Web sustav za praćenje hitnih medicinskih stanja na temelju analize slike i generiranja prirodnog jezika

Česnek, Filip

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

Sveučilišni studij

WEB SUSTAV ZA PRAĆENJE HITNIH MEDICINSKIH STANJA NA TEMELJU ANALIZE SLIKE I GENERIRANJA PRIRODNOG JEZIKA

Diplomski rad

Filip Česnek

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 28.09.2020.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Filip Česnek
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-973R, 23.09.2019.
OIB studenta:	22974242059
Mentor:	Prof.dr.sc. Goran Martinović
Sumentor:	
Sumentor iz tvrtke:	Mario Filipović
Predsjednik Povjerenstva:	Prof.dr.sc. Željko Hocenski
Član Povjerenstva 1:	Prof.dr.sc. Goran Martinović
Član Povjerenstva 2:	Doc.dr.sc. Zdravko Krpić
Naslov diplomskog rada:	Web sustav za praćenje hitnih medicinskih stanja na temelju analize slike i generiranja prirodnog jezika
Znanstvena grana rada:	**Procesno računarstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U diplomskom radu potrebno je proučiti relevantnu literaturu i na temelju toga definirati funkcionalne i nefunkcionalne zahtjeve za razvoj web sustava za prepoznavanje hitnih medicinskih stanja (neurološka, psihička, drugo) na temelju analize slika, te generiranje uputa za postupanje pri tim stanjima koristeći načela generiranja prirodnog jezika. Također, potrebno je analizirati prikladne klasifikacijske postupke strojnog učenja, kao i modele i alate koji omogućuju njihovu ugradnju u navedeni web sustav s ciljem analize slike tijela i lica, te generiranja prirodnog jezika koristeći tehniku sažimanja. Uz to, treba razraditi arhitekturu web aplikacije na strani poslužitelja i korisnika, te opisati potrebne tehnologije za izradu programskog rješenja (Vue.js, Node.js, MongoDB, TensorFlow i drugi). U praktičnom dijelu rada navedeni web sustav treba programski ostvariti, ispitati i analizirati za odgovarajuće ispitne slučajeve, te za dovoljan broj medicinskih stanja i uputa za postupanje. Tema rezervirana za: Filip Česnek Sumentor iz tvrtke: Mario Filipović, Code Consulting
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)

Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	28.09.2020.

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

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 14.10.2020.

Ime i prezime studenta:	Filip Česnek
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-973R, 23.09.2019.
Turnitin podudaranje [%]:	11

Ovom izjavom izjavljujem da je rad pod nazivom**: Web sustav za praćenje hitnih medicinskih stanja na temelju analize slike i generiranja prirodnog jezika**

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

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Rapidni tehnološki napredak u području računalnog vida omogućio je izradu aplikacija i primjenu koncepata koji su mnogo složeniji od nekih klasičnih zadataka kao što je prepoznavanje činjenice da se neki objekt nalazi na slici. Nove primjene, od filtara lica koje koriste razne aplikacije društvenih mreža, korištenja računalnog vida u autonomnim automobilima za prepoznavanje objekata na cesti, pa sve do prepoznavanja identiteta osobe na slici čine računalni vid jednom od najatraktivnijih tehnologija. Osim primjetnog razvitka tehnologije u području računalnog vida, grana umjetne inteligencije koja se također razvija i napreduje jest obrada prirodnog jezika, koja se bavi raznim zadaćama poput prevođenja i sažimanja teksta, analize sentimenta, identifikacije jezika i drugima, no ova grana nailazi i na poteškoće zbog prirode ljudskog jezika i apstraktnih pravila koja se koriste pri govoru i pismu.

Cilj ovog diplomskog rada je proučiti jednu od potencijalnih primjena računalnog vida, te napraviti web aplikaciju koja će koristiti računalnu kameru za prepoznavanje preddefiniranih hitnih medicinskih slučajeva, te davanje uputa kako postupiti pri takvim stanjima tako da se detaljne upute sažmu koristeći metode obrade prirodnog jezika.

Drugo poglavlje prikazuje teorijsku podlogu iz područja strojnog učenja, računalnog vida i obrade prirodnog jezika koju ovaj rad koristi za ostvarenje praktičnog dijela zadatka. Treće poglavlje daje pregled postojećih tehnologija, metoda i komercijalnih rješenja slične onima koji se koriste u ovom diplomskom radu, te prikazuje idejno rješenje web sustava za praćenje hitnih medicinskih stanja na temelju analize slike i generiranja prirodnog jezika. Četvrto poglavlje je prikaz svih relevantnih tehnologija, programskih okvira, biblioteka i usluga koje su važne za ostvarenje web sustava. Peto poglavlje prikazuje cjelovito programsko rješenje, od njegove opće strukture, pa do detalja poput objašnjenja važnih funkcija i prikaza korisničkog sučelja. Šesto poglavlje daje rezultate funkcionalnog testiranja aplikacije, kao i rezultate ispitivanja istreniranog modela strojnog učenja za detekciju položaja tijela.

1.1. Zadatak diplomskog rada

U diplomskom radu potrebno je proučiti relevantnu literaturu i na temelju toga definirati funkcionalne i nefunkcionalne zahtjeve za razvoj web sustava za prepoznavanje hitnih medicinskih stanja (neurološka, psihička, drugo) na temelju analize slika, te generiranje uputa za postupanje pri tim stanjima koristeći načela generiranja prirodnog jezika. Također, potrebno je analizirati prikladne klasifikacijske postupke strojnog učenja, kao i modele i alate koji omogućuju

njihovu ugradnju u navedeni web sustav s ciljem analize slike tijela i lica, te generiranja prirodnog jezika koristeći tehniku sažimanja. Uz to, treba razraditi arhitekturu web aplikacije na strani poslužitelja i korisnika, te opisati potrebne tehnologije za izradu programskog rješenja (Vue.js, Node.js, MongoDB, TensorFlow i drugi). U praktičnom dijelu rada navedeni web sustav treba programski ostvariti, ispitati i analizirati za odgovarajuće ispitne slučajeve, te za dovoljan broj medicinskih stanja i uputa za postupanje.

2. STROJNO UČENJE I RAČUNALNI VID

Prema [1], pojam strojnog učenja odgovara na pitanje kako napisati računalne programe koji će poboljšavati svoje performanse u izvođenju nekog zadatka korištenjem prethodnog iskustva ili korištenjem primjera podataka. Strojno učenje je potrebno u slučajevima kada ne postoji način kako izravno napisati računalni program kako bi se riješio neki problem, nego su potrebni primjeri podataka ili iskustvo. Jedan primjer kada je strojno učenje potrebno je kad ljudska stručnost u nekom području ne postoji. Algoritmi strojnog učenja pokazali su se korisnima u različitim domenama primjene, od raspoznavanja uzoraka i dubinske analize podataka iz velikih baza podataka do računalnog vida, robotike, bioinformatike, računalne lingvistike i područja u kojima se računalni program mora dinamički prilagođavati promjenjivim uvjetima.

Računalni vid je prema [2] znanstvena i tehnološka disciplina, usko vezana s umjetnom inteligencijom, strojnim učenjem, fotografijom i optikom, koja se bavi teorijom i izradom sustava koji služe za ekstrakciju informacija iz slika, bila ona u obliku fotografija ili video isječaka. Cilj računalnog vida je prepoznavanje i praćenje objekata, detekcija unaprijed zadanih događaja, rekonstrukcija slike i sl. Računalni vid koristi se u proizvodnoj industriji za manipulaciju robotske ruke, za uočavanje zastoja i neispravnosti u proizvodnji, u medicini za pomoć liječnicima pri postavljanju dijagnoze, u prometnim svrhama za detekciju objekata kod autonomnih vozila, u svrhe sigurnosti u obliku nadzora zračnih luka, skeniranja lica i identifikaciju potencijalnih prijetnji, u vojne svrhe za navođenje raketa, očitavanje registracijskih pločica, čitanje bar koda, očitavanje otisaka prstiju, analiza krvi i mnoge druge.

2.1. Strojno učenje

Strojno učenje grana je umjetne inteligencije [3] koja se bavi oblikovanjem algoritama koji svoju učinkovitost poboljšavaju na temelju empirijskih podataka, tj. računalo se programira na način da se optimiziraju neki kriteriji uspješnosti na temelju podatkovnih primjera ili prethodnog iskustva. Strojno učenje koristi se za rješavanje složenih problema kod kojih ne postoji ljudsko znanje o procesu i koje nije moguće riješiti na klasičan algoritamski način (npr. raspoznavanje govora, analiza sentimenta teksta, regresijska analiza…). Koristi se i za otkrivanje znanja u velikim skupovima podataka (engl. *data mining*) i u sustavima u kojima je potrebna dinamička prilagodba.

Tri elementa na kojima se princip strojnog učenja temelji su podaci, algoritam i model. Podaci strojnog učenja mogu biti različitih oblika: slike, videozapisi, zvučni zapisi, tekstualni podaci, brojčani podaci, ili neka kombinacija različitih oblika. Podaci mogu biti strukturirani ili

nestrukturirani. Strukturirani podaci su oni koji odgovaraju formalno definiranom podatkovnom modelu koji određuje kako podaci trebaju izgledati, koje operacije se nad njima mogu izvršavati te koja pravila podaci moraju zadovoljavati. Takvi podaci imaju smisleni smještaj unutar neke strukture, kao što je tablica u relacijskoj bazi podataka, i najčešće su tekstualnog oblika. Nestrukturirani podaci su oni koji imaju neku unutarnju strukturu, no ne prate nikakav podatkovni model, te se uglavnom ne mogu spremati u relacijsku bazu podataka. Nestrukturirani podaci mogu biti različitih oblika: od slika, video i zvučnih zapisa, email poruka, do podataka s društvenih mreža, nadzornih zapisa i drugih oblika podataka. Kao takvi, puno su češći od strukturiranih, te prema [4] čine i do 80% prikupljenih podataka.

Algoritmi strojnog učenja su oni koji su sposobni iterativno učiti iz podataka kako bi opisali te podatke ili predvidjeli neke ishode na temelju novih podataka. Na osnovu ulaznih podataka algoritmi oblikuju i kao rezultat daju određeni model. Proces oblikovanja modela naziva se treniranje.

Cilj algoritama jest izgraditi modele s dobrim svojstvima (engl. *features*) koji će moći, na temelju naučenih podataka, uspješno predvidjeti ishode na temelju nepoznatih podataka, tj. izgraditi modele s dobrim svojstvom generalizacije. Stoga je nakon treniranja modela važno evaluirati model, te napraviti promjene u treniranju modela koje će poboljšati performanse modela.

Pri treniranju modela bitno je obratiti pažnju i na hiperparametre algoritma. Hiperparametri su postavke algoritma koje se koriste za kontrolu procesa učenja. U ove parametre spadaju veličina serije (engl. *batch size*), broj epoha, veličina validacijskog skupa (engl. *validation set*), stopa učenja (engl. *learning rate*), broj slojeva i aktivacijska funkcija (ako se radi o neuronskim mrežama) i drugi.

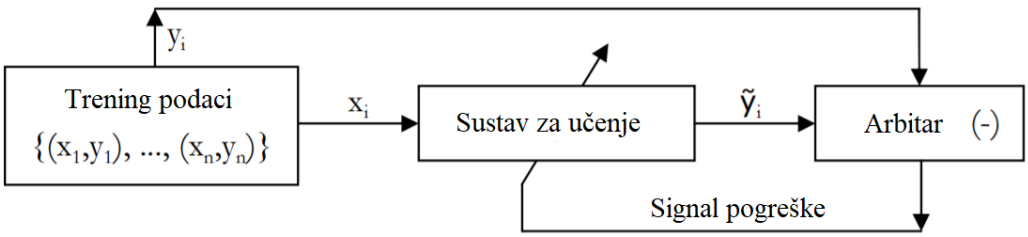
Strojno učenje može se podijeliti na nadzirano (engl. *supervised learning*), polunadzirano (engl. *semisupervised learning*), nenadzirano (engl. *unsupervised learning*) i podržano ili ojačano učenje (engl. *reinforcement learning*).

Prema [5], nadzirano učenje je paradigma strojnog učenja za dobivanje informacija o vezi između ulaza i izlaza sustava, uz zadan skup trening podataka, gdje je svakoj ulaznoj varijabli pridružena izlazna varijabla koja se naziva oznaka.

Cilj nadziranog učenja je izgraditi umjetni sustav koji može naučiti mapirati ulazne i izlazne podatke, te na temelju toga može predvidjeti izlaz sustava uz nove ulazne podatke. Ako izlaz uzima konačan skup diskretnih vrijednosti koje označavaju oznaku klase ulaznih podataka, naučeno

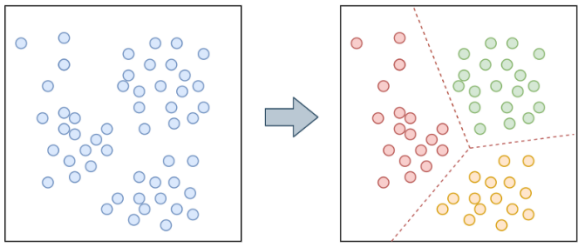
mapiranje naziva se klasifikacija ulaznih podataka. Ako izlaz uzima kontinuirane vrijednosti, tada se govori o regresiji. Informacije o vezi između ulaza i izlaza predstavljena je parametrima modela učenja. Kada ti parametri nisu izravno dostupni iz skupa trening podataka, sustav za učenje mora proći kroz proces procjene kako bi se prikupili potrebni parametri.

