04. 09. 2024.].

[25] What is Maven?, The Apache Software Foundation, [Online]. Dostupno na: https://maven.apache.org/what-is-maven.html. [04. 09. 2024.].

[26] Introducing ChatGPT, OpenAI, [Online]. Dostupno na: https://openai.com/index/chatgpt/. [04. 09. 2024.].

[27] What is Azure?, Microsoft, [Online]. Dostupno na: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure/. [04. 09. 2024.].

[28] React, Meta OpenSource, [Online]. Dostupno na: https://react.dev/. [04. 09. 2024.].

[29] IntelliJ IDEA, Jetbrains, [Online]. Dostupno na: https://www.jetbrains.com/idea/. [04. 09. 2024.].

[30] AravindPCoder, Diabetes Dataset, Kaggle, [Online]. Dostupno na: https://www.kaggle.com/datasets/aravindpcoder/diabetes-dataset/data. [04. 09. 2024.].

[31] N. V. Chawla, K. W. Bowyer, L. O. Hall i W. P. Kegelmeyer, SMOTE: Synthetic Minority Over-sampling Technique, Journal of Artificial Intelligence Research, sv. 16, str. 328-331, 2002.

[32] Evaluate Automated Machine Learning Experiment Results, Microsoft, [Online]. Dostupno na: https://learn.microsoft.com/hr-hr/azure/machine-learning/how-to-understand-automated-ml?view=azureml-api-2. [04. 09. 2024.].

[33] OpenAI Developer Platform, OpenAI, [Online]. Dostupno na: https://platform.openai.com/docs/overview. [04. 09. 2024.].

[34] Prompt Engineering Guide, DAIR.AI, [Online]. Dostupno na: https://www.promptingguide.ai/ [04.09.2024.]

SAŽETAK

U ovom diplomskom radu istražuju se izazovi u dijagnosticiranju i liječenju dijabetesa korištenjem naprednih tehnologija. Prikazani su postojeći sustavi koji koriste strojno učenje i generativnu umjetnu inteligenciju u biomedicinskim aplikacijama. Kreiran je model strojnog učenja temeljen na klasifikacijskom algoritmu stabla odluke te dodan model generativne umjetne inteligencije za analizu unesenih podataka. Razvijen je web sustav koji korisnicima omogućuje unos medicinskih podataka, procjenu rizika od dijabetesa te dobivanje preporuka za daljnje liječenje. Ispitivanjem je potvrđena ispravnost implementiranih funkcionalnosti i zadovoljavajuća točnost primijenjenih algoritama strojnog učenja te generativne umjetne inteligencije.

Ključne riječi: dijabetes, generativna umjetna inteligencija, strojno učenje, web sustav.

ABSTRACT

This thesis explores the challenges in diagnosing and treating diabetes using advanced technologies. Existing systems that employ machine learning and generative artificial intelligence in biomedical applications are presented. A machine learning model was created and a generative artificial intelligence service was added to analyze the entered data. A web system has been designed, allowing users to input medical data, assess the risk of diabetes, and receive recommendations for further treatment. The testing confirmed the correctness of the implemented functionalities and the satisfactory accuracy of the applied machine learning algorithms and generative artificial intelligence.

Keywords: diabetes, generative artificial intelligence, machine learning, web application.

ŽIVOTOPIS

Antonio Šušovček rođen je 1. kolovoza 2000. godine u Osijeku. Pohađao je Osnovnu Školu Vladimira Becića u Osijeku. Nakon toga upisao je i završio srednjoškolsko obrazovanje u III. Gimnaziji Osijek. 2019. godine upisuje Sveučilišni prijediplomski studij Računarstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija te ga uspješno završava 2022. godine. Iste godine upisuje sveučilišni diplomski studij, smjer Računarstvo, modul DRC - Programsko inženjerstvo na istoimenom fakultetu.

_____

Potpis autora

PRILOZI

Prilog 1. Diplomski rad u datoteci .docx

Prilog 2. Diplomski rad u datoteci .pdf

Prilog 3. Poslužiteljski programski kod web sustava

Prilog 4. Korisnički programski kod web sustava

Prilog 5. Razvoj i ispitivanje modela strojnog učenja