Slika 2.1 [5] prikazuje blok dijagram koji prikazuje način nadziranog učenja. U ovom dijagramu, (x_i, y_i) je uzorak iz skupa trening podataka, gdje *x* predstavlja ulaz sustava, *y* predstavlja izlaz sustava (npr. oznaku ulaznog podatka *x*), a *i* je indeks uzorka. Tijekom procesa nadziranog učenja, uzorak iz skupa trening podataka x_i šalje se sustavu za učenje koji generira izlaz $\tilde{y}_i$. Taj se izlaz uspoređuje sa stvarnom oznakom y_i pomoću arbitara koji računa razliku između ta dva izlaza. Razlika, označena kao signal pogreške, šalje se nazad sustavu za učenje kako bi se podesili parametri sustava. Cilj ovog procesa učenja je dobivanje skupa optimalnih parametara sustava za učenje, gdje su razlike između generiranog izlaza $\tilde{y}_i$ i stvarnog, označenog izlaza y_i minimalne.



Slika 2.1. *Blok dijagram koji prikazuje način nadziranog učenja [5]*

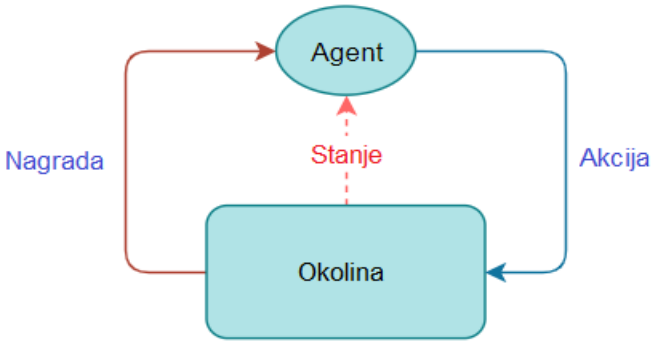
Prema [6], nenadzirano učenje koristi podatke bez označene ciljne vrijednosti u čijoj strukturi postoje pravilnosti na način da se određeni parametri pojavljuju češće od drugih. Cilj nenadziranog učenja je za skup ulaza organizirati podatke ili opisati njihovu strukturu. Najčešće tehnike nenadziranog učenja su grupiranje ili klasteriranje (engl. *clustering*), koje se koristi za pronalaženje skrivenih uzoraka ili grupa u podacima i koje je prikazano na slici 2.2 po uzoru na [7], detekcija iznimaka unutar skupa (engl. *outlier detection*), asocijacija kojom se pokušavaju pronaći pravila koja opisuju velike dijelove podataka, te smanjenje dimenzionalnosti podataka.



Slika 2.2. *Grupiranje ili klasteriranje podataka [7]*

Prema [8], podržano ili ojačano učenje je vrsta strojnog učenja u kojoj agent uči iz interakcije s okolinom. Učenje se obavlja na principu pokušaja i pogreške. Agent u nekoj situaciji u određenom stanju izvodi akciju za koju dobiva nagradu ili kaznu. Sekvencijalnim izvođenjem akcija agent nastoji doći do nekog konačnog cilja.

Proces učenja sastoji se od 3 komponente: agenta, okoline i zadatka agenta kao što je prikazano na slici 2.3 [8]. Agentom se smatra bilo kakav samostalni entitet koji je u stanju percipirati stanje okoline i izvoditi određeni skup akcija. Okolina je svijet koji okružuje agenta. Okolina se promatra kao (konačan ili beskonačan) skup stanja zajedno s pravilima kako akcije koje agent izvodi djeluju na promjenu stanja i kako agent biva nagrađen za izvođenje akcija. Okolina je promjenjiva u vremenu, tj. izmjenu stanja može uzrokovati i protjecanje vremena. Zadatak agenta je da usvoji određeno znanje o okolini. Usvajanje znanja agent postiže izvođenjem onih akcija za koje dobiva maksimalnu nagradu od okoline.

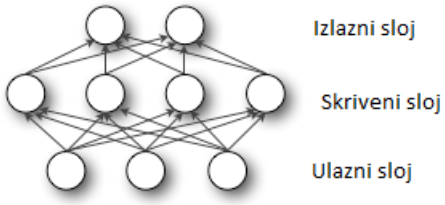


Slika 2.3. *Proces podržanog učenja [8]*

2.2. Neuronske mreže

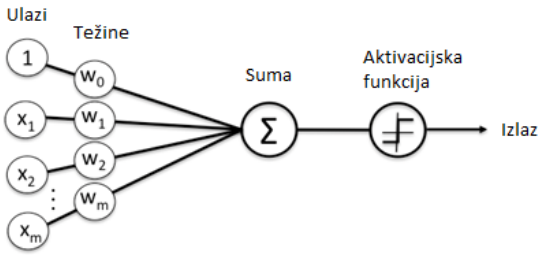
Neuronske mreže su oblik strojnog učenja koje se zasnivaju na principu na kojem ljudski mozak obrađuje informacije i uči [9]. Neuronska mreža je, u suštini, skup međusobno povezanih slojeva, najčešće tri ili više njih, kako je i prikazano na slici 2.4. Ulazi su prvi sloj, i povezani su na izlazni

sloj preko acikličkog grafa koji se sastoji od čvorova i veza između njih, od kojih svaka ima određeni težinski faktor. Između ulaznih i izlaznih slojeva nalazi se jedan ili više skrivenih slojeva, u kojima umjetni neuroni uzimaju u obzir ulaze sa svojim težinskim faktorom i proizvedu izlaz kroz neku aktivacijsku funkciju, koja se odabire ovisno i mreži i njenoj primjeni (*step* funkcija, linearna funkcija, sigmoid, tanh funkcija, ReLu funkcija...). Broj skrivenih slojeva i neurona koji se nalaze u njima definiraju složenost i performanse modela, stoga je ove parametre potrebno regulirati pri treniranju modela.



Slika 2.4. *Struktura neuronske mreže*

Treniranje neuronske mreže odvija se na način da mreža uzima ulazne podatke zajedno s odgovarajućim izlazima, zatim ih šalje kroz svaki neuron mreže. Svaki od neurona na određen način preoblikuje ulaz kojeg šalje sljedećem sloju. Dobiveni izlaz neurona je težinski faktor čvora prema kojem se odlučuje hoće li se neki neuron aktivirati ili ne. Kroz učenje, mijenjaju se težinski faktori na vezama s ciljem smanjivanja pogreške na izlaznom sloju koristeći razne algoritme za optimizaciju (npr. gradijentni spust, Adagrad, RMSprop, Adam, LBFGS) koji za cilj imaju što više približiti vrijednosti očekivanih i proizvedenih izlaza. Taj se proces ponavlja veći broj puta, te na taj način model uči. Slika 2.5 prikazuje strukturu neurona. Neuron za svaku od ulaznih veza korigira ulaznu vrijednost na način da množi ulaz s težinom veze, zatim sumira sve korigirane ulazne vrijednosti, te dobivenu sumu prosljeđuje kroz aktivacijsku funkciju koja definira izlaz neurona.



Slika 2.5. *Model umjetnog neurona*

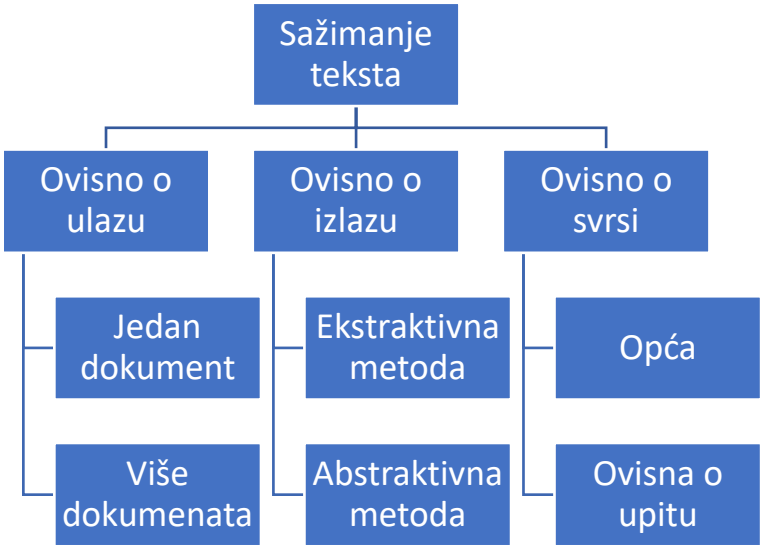
2.3. Obrada prirodnog jezika

Prema [10], obrada prirodnog jezika (engl. *Natural language processing*) je grana umjetne inteligencije koja se bavi sposobnošću strojeva da razumiju i interpretiraju ljudski govor ili pismo.

Cilj obrade prirodnog jezika je učiniti interakciju između čovjeka i računala laganom i efikasnom. Računalo se uči sintaksi i značenju ljudskog jezika kako bi bilo u mogućnosti obraditi ljudski jezik i dati adekvatan izlaz korisniku.

Prema [11], koraci koji se poduzimaju pri obradi prirodnog jezika uključuju leksičku analizu kojom se identificira i analizira struktura riječi, sintaktičku analizu kojom se vrši analiza struktura riječi u rečenici, semantičku analizu kojom se identificira značenja riječi, integraciju diskursa za analizu značenja rečenice u odnosu na prethodnu rečenicu, pragmatičku analizu za ponovnu interpretaciju onoga što se reklo kako bi se identificiralo ono što se točno mislilo reći.

Jedna od primjena obrade prirodnog jezika relevantna za ovaj rad je sažimanje teksta (engl. *text summarization*). Sažimanje teksta je proces reduciranja velike količine teksta u kratak sažetak koji će čitatelju omogućiti pregled najbitnijih stavki teksta i na taj način uštedjeti vrijeme čitatelja. Razlog ove primjene je ogromna količina podataka na internetu zbog koje se razvila želja da se smanji vrijeme potrebno za čitanje i ubrza postupak pretrage informacije, uz maksimiziranje količine informacija u sažetom tekstu. Na slici 2.6 napravljenoj po uzoru na [12], vidljiva je podjela procesa sažimanja teksta u tri kategorije.

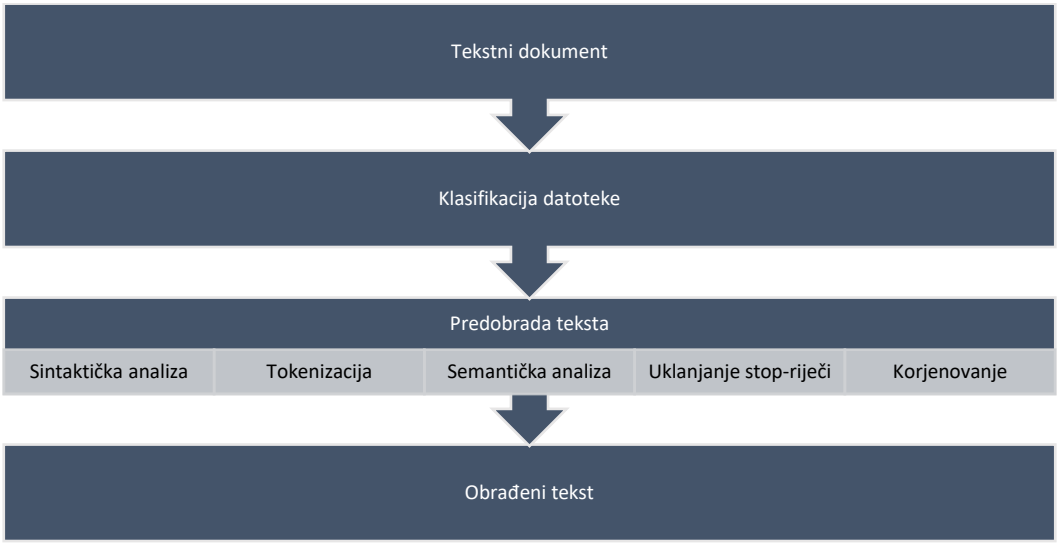


Slika 2.6. *Podjela procesa sažimanja teksta [12]*

Prva kategorija uzima u obzir broj ulaza, tj. broj dokumenata. Može se uzeti jedan dokument ili više dokumenata iz kojih će se tekst sažeti u jedan izlazni dokument. Druga kategorija uzima u obzir svrhu sažimanja teksta, koja može biti opća ili može biti ovisna o upitu na temelju kojeg će se tekst sažeti. Zadnja stavka koju treba uzeti u obzir su metode sažimanja teksta, koje se izabiru ovisno o željenom izlazu. Ekstraktivne metode rade tako da izabiru podskup riječi, fraza i rečenica prisutan u dokumentu i kao izlaz daju sažetak teksta koristeći izabrane stavke podskupa. Kod

apstraktivnih metoda algoritam izgrađuje unutarnji semantički prikaz i zatim koristi tehnike generiranja prirodnog jezika, tj. pri ovoj metodi računalo radi na način sličnom ljudskom mozgu i ima mogućnost stvaranja sažetka razumijevajući tekst prisutan u dokumentu. Ovaj način rezultira sažetkom koji je puno bliže onome što bi i čovjek napisao, te za razliku od ekstraktivne metode ovom se metodom mogu generirati i riječi i fraze koji nisu prisutni u originalnom tekstu.

Algoritmi za sažimanje teksta evaluiraju rečenice na temelju nekog osnovnog kriterija ili značajki i rečenice u izlaznom tekstu su poredane u istom redoslijedu kao i kod originalnog teksta, osim što se kod ekstraktivnih metoda prolazi kroz dodatni korak uređivanja teksta na gramatički točan način kako bi izlazni dokument bio što sličniji ljudskom radu. Algoritmi se implementiraju u dvije faze: predobrada teksta i ekstrakcija informacija. Na slici 2.7 je prikazan postupak predobrade teksta, koja se dijeli u pet koraka: sintaktička analiza, tokenizacija, semantička analiza, uklanjanje zaustavnih ili stop-riječi i korjenovanje.



Slika 2.7. *Postupak predobrade teksta*

Sintaktička analiza podrazumijeva detektiranje početka i kraja svake rečenice u ulaznom dokumentu. U gramatički strukturiranim dokumentima algoritam može uzimati simbol točke kao početak i završetak rečenice, no kod gramatički neispravnih dokumenata i kod pisama kao što je kinesko potrebne su kompleksnije metode.

Tokenizacija je postupak razlamanja rečenice dobivene iz sintaktičke analize u tokene koji mogu biti riječi, brojevi ili interpunkcijski znakovi.

Semantička analiza je faza u kojoj se svakoj riječi u rečenici dodjeljuje njena uloga. Svaka riječ označava se kao imenica, glagol, pridjev, prilog itd. Proces dodjeljivanja uloge i raspoređivanja riječi u različite klase naziva se označavanje dijela govora (engl. *part-of-speech tagging*).

Zaustavne riječi (engl. *stop-words*) su riječi koje imaju sintaksnu ulogu, ali nemaju značenja (kao npr. engleske riječi *the, is, at, which, on*), ili su to riječi koje se često pojavljuju u tekstu, ali njihova vrijednost kod određivanja značenja je vrlo mala kada razmatramo cijelu rečenicu, te se te riječi filtriraju pri procesuiranju teksta.

Korjenovanje (engl. *stemming*) je zadatak koji se bavi evaluacijom osnovnog oblika neke riječi u ulaznom dokumentu. Iste riječi mogu biti napisane u drugim glagolskim vremenima ili proći kroz bilo koju promjenu, no u svakom će slučaju imati isto značenje. Stoga se radi korjenovanje kojim se riječi različitih oblika, ali istog značenja pretvaraju u osnovni oblik.

Druga faza algoritma je ekstrakcija ili izvlačenje informacija (engl. *information extraction*). To je proces automatskog izvlačenja i pretvorbe nestrukturiranih informacija iz teksta u strukturirane koje se spremaju u bazu podataka.

2.4. Računalni vid

Prema [2], računalni vid je multidisciplinarno područje računarstva, koje se općenito može kategorizirati kao potpodručje umjetne inteligencije i strojnog učenja, usredotočeno na repliciranju dijelova kompleksnosti ljudskog sustava vida i na omogućavanju računalima da identificiraju i procesuiraju objekte na slikama ili videima na isti način kao i ljudi. Donedavno, računalni vid je bio relativno ograničeno područje, no zahvaljujući napretku u umjetnoj inteligenciji i inovacijama u dubokom učenju i neuronskim mrežama, područje je znatno napredovalo, te je u nekim slučajevima i nadmašilo mogućnosti ljudi u zadacima vezanima za detektiranje i označavanje objekata.

Prema [13], primjena računalnog vida može se podijeliti u nekoliko skupina:

- Klasifikacija objekta: U koju opću kategoriju spada objekt na slici
- Identifikacija objekta: Što se nalazi na slici
- Verifikacija objekta: Nalazi li se objekt na slici
- Detekcija objekta: Gdje se nalazi objekt na slici
- Detekcija značajki objekta: Koje su ključne točke objekta na slici
- Segmentacija objekta: Koji pikseli pripadaju objektu na slici
- Prepoznavanje objekta: Koji objekti su na slici i gdje se nalaze

- Prikupljanje informacija: Uz zadani objekt pronaći slike na kojima se objekt nalazi

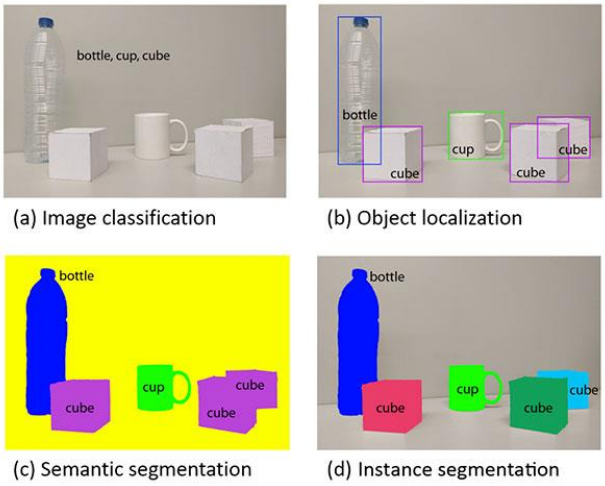
Konkretno, računalni vid može se primijeniti za medicinske svrhe, u obliku dijagnostičkih aparata kao što su mikroskopi, ultrazvuk i x-zrake, u industriji za pozicioniranje robotskih ruku, u prometu kao dio autonomnih vozila gdje se koristi prepoznavanje znakova i objekata na cesti, za sigurnost u obliku nadgledanja osoba i identificiranja potencijalnih prijetnji, te u vojne svrhe kao što je navođenje raketa.

2.5. Prepoznavanje položaja tijela

U računalnom vidu i robotici, tipičan zadatak je identifikacija specifičnih objekata na slici i određivanje pozicije i orijentacije svakog objekta u nekom koordinatnom sustavu. Dobivena informacija može se iskoristiti za npr. manipuliranje objekta robotskom rukom ili izbjegavanje naleta na objekt. Ova kombinacija položaja i orijentacije naziva se poza objekta.

Slikovni podaci iz kojih se može odrediti poza objekta može biti jedna slika, par stereo slika ili sekvenca slika u kojoj se kamera pomiče poznatom brzinom. Razmatrani objekti mogu biti neki opći predmeti ili mogu biti specifičniji, kao što je tijelo ili dijelovi tijela čovjeka kao npr. ruke ili lice. Metode koje se koriste za određivanje poze objekta su uglavnom specifične za određene klase objekata i neće biti točne za druge klase.

Prema [14], jedna od novijih mogućnosti primjene je kombinacija više zadataka, kao što je detekcija većeg broja osoba, 2D estimacija poze i segmentacija na razini primjerka (engl. *instance segmentation*). Ti se koncepti mogu iskoristiti za zadatke kao što su identificiranje pojave svake osobe na slici, lokalizacija ključnih točaka na licu i tijelu i estimacija maske segmentacije na razini primjerka (engl. *instance segmentation mask*), koji bi se mogli koristiti u aplikacijama kao što je pametno uređivanje slike, prepoznavanje osoba i aktivnosti, virtualna ili proširena stvarnost i robotika. Neke od mogućnosti robotskog vida, kao što je klasifikacija slike, lokalizacija objekta, semantička segmentacija i segmentacija na razini primjerka prikazane su na slici 2.8, prema [14].



Slika 2.8. *Mogućnosti robotskog vida [14]*

Segmentacija je postupak pridruživanja kategorije svakom pikselu slike. Tako će se, za sliku iz prometne scene, segmentacijom odrediti maske koje određuju piksele koji predstavljaju cestu, nebo, okolinu i ostale prometne sudionike. Kategorije kao što su cesta, nebo i vegetacija tipično unutar slike nemaju više instanci, no kod drugih faktora, kao što su različita vozila, pješaci i biciklisti, postoje višestruke instance, i svaka je od tih instanci važna i za svaku je važno odrediti masku. Prikaz detekcije i segmentacije osoba nalazi se na slici 2.9, prema [15].

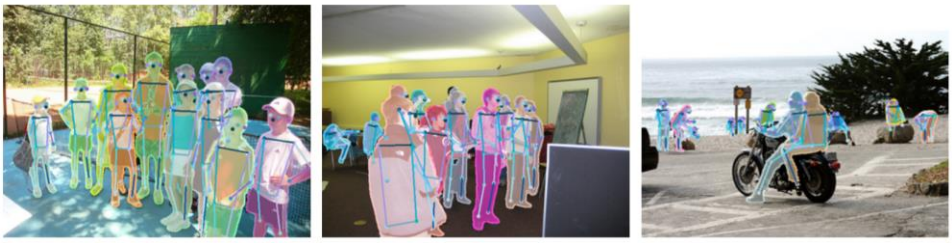
Slika 2.9. *Detekcija i segmentacija osoba [15]*

Prema [16], postoje dva glavna pristupa za rješavanje problema detekcije više osoba, estimacije poze i segmentacije.

- Pristup odozgo prema dolje (engl. *top-down approach*) dijeli složeni problem ili algoritam na više manjih dijelova (modula). Ovi moduli se dalje dekomponiraju sve dok se rezultirajući modul ne pretvori u temeljni program koji ne može dalje dekomponirati. U kontekstu detekcije osoba, ovaj pristup počinje identifikacijom i lokalizacijom individualne instance osobe pomoću detektora objekta koji oko detektirane osobe odredi okno koje ga uokviruje (engl. *bounding box*), nakon čega slijedi estimacija poze jedne osobe ili binarna segmentacija pozadine i objekta u fokusu u uokvirenom području.

- Pristup odozdo prema gore (engl. *bottom-up approach*) uključuje projektiranje najosnovnijih dijelova koji se zatim kombiniraju kako bi zajedno činili modul više razine. Ova integracija podmodula i modula u jedan modul više razine se ponavlja sve dok se ne dobije potreban potpuni algoritam. Kod detekcije osoba, ovaj pristup počinje lokalizacijom semantičkih entiteta (predodređene individualne ključne točke ili semantičke oznake segmentacije osobe) koji se grupiraju u instance osoba.

Na slici 2.10 preuzetoj iz [16], prikazana je estimacija poze na slikama na kojima se nalazi više osoba.



Slika 2.10. *Detekcija osoba i estimacija poza za više osoba [16]*

3. PREGLED POSTOJEĆIH RJEŠENJA I IDEJNO RJEŠENJE VLASTITOG SUSTAVA

Prije izrade vlastitog sustava za detektiranje položaja tijela i generiranja prirodnog jezika podataka, dat će se prikaz postojećih rješenja u tom području, kako u obliku gotovih komercijalnih proizvoda, tako i u obliku znanstvenih metoda. Na kraju poglavlja dat će se prijedlog idejnog programskog rješenja koje će rješavati problem detekcije položaja tijela osobe na slici, klasificiranja položaja tijela kao medicinsko stanje, te generiranja teksta koji će predstavljati opis medicinskog stanja, kao i upute kako postupiti u tom slučaju.

3.1. Pregled postojećih metoda i tehnologija za sažimanje teksta

3.1.1. Pegasus

Pegasus (*Pre-training with Extracted Gap-sentences for Abstractive SUmmarization Sequence-to-sequence*) je sustav kojeg su razvili istraživački tim *Google Brain* u suradnji s timom s Imperijalnog koledža u Londonu. Sustav koristi arhitekturu *Google Transformer* neuronske mreže u kombinaciji s preddefiniranim zadacima namijenjenima za apstraktivno generiranje teksta. Prema rezultatima njihovog istraživanja [17], sustav Pegasus postiže značajne rezultate u 12 zadataka sažimanja, koji uključuju sažimanje vijesti, znanstvenih članaka, priča, uputa, emailova, patenata, prijedloga zakona i sl. Kako sustav koristi apstraktivnu metodu sažimanja, sažetak teksta kojeg generira model može sadržavati nove riječi i pokriti sve važne informacije, a da tekst ostane lingvistički tečan.

Google Transformer je vrsta neuronske arhitekture koju su u znanstvenom radu uveli istraživači iz Googleova odjela za istraživanje umjetne inteligencije *Google Brain,* a koristi se primarno kao mreža za obradu prirodnog jezika, poglavito u Googleovom proizvodu *Google Translate*. Kao i sve duboke neuronske mreže, ona sadrži funkcije (neurone) poredane u međusobno povezane slojeve koji prenose signale iz ulaznih podataka i polako prilagođavaju sinaptičku snagu (težine) svake veze - tako svi AI modeli izvlače značajke i uče kako predviđati nove slučajeve, ali *Transformer,* jedinstveno za tu mrežu, ima i parametar koji predstavlja jedinicu pažnje (engl. *attention units*), koja prikazuje koliko je određena riječ u rečenici relevantna s drugim riječima u istoj rečenici. Svaki je izlazni element povezan sa svakim ulaznim elementom i težinski faktori između njih izračunavaju se dinamički.

Tim koji je radio na *Pegasus* modelu osmislio je zadatak za treniranje modela u kojem su cijele, važne rečenice u tekstu bile prikrivene. Model je zatim morao ispuniti sve praznine u tekstu

koristeći internetske i novinske članke, uključujući i one sadržane u novom korpusu pod nazivom *HugeNews* kojeg su istražitelji sastavili.

U eksperimentima, tim je izabrao *Pegasus* model s najboljim performansama – onaj s 568 milijuna parametara, koji je istreniran na ili 750 GB teksta izvučenog iz preko 350 milijuna web stranica, ili na *HugeNews* korpusu, koji sadrži 1.5 milijardu članaka, ukupne veličine 3.8 TB, koji su skupljeni iz novinskih web stranica. Primjer sažetog teksta modelom *Pegasus* dan je na slici 3.1.

	XSum
Document (ID #198)	Media playback is not supported on this device Craig Cathcart put the visitors ahead before substitute Simon Church won and scored an 89th-minute penalty. "There were lots of positives out of it even if we'd have come off and lost 1-0. They had a good mentality and attitude," said Coleman. Wales face another Euro 2016 warm-up game against Ukraine in Kiev on Monday. "We look forward to our next challenge now," added Coleman. "The team will change up again, and we'll see how they go again." Striker Church, currently on loan at Scottish Premiership side Aberdeen from Reading, was delighted with his equaliser from the spot. "Northern Ireland were a tough side to play against. They've obviously done well to get where they are and it was a tough game," he said. "We wanted to do well because it was the last time a Wales crowd would see us before the Euros and we wanted to put in a good performance. "I've just got to keep going now and hopefully score some goals. This is a great squad to be part of."
Gold	Wales manager Chris Coleman said he was pleased with his team's performance after they came from behind to draw 1-1 with Northern Ireland in Cardiff.
Model	Wales manager Chris Coleman praised his side's attitude after they came from behind to draw 1-1 with Northern Ireland in Cardiff.

Slika 3.1. *Tekst sažet Pegasus modelom [17]*

3.1.2. MeaningCloud Text Summary

Prema [18], MeaningCloud je SaaS (Software as a Service) proizvod koji omogućuje korisnicima ugradnju tekstualne analitike ili semantičkog procesuiranja u aplikaciju. MeaningCloud proširuje koncept semantičkog API-ja sa programskim okvirom baziranom u oblaku računala, što čini integraciju semantičke obrade teksta vrlo jednostavnim procesom. Neke od usluga koje proizvod MeaningCloud pruža su:

- Duboka kategorizacija (engl. *deep categorization*), koja omogućuje dodjeljivanje jedne ili više kategorija nekom tekstu
- Ekstrakcija tema (engl. *topics extraction*), koja omogućuje izvlačenje relevantnih informacija kao što su imena entiteta, koncepata i činjenica iz teksta
- Analiza sentimenta, koja omogućuje analizu teksta i detekciju subjektivnosti, ironije i emocije sadržane u tekstu
- Identifikacija jezika
- Sažimanje teksta

3.2. Pregled postojećih metoda za detekciju položaja tijela

3.2.1. DeepPose

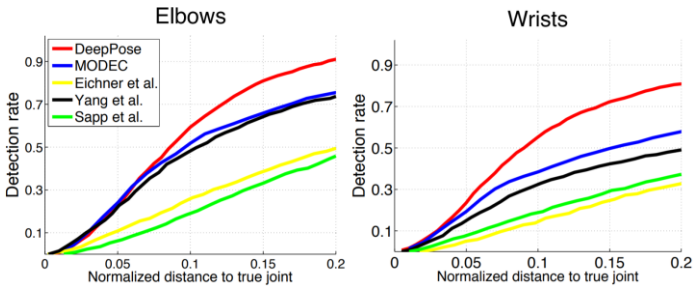
Prema [19], *DeepPose* je model za estimaciju položaja tijela koji se temelji na korištenju dubokih neuronskih mreža (DNN). Autori modela su Alexander Toshev i Christian Szegedy iz *Googlea,* koji u svojem znanstvenom radu rješavaju problem estimacije položaja tijela regresijom, kojom se pronalaze različiti zglobovi na tijelu osobe koji se predstavljaju x i y koordinatama.

Korištena neuronska mreža sastoji se od sedam slojeva, koji uključuju konvolucijski sloj, sloj za sažimanje (engl. *pooling layer*), sloj za normalizaciju lokalnog odgovora (engl. *local response normalization layer*), te potpuno povezani sloj (engl. *fully connected layer*). Konvolucijski sloj i potpuno povezani sloj su jedini slojevi sa parametrima koji se mogu učiti, te sadrže linearne transformacije koje prati ispravljena linearna jedinica (engl. *rectified linear unit, ReLU*).

Na slici 3.2 [19] je prikazan postupak regresije položaja tijela temeljen na dubokoj neuronskoj mreži. Slojevi mreže vizualizirani su s pripadajućim dimenzijama, gdje su konvolucijski slojevi obojani plavo, a potpuno povezani slojevi zeleno. Slojevi bez parametara za učenje nisu prikazani. Nakon inicijalnog stadija kojim se detektira poza osobe, izlazni parametri šalju se ponovno na početak ciklusa radi boljeg i točnijeg definiranja položaja tijela.

Slika 3.2. *Postupak regresije položaja tijela temeljen na dubokoj neuronskoj mreži [19]*

Na slici 3.3 preuzetoj iz [19], prikazana je usporedba performansi *DeepPose* modela s drugim relevantnim modelima, kao što su MODEC (*Multimodal Decomposable Models for Human Pose Estimation*), te istraživanja drugih znanstvenika.



Slika 3.3. *Usporedba performansi DeepPose modela [19]*

3.2.2. OpenPose

Prema [20], OpenPose je sustav otvorenog koda za otkrivanje 2D poza više osoba u stvarnom vremenu, uključujući ključne točke tijela, stopala, šake i lica. Ovaj rad predlaže pristup u stvarnom vremenu za otkrivanje 2D ljudskih poza na slikama i video zapisima. Predložena metoda u radu koristi neparametrijske prikaze poznate kao polja sklonosti dijelova (engl. *Part Affinity Fields,*

PAF), koja su skup 2D vektorskih polja koji sadrže informacije o lokaciji i orijentaciji uda tijela na slici.

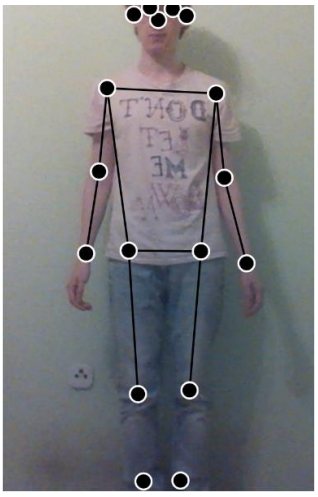
Na slici 3.4, preuzetoj iz [20] prikazana je estimacija položaja tijela za više osoba na slici. Na gornjem dijelu slike prikazana je povezanost različitih dijelova tijela koji pripadaju jednoj osobi. U donjem lijevom dijelu slike prikazana su polja sklonosti dijelova za svaku osobu na slici za desnu podlakticu svake osobe. Različitim bojama kodirana je informacija o orijentaciji uda. U donjem desnom dijelu slike prikazani su različiti vektori koji sadrže informacije o lokaciji i orijentaciji uda.

Slika 3.4. *Estimacija položaja tijela za više osoba [20]*

OpenPose je također programsko rješenje i programsko sučelje (engl. *Application Programming Interface*, *API*) koje ima mogućnost dohvaćanja slika iz različitih izvora. Na primjer, može se odabrati ulaz kao snimak kamere, kao video ili kao slika. Može se pokrenuti na različitim platformama kao što su Ubuntu, Windows, Mac OS X i ugradbeni sustavi (npr. Nvidia Tegra TX2). Također pruža podršku za različiti hardver, poput CUDA grafičke procesorske jedinice (engl. *Graphics Processing Unit*, *GPU*), OpenCL GPU-a i uređaja koji imaju samo centralnu procesorsku jedinicu (engl. *Central Processing Unit*, *CPU*).

3.2.3. PoseNet

Prema [21], PoseNet je istrenirani model u sklopu Tensorflow biblioteke koji se može koristiti za estimaciju poze osobe na slici ili video zapisu procjenjujući mjesta ključnih dijelova tijela (engl. *key body joints*) kao što je prikazano na slici 3.5 [21].



Slika 3.5. *Estimiranje mjesta ključnih dijelova tijela i poze osobe [21]*

Prema [22], TensorFlow je biblioteka otvorenog koda prvenstveno korištena za strojno i duboko učenje. Razvili su je istraživači i inženjeri iz Google Brain tima u Googleovoj AI organizaciji 2015. godine koji su istraživali strojno učenje i neuronske mreže. Tensorfow ima sadržajan, fleksibilan ekosustav alata, biblioteka i ostalih resursa koje pruža TensorFlow zajednica koja omogućava istraživačima istražiti nove koncepte strojnog učenja, a programerima laku izradu aplikacija koje koriste strojno učenje. TensorFlow ima API-je dostupne u nekoliko programskih jezika. Python API je najkompletniji i najjednostavniji za korištenje, no postoji podrška i za JavaScript, C++, Java, Go, Swift i druge programske jezike.

Ključne točke koje se detektiraju nalaze se u tablici 3.1, a za svaku postoji broj koji se kreće od 0.0 do 1.0 koji označava uvjerenost u točnost detekcije (engl. *confidence score*)

Tablica 3.1. *Ključne točke PoseNet modela*

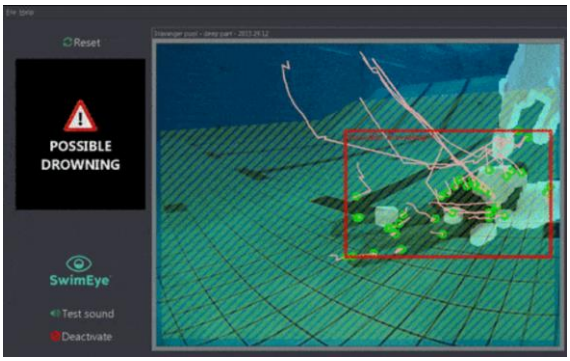
ID	KLJUČNA TOČKA	
0	nose	nos
1	leftEye	lijevo oko
2	rightEye	desno oko
3	leftEar	lijevo uho
4	rightEar	desno uho
5	leftShoulder	lijevo rame
6	rightShoulder	desno rame
7	leftElbow	lijevi lakat
8	rightElbow	desni lakat
9	leftWrist	lijevo zapešće
10	rightWrist	desno zapešće
11	leftHip	lijevi zglob kuka
12	rightHip	desno zglob kuka
13	leftKnee	lijevo koljeno
14	rightKnee	desno koljeno
15	leftAnkle	lijevi gležanj
16	rightAnkle	desni gležanj

3.3. Pregled postojećih komercijalnih rješenja za detekciju položaja tijela

Iako postoje različita rješenja za detekciju položaja tijela, ona su uglavnom implementirana kao komercijalni proizvodi u obliku dodatne opreme automobila, uređaja i opreme za virtualnu stvarnost, te samostalni sustavi za detekciju specifičnih događaja, kao što je utapanje osobe. Rješenja navedena u sljedećim potpoglavljima koriste metode računalnog vida i analize slike za detekciju određenih događaja na slici.

3.3.1. SwimEye

SwimEye [23] je autonomni sustav računalnog vida koji koristi kamere i računala za detekciju slučaja utapanja u bazenima. Sustav koristi višestruke podvodne kamere koji nadziru i prate radnje u ponašanje ljudi u bazenu. Informacije prikupljene kamerama šalju se stanici za nadzor i kontrolu, te ih sustav za prepoznavanje objekata analizira. Kada SwimEye detektira osobu u opasnosti, aktivira alarm koji obavijesti spasioce s lokacijom incidenta, te se na taj način smanjuju slučajevi utapanja, kao što je prikazano na slici 3.6 [23].



Slika 3.6. *Detekcija potencijalnog utapanja koristeći SwimEye sustav [23]*

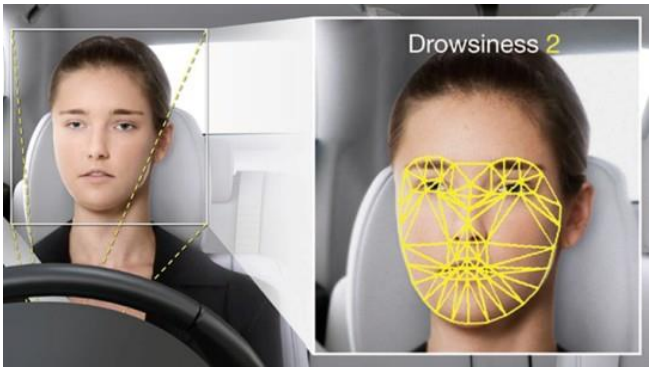
3.3.2. CORAL MANTA

CORAL MANTA [24] je sustav za detekciju utapanja, prikazan na slici 3.7, koji nije namijenjen uporabi u javnim bazenima, nego je namijenjen privatnim bazenima. Slično kao i kod SwimEye sustava, uz AI tehnologiju za praćenje osoba u bazenu i detektiranje potencijalnih slučajeva utapanja koriste se podvodne kamere koje detektiraju pokrete osoba u bazenu. Sustav je pogonjen solarnom energijom, tako da je raspoloživ cijelo vrijeme. U slučaju detektiranja slučaja utapanja, oglašava se alarm, te se šalju obavijesti članovima kućanstva preko aplikacije na pametnom telefonu ili tabletu.

Slika 3.7. *Sustav za detekciju utapanja CORAL MANTA [24]*

3.3.3. Detekcija pospanosti u automobilima

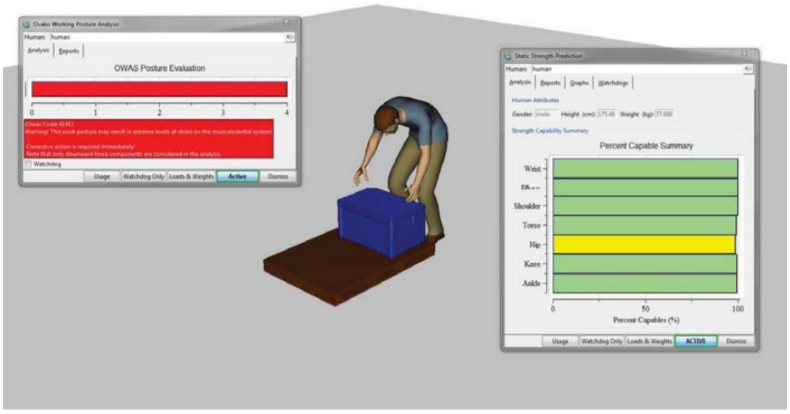
Prema [25], svake godine mnogi ljudi izgube živote u fatalnim prometnim nesrećama diljem svijeta i pospanost u vožnji je jedan od glavnih razloga stradavanja. Umor i mikro san kod vozača su često korijenski uzrok ozbiljnih nesreća. Međutim, inicijalni znakovi umora mogu se detektirati prije nastanka kritične situacije, pa je stoga detekcija umora kod vozača stalan predmet istraživanja. Umor uzrokuje tjelesne promjene, primjerice promjene u aktivnosti mozga, usporavanje rada srca i pokreta očiju. Vozački umor je mentalni oblik umora kojeg karakterizira smanjenje motivacije, pospanost, smanjenja pažnje, slabija kontrola nad vozilom (npr. varijacije u brzini, krivudanje unutar prometne trake). Još opasnije je to što vozač nije svjestan koliko je umoran i sjeda u automobil. Kao posljedica umora može doći do mikro-sna, što je kratka epizoda spavanja koja može trajati djelić sekunde do trideset sekundi, a može se dogoditi u bilo koje vrijeme i obično nenadano. Metode za detekciju pospanosti načelno koriste sljedeće koncepte: analiza načina vožnje, analiza odziva vozila na vožnju, nadzor vozača u različitim situacijama koje izazivaju mjerljive fiziološke reakcije vozača. Mnogi proizvođači vozila ugrađuju ovakve sustave koji, između ostaloga, mogu pratiti i razinu sklopljenosti očiju vozača [26], analizirati zijevanje [27], pratiti položaj ruku za upravljačem i slično, te na temelju analiziranih parametara vozaču poručuju da odmori. Na slici 3.8 prikazan je prototip sustava tvrtke Denso kojim se detektira pospanost vozača analizom izraza lica.



Slika 3.8. *Sustav za detekciju pospanosti analizom izraza lica [28]*

3.3.4. Detekcija položaja tijela pri snimanju pokreta

Snimanje pokreta (engl. *motion capture*) je proces digitalnog snimanja pokreta tijela. Koristi se u zabavnoj industriji pri snimanju filmova i izradi video igara, u sportu za analizu pokreta tijela i uzroka ozljeda sportaša [29], u ergonomskim analizama posture tijela [30], od kojih je jedna prikazana na slici 3.9, te u robotici [31].



Slika 3.9. *Ergonomska analiza posture osobe [30]*

Popularni komercijalni proizvodi koji mogu detektirati položaje tijela su:

- Microsoft Kinect, prvotno namijenjen za korištenje u industriji video igara za kontroliranje likova u video igrama, sada se često koristi i robotici i medicini u obliku Azure Kinecta, koji je spojen i na Microsoft Azure oblak računala. Slika 3.10 prikazuje prepoznavanje geste pomoću Kinecta.
- Različiti igraći sustavi koji koriste virtualnu stvarnost (Oculus Rift, HTC Vive, Valve Index)

Slika 3.10 *Prepoznavanje geste pomoću Microsoft Kinecta [32]*

3.4. Idejno rješenje web sustava

Web sustav za praćenje hitnih medicinskih stanja na temelju analize slike i generiranja prirodnog jezika zamišljen je kao usluga koja svim korisnicima omogućuje korištenje istreniranog modela

strojnog učenja koji klasificira različite položaje tijela detektirane na kameri spojenom na računalo (može biti eksterna kamera ili web-kamera laptopa ili nešto drugo). Moguća je klasifikacija tri različita položaja tijela: držanje za glavu, koje predstavlja neku vrstu glavobolje, držanje za prsa, koje predstavlja bol u prsima, te držanje za trbuh, koje predstavlja bol u trbuhu. Za detektirani položaj tijela ispisuje se sažeti tekst opisa potencijalnog medicinskog stanja, kao i uputa za postupanje u nekoj takvoj situaciji. Osim korištenja već istreniranog modela, korisnicima se pruža mogućnost registracije i prijave. Prijavljeni korisnici imaju dodatne mogućnosti, kao što je treniranje vlastitog modela strojnog učenja sa vlastitim klasama i tekstom koji bi se sažeo. Skupove podataka za treniranje korisnici mogu spremati za kasniju upotrebu, uređivati i na taj način poboljšati model, te brisati u slučaju da nisu zadovoljni prikupljenim skupom podataka. Osim prikupljanja skupa podataka za treniranje, nakon treniranja vlastitog skupa, prijavljeni korisnici imaju dodatnu mogućnost ispitivanja istreniranog modela, te spremanje modela za kasniju upotrebu, bilo lokalno na računalo, ili u bazu podataka.

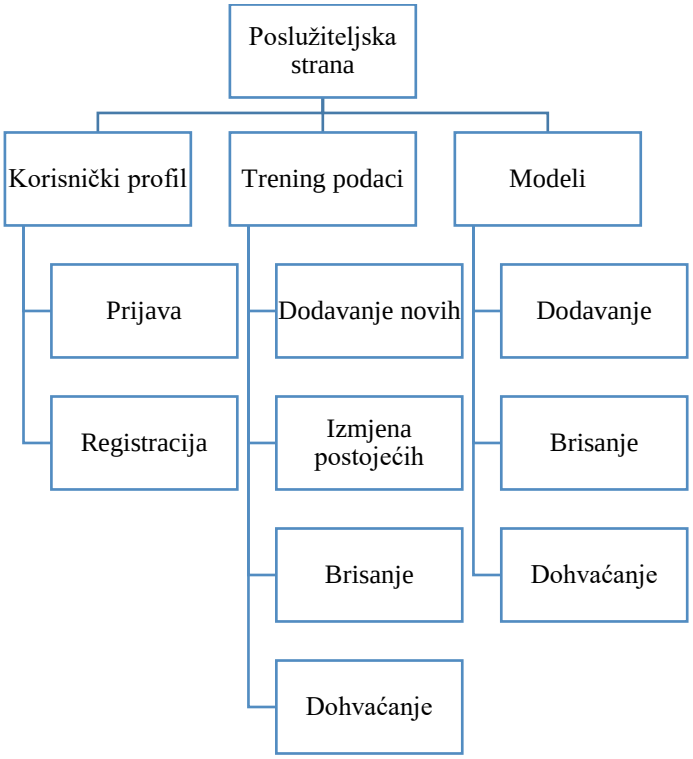
Sve funkcionalnosti aplikacije prikazane su u tablici 3.2.

Tablica 3.2. *Popis funkcionalnosti aplikacije*

Funkcionalnost aplikacije	Potreban korisnički račun
Registracija korisničkog računa	NE
Prijava u svoj korisnički račun	DA
Ispitivanje istreniranog modela koji detektira tri medicinska stanja (glavobolja, bol u prsima, bol u trbuhu)	NE
Prikupljanje podataka za vlastiti model klasifikacije položaja tijela	DA
Spremanje vlastitih skupova podataka za treniranje modela	DA
Uređivanje vlastitih skupova podataka za treniranje modela	DA
Brisanje vlastitih skupova podataka za treniranje modela	DA
Pregled vlastitih skupova podataka za treniranje modela	DA
Ispitivanje vlastitih modela strojnog učenja	DA
Učitavanje modela pomoću datoteka spremljenih na računalo	DA
Učitavanje modela pomoću datoteka spremljenih u bazu podataka	DA
Spremanje vlastitih modela na računalo	DA
Spremanje vlastitih modela u bazu podataka	DA
Brisanje spremljenih modela iz baze podataka	DA
Pregled vlastitih modela strojnog učenja	DA

Aplikacija je podijeljena na poslužiteljski i klijentski dio. Na poslužiteljskoj strani nalaze se dijelovi aplikacije odgovorni za interakciju s bazom podataka, pokretanje poslužitelja, validaciju podataka, te interakciju s Amazon S3 uslugom. Poslužiteljska strana napravljena je u izvršnom okruženju Node.js, programskom okviru Express.js, te koristi bazu podataka MongoDB i Amazon S3 uslugu za spremanje datoteka. Sastavni dijelovi ove strane programskog rješenja su sljedeći, a prikazani su na slici 3.11:

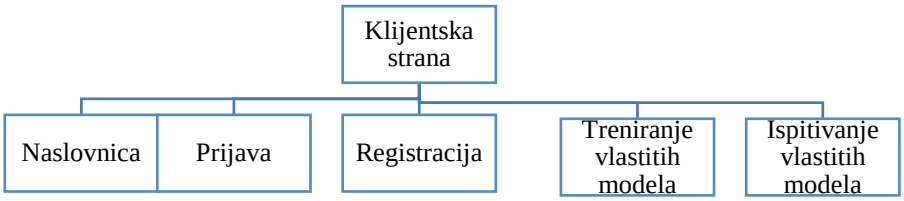
1. **Upravljanje korisničkim računom** – Dio programskog koda za validaciju primljenih podataka na temelju kojih se korisnik sprema u bazu podataka, prijavljuje u sustav ili mu se vraća greška.
2. **Trening podaci** – Sav programski kod koji se brine o spremanju novih trening podataka u bazu podataka, uređivanju i brisanju postojećih, te dohvaćanju informacija o svim korisničkim trening podacima.
3. **Modeli strojnog učenja** – Programski kod koji omogućuje brisanje, dohvaćanje i dodavanje istreniranog modela u bazu podataka i Amazon S3.



Slika 3.11. *Vizualni prikaz poslužiteljskog dijela aplikacije*

Na klijentskom dijelu koji je vizualno prikazan na slici 3.12 nalaze se različiti vizualni dijelovi aplikacije izrađeni pomoću programskog okvira Vue.js:

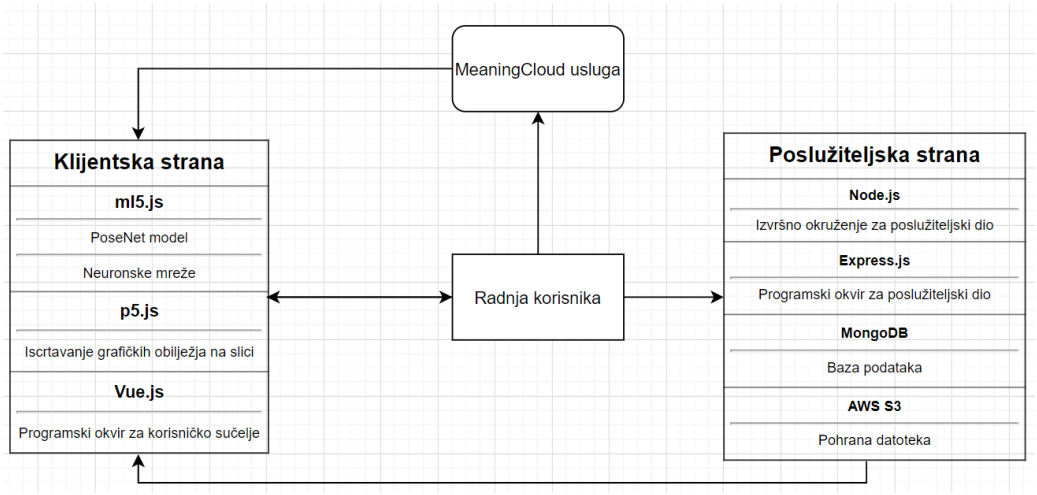
1. **Registracija** – Stranica na kojoj korisnik unosi korisničko ime i zaporku kako bi dobio mogućnost spremanja vlastitih skupova podataka i modela strojnog učenja.
2. **Prijava** – Stranica na kojoj registrirani korisnik unosi korisničko ime i zaporku radi prijave na stranicu i pristupu svim mogućnostima web aplikacije.
3. **Početna stranica** – Stranica kojoj svi imaju pristup, a omogućuje evaluaciju i isprobavanje već istreniranog modela strojnog učenja koji detektira hitne medicinske slučajeve
4. **Treniranje modela** – Stranica na kojoj prijavljeni korisnici mogu istrenirati i pohraniti vlastite modele strojnog učenja
5. **Evaluacija i pohrana vlastitih modela** – Stranica na kojoj prijavljeni korisnici imaju mogućnost ispitati vlastite modele te ih pohraniti u bazu podataka



Slika 3.12. *Vizualni prikaz klijentskog dijela aplikacije*

Za sažimanje teksta koristi se *MeaningCloud* usluga, za detektiranje položaja tijela model *PoseNet* u sklopu ml5.js biblioteke, a za prezentaciju detektiranog položaja tijela biblioteka p5.js za grafičko iscrtavanje.

Blok dijagram cijelog sustava, zajedno s poslužiteljskom i klijentskom stranom prikazan je na slici 3.13. Na klijentskoj strani nalazi se ml5.js biblioteka za potrebe detektiranja položaja tijela pomoću modela *PoseNet* i za potrebe klasifikacije slike i za treniranje modela koristeći neuronske mreže, p5.js biblioteka za iscrtavanje raznih grafičkih obilježja na slici, te Vue.js programski okvir koji koristi navedene biblioteke i kojim je izrađeno korisničko sučelje. Na neku radnju korisnika, koja može biti navođenje na određenu stranicu, klik mišem na neki gumb ili pritisak tipke na tipkovnici, može se kontaktirati *MeaningCloud* usluga za sažimanje teksta, poslužiteljska strana koja će dohvaćati ili spremati podatke, te klijentskoj strani vratiti neki odgovor, ili se može pozvati neka funkcionalnost koja je dio isključivo s klijentske strane, kao što je promjena nekog elementa korisničkog sučelja.



Slika 3.13. *Blok dijagram web sustava*

Za postupak klasificiranja položaja tijela na slici koristi se neuronska mreža koja dolazi kao sastavni dio ml5.js biblioteke. Razlog korištenja upravo neuronske mreže, a ne nekog drugog klasifikatora je taj što ml5.js biblioteka ne pruža mogućnost korištenja nekog drugog prikladnog algoritma. Kao ulaz u neuronsku mrežu kojom se klasificiraju položaji tijela osobe koristit će se izlazi *PoseNet* modela, kao što je prikazano na slici 3.14. *PoseNet* model, koji služi za detekciju ključnih točaka osobe na slici, kao ulaz prima sliku, a kao izlaz daje *x* i *y* koordinate za svaku od detektiranih 17 ključnih točaka zajedno s ocjenom pouzdanosti (engl. *confidence score*) za svaku ključnu točku.

KLJUČNA TOČKA	X-KOORDINATA	Y-KOORDINATA	OCJENA POUZDANOSTI
nose	x_1	y_1	c_1
leftEye	x_2	y_2	c_2
rightEye	x_3	y_3	c_3
leftEar	x_4	y_4	c_4
rightEar	x_5	y_5	c_5
leftShoulder	x_6	y_6	c_6
rightShoulder	x_7	y_7	c_7
leftElbow	x_8	y_8	c_8
rightElbow	x_9	y_9	c_9
leftWrist	x_{10}	y_{10}	c_{10}
rightWrist	x_{11}	y_{11}	c_{11}
leftHip	x_{12}	y_{12}	c_{12}
rightHip	x_{13}	y_{13}	c_{13}
leftKnee	x_{14}	y_{14}	c_{14}
rightKnee	x_{15}	y_{15}	c_{15}
leftAnkle	x_{16}	y_{16}	c_{16}
rightAnkle	x_{17}	y_{17}	c_{17}

Slika 3.14. *Ulazi i izlazi PoseNet modela i neuronske mreže*

Korištena neuronska mreža sastoji se od ulaznog sloja koji se sastoji od 34 koordinate (17 ključnih točaka, svaka sa svojom x i y koordinatom), potpuno povezanog sloja sa 16 skrivenih jedinica koji koristi ispravljenu linearnu aktivacijsku funkciju (*ReLU*), koja je definirana izrazom (3-1) i propušta vrijednosti veće od nule, drugog potpuno povezanog sloja sa 3 skrivene jedinice i *Softmax* aktivacijskom funkcijom koja kao izlaz daje vektor koji sadrži distribuciju vjerojatnosti

potencijalnih izlaza, te izlaznog sloja koji ima neku od tri vrijednosti koje predstavljaju konačne klase.

$$f(x) = \max(0, x) \tag{3-1}$$

Kako ulazi u neuronsku mrežu nisu značajno velikih dimenzija, dodavanjem dodatnih slojeva ili izmjenom parametara u nekom od postojeća dva sloja negativno bi utjecalo na performanse modela. Mreža je istrenirana na 60 epoha, uz stopu učenja u iznosu od 0.2, te s veličinom serije od 16 uzoraka.

4. PREGLED KORIŠTENIH TEHNOLOGIJA

Za potrebe izrade aplikacije korišteni su JavaScript programski jezik, programski okvir Vue.js za razvoj korisničkog sučelja, MongoDB kao baza podataka, a sa strane poslužitelja izvršno okruženje Node.js i programski okvir Express.js. Za potrebe detekcije poze korišten je PoseNet model u sklopu TensorFlow biblioteke uključene u ml5.js biblioteku, te se još koristi i p5.js biblioteka za prikaz i manipulaciju video zapisom.

4.1. Programski jezik JavaScript

Prema [33], JavaScript je dinamičan, slabo tipiziran i interpretiran programski jezik visokog nivoa, koji se najčešće koristi za dodavanje interaktivnosti web-stranicama, no može se koristiti i za pisanje programskog koda na strani poslužitelja uz npr. izvršno okruženje Node.js, za izradu Android i iOS aplikacija koristeći React Native programski okvir ili za izradu višeplatformskih *desktop* aplikacija koristeći Electron programski okvir.

JavaScript je skriptni jezik, jer se sastoji od niza naredbi koje se čitaju u interpreteru, a da se prethodno ne kompajlira sadržaj. Zbog ove karakteristike JavaScript se izvršava na strani korisnika, tj. na računalu na kojem je pokrenut sadržaj sa JavaScriptom.

JavaScript koristi strukturiranu sintaksu svakog programskog jezika više razine (npr. *if* izjave, *while* petlje, *switch* izjave, *do-while* petlje, i dr.). Kao i većina skriptnih jezika, JavaScript je jezik sa dinamičnom provjerom tipa podataka; tip je prije povezan sa vrijednošću nego sa svakim izrazom. Na primjer, ukoliko postoji varijabla koja je vezana za broj, ta ista varijabla kasnije može biti vezana za *String* tip.

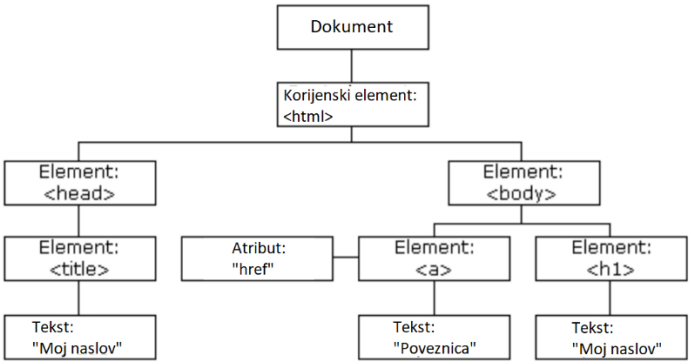
4.2. Tehnologije s klijentske strane

4.2.1. Vue.js

Prema [34], Vue je progresivni programski okvir za izradu korisničkih sučelja i, uz React i Angular, jedan je od najpopularnijih. Vrlo je brz, zauzima malo prostora, ima vrlo detaljnu dokumentaciju, fleksibilan je i prilagodljiv različitim vrstama projekata.

Vue koristi virtualni DOM, što je apstrakcija stvarnog HTML DOM-a (engl. *Document Object Model*), za manipulaciju HTML elementima zbog veličina DOM stabala u modernim web aplikacijama koje je potrebno konstantno modificirati, što utječe na performanse web aplikacije. Virtualni DOM se modificira prema potrebi programera, nakon čega se promjene spremaju u stvarno DOM stablo.

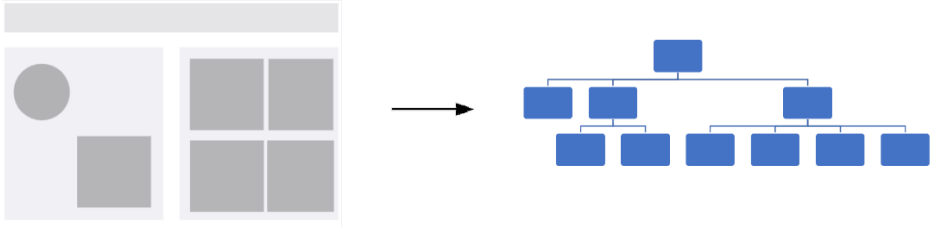
HTML DOM ili objektni model dokumenta jest stablo objekata neke HTML stranice. Nakon učitavanja stranice, preglednik izrađuje stablo objekata prema strukturi navedenoj u samom HTML dokumentu, što je prikazano slikom 4.1 [35].



Slika 4.1. *Prikaz HTML DOM stabla za jednostavnu web stranicu [35]*

Podaci u instanci Vue aplikacije i u DOM-u su povezani, stoga se izmjena nekog podatka u Vue instanci odražava i na DOM element.

Sustav komponenti je još jedan važan koncept svake Vue aplikacije, jer je on apstrakcija koja omogućuje izradu aplikacija velikih razmjera koristeći male, zasebne i često ponovno upotrebljive komponente. Svako sučelje neke aplikacije može se zamisliti kao stablo manjih komponenti, što je vizualno prikazano na slici 4.2, po uzoru na [34].



Slika 4.2. *Apstrakcija sučelja u stablo komponenti*

Komponenta je Vue instanca sa nekim preddefiniranim opcijama. Stvaranje jednostavne komponente prikazano je isječkom programskog koda koji se nalazi na slici 4.3. Prikazana komponenta naziva se „*todo-item*“, te joj je moguće preko *props* polja proslijediti podatak „*todo*“. Proslijeđeni podatak moguće je koristiti unutar HTML prikaza komponente koji se definira u *template* atributu.

```
Vue.component('todo-item', {
  props: ['todo'],
  template: '<li>{{ todo.text }}</li>'
})
```

Slika 4.3. *Definiranje komponente „todo-item“ u Vue-js programskom okviru*

Izrađena komponenta se zatim može koristiti u drugim dijelovima programskog koda na način da se unese pomoću ključne riječi *import*, kao što je prikazano na slici 4.4.

```
<template>
  <div id="app">
    <ol>
      <todo-item v-for="item in groceryList" :todo="item" :key="item.id"></todo-item>
    </ol>
  </div>
</template>

<script>
import TodoItem from '../components/TodoItem.vue';
export default {
  data() {
    return {
      groceryList: [
        { id: 0, text: 'Vegetables' },
        { id: 1, text: 'Cheese' },
        { id: 2, text: 'Chocolate' },
      ],
    };
  },
};
</script>
```

Slika 4.4. *Korištenje definirane komponente*

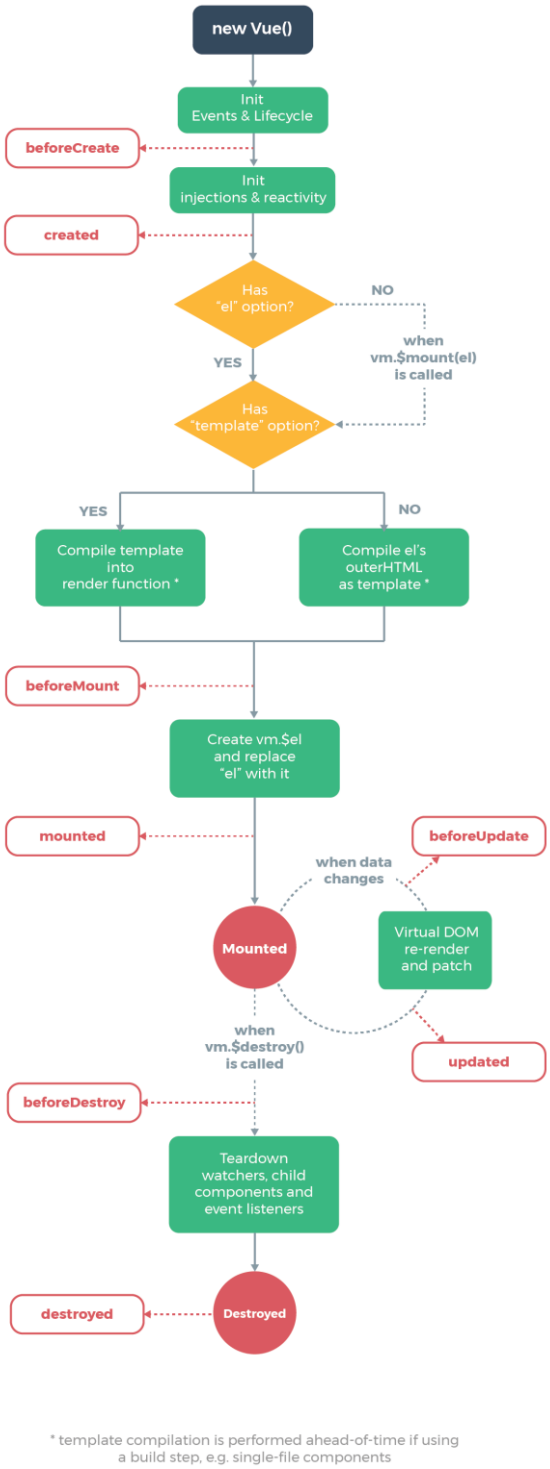
Komponenta se u HTML-u koristi kao posebna oznaka „*<todo-item>*“, koja se posebnom Vue direktivom *v-for* ponavlja za svaku stavku unutar „*groceryList*“ objekta definiran u *data()* funkciji. Svakoj *todo-item* komponenti prosljeđuje se po jedna stavka unutar polja koja se prikazuje u poredanoj listi. Rezultat programskog koda vidljiv je na slici 4.5.

1. Vegetables
2. Cheese
3. Chocolate

Slika 4.5. *Prikaz izrađene komponente*

Da bi se izvršila neka radnja u određenom trenutku u “životu” jedne Vue instance ili komponente, u Vue su integrirane tzv. udice životnog ciklusa (eng. *lifecycle hooks*). Udice omogućavaju da se željena akciju pokrene u odgovarajućem trenutku životnog ciklusa neke komponente. Udice u

okviru Vue.js programskog okvira su predstavljene kroz specifične predefinirane metode koje to omogućuju, a koje su dijagramom prikazane na slici 4.6 [34].



4.6. *Životni ciklus Vue.js komponente [34]*

Udice se mogu podijeliti u nekoliko kategorija:

1. Inicijalne udice - omogućuju da se doda neka akcija prije nego što je instanca dodana u DOM, tj. prije renderiranja
    - *beforeCreate* - Radnje koje su definirane u ovoj metodi se izvršavaju pri samoj inicijalizaciji instance, "*data*" svojstvo još nije reaktivno, a događaji još nisu aktivirani
    - *created* - Radnje koje su definirane u ovoj metodi mogu pristupiti "*data*" svojstvu i događaji su aktivirani, no "*template*" i virtualni DOM nisu renderirani
2. *Mounting* udice – vezane su za trenutak renderiranja instance u DOM-u. Jedna je vezana za trenutak pre renderiranja, a druga za trenutak nakon renderiranja. Najčešće se koristi za preuzimanje podataka za komponentu.
    - *beforeMount* – Poziva se prije prvog poziva *render* funkcije
    - *mounted* – Udica kojom se dobije potpun pristup reaktivnim komponentama i renderiranom DOM-u. Radnje koje su definirane u ovoj metodi se izvršavaju nakon renderiranja instance.
3. *Update* udice – koriste se da definiraju trenutak prije i nakon neke promjene koja uzrokuje ponovno renderiranje
    - *beforeUpdate* - Radnje koje su definirane u ovoj metodi se izvršavaju nakon promjene podataka u komponenti u trenutku kada ciklus ažuriranja počinje, točno prije nego što se DOM ažurira i ponovno renderira.
    - *updated* - Radnje koje su definirane u ovoj metodi se izvršavaju nakon promjene podataka i komponenti, točno nakon što se DOM ažurira i ponovno renderira
4. *Destruction* udice – omogućuju izvršavanje radnji u vrijeme kada je komponenta uništena i uklonjena iz DOM-a.
    - *beforeDestroy* - Radnje koje su definirane u ovoj metodi se izvršavaju prije nego što je komponenta uništena, ali dok je još funkcionalna
    - *destroyed* - Radnje koje su definirane u ovoj metodi se izvršavaju nakon uništenja instance i uklanjanja iz DOM-a

4.2.2. p5.js

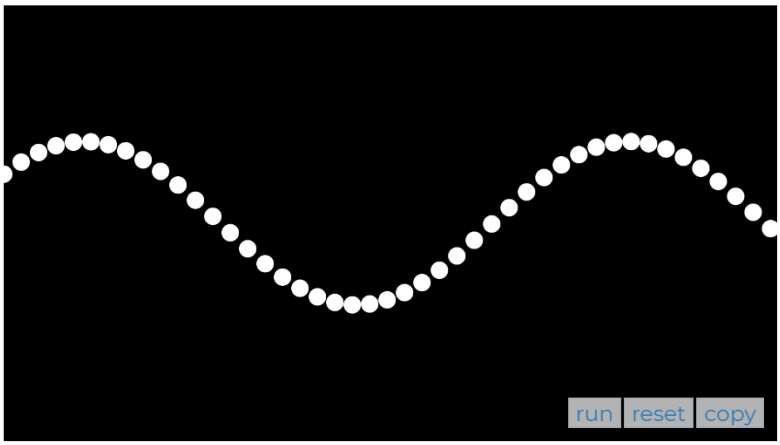
p5.js [36] je JavaScript biblioteka otvorenog koda za kreativno kodiranje, čiji je cilj učiniti programiranje pristupačno za umjetnike, dizajnere, edukatore, početnike i druge. Biblioteka je

inspirirana i napisana po uzoru na integrirano razvojno okruženje (engl. *integrated development environment*) i grafičku biblioteku Processing koja koristi Java programski jezik.

p5.js nudi cijeli niz funkcija za crtanje oblika i manipulaciju HTML DOM elementima, koje se mogu razvrstati u kategorije kao što su: manipulacija bojom, crtanje raznih oblika (kružnica, linija, trokuta, točaka itd.), kreiranje, uklanjanje i transformacija DOM elemenata, detektiranje događaja (klik i pozicija miša, pritisak tipke), matematičke operacije, manipulacija 3D crtežima, te manipulacija datotekama. Sve te funkcije mogu se koristiti u JavaScript datoteci koja se učita u HTML stranicu, a takva JavaScript datoteka u kontekstu p5.js-a zove se skica (engl. *sketch*).

Obavezni dio svake p5.js skice su dvije funkcije: *setup* funkcija koja se pokreće samo jednom i u kojoj se inicijaliziraju sve varijable potrebne za skicu, te *draw* funkcija koja se izvršava direktno nakon *setup* funkcije, a programski kod koji je dio *draw* funkcije izvršava se u beskonačnoj petlji, te se na taj način mogu dobiti razni efekti, kao što je npr. iluzija kretanja nekog objekta.

Na slici 4.7. prikazan je rezultat skice za crtanje i animiranje sinusoidalnog vala, a programski kod koji iscrtava skicu prikazan je na slici 4.8. Skica se sastoji od dvije glavne funkcije, *draw* i *setup*, te dvije pomoćne funkcije *calcWave* i *renderWave*. U *setup* funkciji se postavljaju vrijednosti kao što su početne vrijednosti x i y koordinata na kojima će se crtati elipse i HTML *canvas* element u kojem će se skica crtati, dok se u *draw* funkciji u beskonačnoj petlji pozivaju pomoćne funkcije kojima se izračunavaju nove vrijednosti x i y koordinata, te se onda na tim novim vrijednostima opet iscrtavaju elipse koje zajedno čine sinusoidu.



Slika 4.7. *Sinusoidalni val nacrtan pomoću p5.js biblioteke*

```
let xspacing = 16; // Udaljenost između horizontalnih lokacija
let w; // Širina vala
let theta = 0.0; // Početni kut
let amplitude = 75.0; // Visina vala
let period = 500.0; // Broj piksela prije ponavljanja vala
let dx; // Vrijednost za inkrement x vrijednosti
let yvalues; // Polje za spremanje vrijednosti visina vala

function setup() {
  createCanvas(710, 400);
  w = width + 16;
  dx = (TWO_PI / period) * xspacing;
  yvalues = new Array(floor(w / xspacing));
}
function draw() {
  background(0);
  calcWave();
  renderWave();
}
function calcWave() {
  theta += 0.02;

  let x = theta;
  for (let i = 0; i < yvalues.length; i++) {
    yvalues[i] = sin(x) * amplitude;
    x += dx;
  }
}
function renderWave() {
  noStroke();
  fill(255);
  for (let x = 0; x < yvalues.length; x++) {
    ellipse(x * xspacing, height / 2 + yvalues[x], 16, 16);
  }
}
```

Slika 4.8. *Programski kod za iscrtavanje sinusoidalnog vala koristeći p5.js biblioteku*

4.2.3. ml5.js

Prema [37], ml5.js je JavaScript biblioteka otvorenog koda izgrađena povrh Tensorflow.js biblioteke i p5.js biblioteke, te omogućava jednostavno korištenje već istreniranih modela u internetskom pregledniku, kao i treniranje vlastitih.

Mogućnosti koje ml5.js biblioteka nudi dijele se u četiri skupine: kategorija „pomagača“ (engl. *helpers*) u koju su svrstane funkcije za pomoć u obradi podataka i treniranje modela strojnog učenja, te kategorije za sliku, zvuk i tekst.

Pod kategoriju „pomagača“ spadaju nekoliko algoritama:

- Neuronske mreže – treniranje vlastitih modela za zadatke klasifikacije ili regresije, učitavanje i evaluacija postojećih modela
- Ekstrakcija značajki
- Klasifikacija podataka koristeći algoritam k-najbližih susjeda
- Klasterizacija metodom K-srednjih vrijednosti

Ostale kategorije koriste već istrenirane modele strojnog učenja, kao što su *ImageClassifier* model za prepoznavanje sadržaja slike koji koristi *ImageNet* bazu podataka koja sadrži 15 milijuna slika,

PoseNet model za prepoznavanje položaja tijela, *BodyPix* model za segmentaciju osobe na slici, *FaceApi* model za detekciju lica, *SketchRNN* model za prepoznavanje nacrtanih objekata, *PitchDetection* model za detekciju visine tona zvuka, *Sentiment* model za analizu sentimenta teksta, te drugi modeli.

4.3. Tehnologije s poslužiteljske strane

4.3.1. MongoDB

Prema [38], MongoDB jedna je od najčešće upotrebljavanih NoSQL baza podataka optimizirana za protok velike količine podataka, te se temelji na konceptu zbirki i dokumenata, koje su ekvivalentne tablicama, odnosno unosima u tablici u relacijskim bazama podataka. Dokumenti koji se pohranjuju u MongoDB bazu podatka su JSON (engl. *JavaScript Object Notation*) oblika, što znači da se svaki dokument sastoji se od niza parova ključ-vrijednost, koji su temeljne jedinice podataka u Mongo bazi podataka. Ključ je uvijek znakovni niz, a vrijednost koja se sprema pod ključem može biti bilo koja podržana JSON vrijednost: od broja, *Stringa*, objekta, polja, pa do *null* ili *undefined* vrijednosti, ili nekog korisnički definiranog tipa podatka.

U tablici 4.1. dana je usporedba između pojmova kod Mongo baze i općenite relacijske baze podataka.

Tablica 4.1. *Usporedba relacijskih i NoSQL baza podataka*

Relacijska baza podataka	**NoSQL baza podataka**
Baza podataka	Baza podataka
Tablica	Zbirka
Redak	Dokument
Stupac	Polje
Spajanje tablica	Ugrađeni dokumenti i povezivanje
Primarni ključ	Primarni ključ

4.3.2. Node.js

Prema [39], Node.js je JavaScript platforma na strani poslužitelja koje koristi Google Chrome V8 *engine*. Node.js razvio je Ryan Dahl 2009. godine, a 2011. postao je projekt otvorenog koda. Razlozi korištenja Node.js-a su: velika brzina, koja je postignuta korištenjem Chrome V8 *enginea*, jednonitan je, ali vrlo skalabilan, neblokirajuć je, te je asinkron i pogonjen događajima. Sve značajke Node.js platforme su asinkrone i neblokirajuće, što znači da poslužitelj temeljen na

Node.js-u nikad ne čeka da se rezultat operacije završi, nego nastavi s izvođenjem dok ne dođe do događaja, kada mehanizam za obavijesti o događajima potvrđuje završetak prethodne operacije.

Program zatim može iskoristiti rezultat te asinkrone operacije i pristupiti mu, te nastaviti s izvođenjem svog zadatka. Za upravljanje asinkronim funkcijama u JavaScriptu postoji nekoliko različitih pristupa: povratne (engl. *callback*) funkcije, obećanja (engl. *promises*) i ključne riječi *async/await* koje, konceptualno, koriste obećanja.

Callback je mehanizam koji omogućuje da se funkcija proslijedi kao parametar, a kasnije se poziva po potrebi. Jedan od primjera korištenja *callback* funkcije su metode *Array* objekta. Ovaj objekt pruža mogućnost prolaska kroz sve elemente polja uz mogućnost da programer isprogramira obradu pojedinog elementa. Primjer korištenja *callback* funkcije dan je na slici 4.9.

```
1 const array = [1, 4, 9, 16];
2 function callback(x) {
3     return x * 2;
4 }
5 // proslijedi se funkcija callback u map metodu
6 const map = array.map(callback);
7
8 console.log(map);
9 // izlaz: Array [2, 8, 18, 32]
```

Slika 4.9. *Korištenje callbacka zajedno s map metodom*

Definira se polje brojeva *array*, te funkcija naziva *callback* koja kao parameter prima varijablu *x*, a kao rezultat vraća dvostruku vrijednost varijable *x*. Ako se ta funkcija proslijedi metodi *map* koja djeluje nad poljem brojeva *array*, kao rezultat dobit će se novo polje koje sadrži vrijednosti originalnog polja *array* uvećane za dva puta.

U najvećem broju slučajeva *callback* funkcije koriste se kao funkcije za obradu događaja (engl. *event handler*). To su funkcije koje se aktiviraju kada u web aplikaciji dođe do nekog događaja (najčešće kao posljedica korisnikove aktivnosti), ili kada je potrebno čekati na odgovor poslužitelja.

U JavaScriptu postoji izraz za programski kod sa ugniježdenim *callback* funkcijama pod nazivom "*callback hell*". Otklanjanje pogrešaka u takvom programskom kodu, a i samo razumijevanje je prilično otežano. Sa ES2015 standardom je došla nova sintaksa pod nazivom "*Promise*", koja sa svojim API-jem osigurava bolji i pregledniji način za organiziranje *callback* funkcija. *Promise* je sinkrono vraćen objekt pri asinkronoj operaciji, koji predstavlja privremenu zamjenu za moguće rezultate te asinkrone operacije. *Promise* umjesto krajnje vrijednosti daje obećanje da će dostaviti

tu vrijednost u nekom trenutku u budućnosti. Na taj način i sinkrone i asinkrone operacije mogu vratiti neku vrijednosti, s tim što sinkrone odmah vraćaju krajnji podatak, a asinkrone rezerviraju mjesto za budući podatak. Asinkrona funkcija može imati dva moguća krajnja rezultata, a to su "uspješno izvršena operacija" ili "neuspješno izvršena operacija", dok se *Promise* može nalaziti u jednom od tri stanja:

- *Pending* – kada se asinkrona radnja još uvijek izvršava
- *Resolve* – kada je asinkrona radnja završena uspješno
- *Reject* – kada je asinkrona radnja neuspješno završena greškom

Korištenje samo jednog *Promisea* je jednostavno, ali kada se program zakomplicira asinkronom logikom, rad sa *Promiseima* se otežava. Sa ES2017 standardom stigla je i nova sintaksa "*async/await*", koja olakšava rad sa *Promiseima* i omogućava jednostavnije predstavljanje niza asinkronih *Promisea*. Korištenje "*async/await*" sintakse omogućuje da višestruke zavisne asinkrone radnje prikažemo čitljivije, te programski kod postane sličan sinkronom programskom kodu. Razlika između sva tri načina pisanja asinkronog JavaScript koda prikazan je na slici 4.10.

Callback funkcije

```
GetUser(function (err, user) {
  GetProfile(user, function (err, profile) {
    GetAccount(profile, function (err, account) {
      GetReport(account, function (err, report) {
        SendStatistics(report, function (err, statistics) {
          // ...
        });
      });
    });
  });
});
```

Promise sintaksa

```
GetUser()
  .then(GetProfile)
  .then(GetAccount)
  .then(GetReport)
  .then(SendStatistics)
  .then(function(result) {
    console.log(result)
  })
  .catch(function(error) {
    console.log(error);
  })
```

Async/await sintaksa

```
async function doAsyncOperation() {
  const user = await GetUser(1);
  const profile = await GetProfile(user);
  const account = await GetAccount(profile);
  const report = await GetReport(account);

  const sendResults = SendStatistics(report);
  console.log(sendResults);
}
```

Slika 4.10. *Programski kod napisan callback funkcijama, Promise sintaksom i async/await sintaksom*

4.3.3. Amazon S3

Prema [40], Amazon S3 ili *Amazon Simple Storage Service* je usluga koju pruža *Amazon Web Services* (*AWS*) platforma u računalu oblaka koja omogućuje pohranu datoteka preko web sučelja. Amazon S3 koristi istu skalabilnu infrastrukturu koju koristi i Amazon.com za potrebe mreže e-trgovina. Usluga se može koristiti za spremanje bilo koje vrste datoteka, što ju čini prikladnom za različite vrste namjena: pohrana datoteka za web-aplikacije, pohrana sigurnosnih kopija i datoteka za oporavak, pohrana arhiva podataka i slično. AWS je pokrenuo uslugu u Sjedinjenim Američkim Državama 14. ožujka 2006., te kasnije i u Europi u studenome 2007. godine.

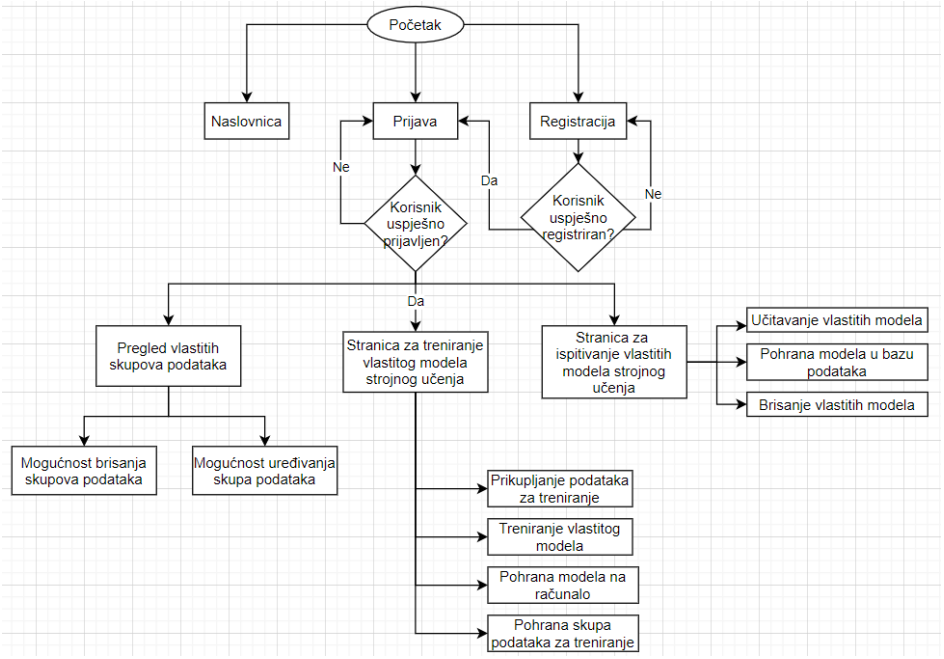
Osnovne jedinice za pohranu su objekti koji su organizirani unutar tzv. kanti (engl. *buckets*). Svaki pohranjeni objekt identificiran je jedinstvenim, korisnički dodijeljenim ključem. Svakoj kanti

moguće je pristupiti preko AWS konzole, programski koristeći AWS SDK, ili koristeći Amazon S3 REST API.

5. PROGRAMSKO RJEŠENJE WEB SUSTAVA

U ovom poglavlju detaljno će se opisati struktura i mogućnosti izrađene web aplikacije, te će se opisati i objasniti različiti dijelovi programskog koda aplikacije na korisničkom sučelju i na poslužiteljskoj strani. Aplikacija će imati mogućnost registracije, prijave, testiranja gotovog modela detektiranja položaja tijela, izradu vlastitog modela detektiranja položaja tijela, te spremanje i uređivanje vlastitih modela.

Cijeli tijek rada sustava dan je dijagramom na slici 5.1. Svaki korisnik na raspolaganju ima opciju naslovnice, na kojoj se nalazi istrenirani model koji pomoću kamere spojene na računalo korisnika klasificira tri medicinska stanja: glavobolju, bol u prsima i bol u trbuhu. Osim testiranja modela s naslovnice, korisnik ima i mogućnost prijave i registracije. Registracijom korisnik stvara korisnički račun kojim se prijavljuje u sustav. Pri prijavi i registraciji postoji validacija kojom se osigurava jedinstvenost korisničkog imena, duljina korisničkog imena i lozinke, te podataka za prijavu. Nakon uspješne prijave, korisnik na raspolaganju ima dodatne mogućnosti kao što su stranica za treniranje vlastitog modela strojnog učenja i ispitivanje vlastitih istreniranih modela.

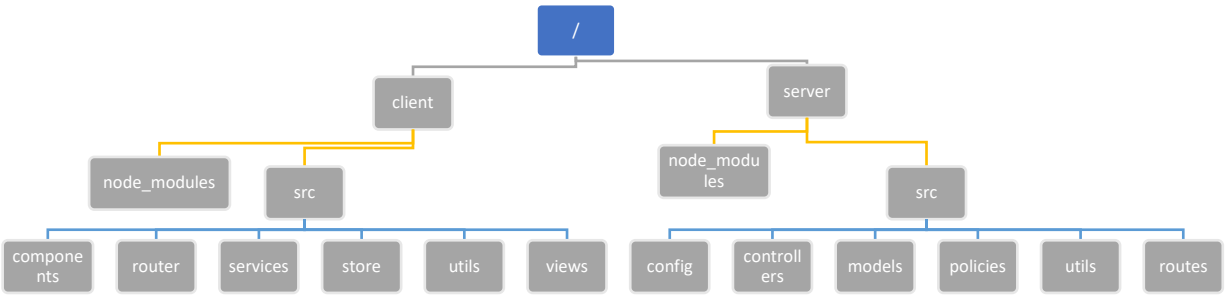


Slika 5.1. *Dijagram toka rada cijelog web sustava*

5.1. Struktura programskog rješenja

Programsko rješenje podijeljeno je u dvije cjeline: klijentsku i poslužiteljsku stranu. Klijentska strana sadrži različite vizualne i interaktivne elemente koji su izrađeni koristeći programski okvir Vue.js, te biblioteke p5.js i ml5.js. Poslužiteljska strana sadrži dio programskog koda odgovoran za interakciju s bazom podataka, za pokretanje poslužitelja, validaciju korisničkih podataka, te

spremanje i dohvaćanje modela s Amazon Web Services S3 *bucketa*. Prikaz strukture programskog rješenja dan je na slici 5.2.



Slika 5.2. *Stablo strukture programskog rješenja*

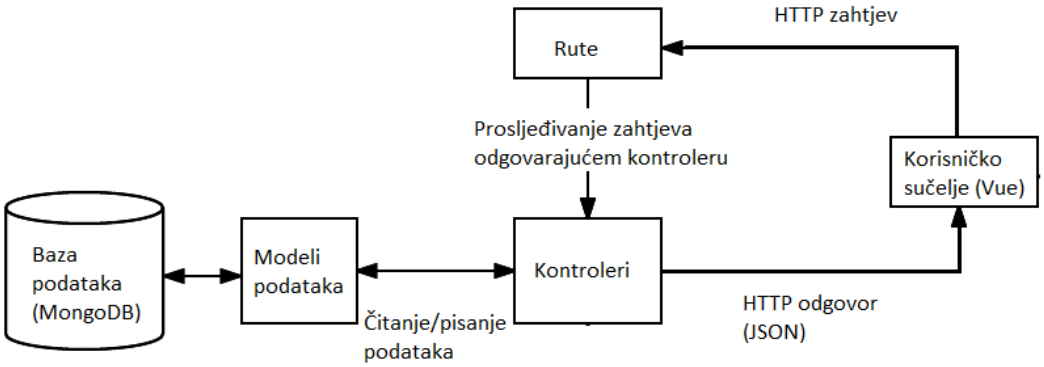
Detaljniji opisi sadržaja i funkcija pojedinih direktorija dan je u tablici 5.1.

Tablica 5.1. *Detaljnije objašnjenje strukture programskog rješenja*

DIREKTORIJ	SADRŽAJ DIREKTORIJA
/client/src	Cjelokupan programski kod korisničkog sučelja
/client/src/components	Sve manje Vue.js komponente koje se mogu ponovno koristiti
/client/src/router	Programski kod odgovoran za preusmjeravanje na različite URL-ove, tj. rute
/client/src/services	Programski kod za komuniciranje s poslužiteljskim dijelom aplikacije i vanjskim API-jima
/client/src/store	Stanje aplikacije koji se koriste kroz cijelu aplikaciju (podaci o korisniku, informaciju o stanju prijave korisnika)
/client/src/views	Veće Vue.js komponente koje predstavljaju različite stranice aplikacije
/client/src/utils	ml5.js i p5.js funkcije za prikupljanje podataka, treniranje i evaluaciju modela
/server/src	Cjelokupan kod poslužiteljske strane
/server/src/config	Konfiguracijske datoteke vezane za aplikaciju (port poslužitelja, URL za bazu podataka, API ključevi)
/server/src/controllers	Programski kod za komunikaciju između korisničkog sučelja i baze podataka
/server/src/models	Modeli MongoDB zbirki
/server/src/policies	Pravila za ovjere pristiglih podataka poslužitelju
/server/src/utils	Metode za komunikaciju s Amazon S3, konstante koje se koriste kroz aplikaciju
/server/src/routes	Sve krajnje točke (engl. endpoint) i HTTP metode kojima se točkama pristupa
../node_modules	Svi *npm* paketi potrebni za rad aplikacije

Kompletan postupak komunikacije između klijentske strane i poslužiteljske strane vizualno je prikazan na slici 5.3. Pomoću korisničkog sučelja izrađenog u Vue.js programskom okviru šalju

se HTTP zahtjevi na neku od ruta definiranima na poslužiteljskoj strani. Svaka ruta povezana je s odgovarajućim kontrolerom u kojem je definiran postupak koji treba obaviti, što je uvijek neka vrsta komunikacije s bazom podataka u obliku dohvaćanja, brisanja, uređivanja ili dodavanja novih podataka. Nakon što kontroler završi s komunikacijom s bazom podataka, klijentskoj strani šalje se HTTP odgovor zajedno s podacima u JSON obliku.



Slika 5.3. *Proces komunikacije između korisničkog sučelja i poslužiteljske strane*

5.2. Postupak treniranja modela

Postupak kojim je izrađen model na početnoj stranici aplikacije jednak je postupku kojim registrirani korisnici mogu istrenirati vlastiti model, a sastoji se od dvije funkcije: *collectData* kojom se skupljaju podaci za treniranje modela i *trainModel* kojom se prikupljeni podaci koriste za trening modela strojnog učenja koji koristi neuronsku mrežu. Obje funkcije izvršavaju rad u internetskom pregledniku korisnika, te koriste ml5.js biblioteku i PoseNet model za potrebe strojnog učenja, te p5.js za aktivaciju kamere, vizualni prikaz ključnih točaka tijela osobe na slici, detekciju pritisnute tipke, te petlju u kojoj se ažuriraju podaci o pozi osobe na slici.

Prije treniranja modela, potrebno je prikupiti podatke za trening, za čiju se svrhu koristi *collectData* funkcija, koja igra ulogu p5.js skice. Funkcija *collectData* sastoji se od nekoliko podfunkcija: obaveznih p5.js funkcija *setup* i *draw*, p5.js funkcije *keyPressed* koja detektira pritisak tipke, te *callback* funkcija *gotPoses* i *modelLoaded*.

Unutar funkcije *setup*, prikazanoj na slici 5.4, inicijaliziraju se sve početne vrijednosti koje su potrebne kako bi p5.js skica pravilno radila; stvara se HTML *canvas* element u kojem će se snimati video s kamere računala, stvara se HTML *Media Capture* element kojim će se snimati video, učitava se PoseNet model, postavljaju se *callback* funkcije *modelLoaded* koja se izvršava kada se PoseNet model učita, te *callback* funkcija *gotPoses* kada se detektira položaj tijela. Osim toga, postavljaju se i parametri neuronske mreže po kojoj će se model trenirati. Kao što je vidljivo na

slici, neuronska mreža ima 34 ulaza (x i y koordinate za 17 ključnih točaka), postavlja se željeni broj izlaza, postavlja se zadatak neuronske mreže koji je u ovom slučaju klasifikacija, te se naposljetku neuronska mreža inicijalizira koristeći ml5.js biblioteku.

```
p5.setup = (_) => {
  canvas = p5.createCanvas(settings.videoWidth, settings.videoHeight);

  video = p5.createCapture(p5.VIDEO);
  sendVideo(video, canvas);
  video.hide();
  poseNet = ml5.poseNet(video, modelLoaded);
  poseNet.on('pose', gotPoses);

  let options = {
    inputs: 34,
    outputs: settings.numOfOutputs,
    task: 'classification',
    debug: true
  };
  neuralNetwork = ml5.neuralNetwork(options);
};
```

Slika 5.4. *Podfunkcija setup unutar collectData funkcije*

Callback funkcije *modelLoaded* i *gotPoses*, prikazane na slici 5.5, definiraju što će se dogoditi kada se učita PoseNet model, odnosno kada se detektira neki položaj tijela. Funkcija *modelLoaded* samo ispiše poruku da je PoseNet model spreman, a u funkciji *gotPoses* definira se polje podataka *inputs*, te ako je detektiran neki položaj tijela i ako je varijabla *state* u stanju „*collecting*“, tj. ako se podaci prikupljaju, tada se polje *inputs* počne popunjavati x i y koordinatama svake ključne točke, te se to polje detektiranih koordinata dodaje neuronskoj mreži zajedno s oznakom klase kojoj prikupljeni ulazi pripadaju pomoću funkcije *addData* koja je dio ml5.js biblioteke.

```
function gotPoses(poses) {
  let inputs = [];
  if (poses.length > 0) {
    pose = poses[0].pose;
    skeleton = poses[0].skeleton;
    if (state == 'collecting') {
      inputs = [];
      for (let i = 0; i < pose.keypoints.length; i++) {
        let x = pose.keypoints[i].position.x;
        let y = pose.keypoints[i].position.y;
        inputs.push(x);
        inputs.push(y);
      }
      let target = [settings.targetLabel];
      neuralNetwork.addData(inputs, target);
    }
  }
}

function modelLoaded() {
  console.log('poseNet ready');
}
```

Slika 5.5. *Callback funkcije gotPoses i modelLoaded unutar collectData funkcije*

Unutar funkcije *draw* prikazanoj na slici 5.6, koja se vrti u petlji, osvježava se video element novom slikom, te se, ako je detektiran neki položaj tijela, iscrtavaju linije između određenih

ključnih točaka na način da se na tijelu osobe iscrta tzv. kostur, a na pozicijama samih ključnih točaka iscrtavaju se crni krugovi kako bi se ključne točke istaknule.

```
p5.draw = (_) => {
  p5.translate(video.width, 0);
  p5.scale(-1, 1);
  p5.image(video, 0, 0, video.width, video.height);

  if (pose) {
    for (let i = 0; i < skeleton.length; i++) {
      let a = skeleton[i][0];
      let b = skeleton[i][1];
      p5.strokeWeight(2);
      p5.stroke(0);

      p5.line(a.position.x, a.position.y, b.position.x, b.position.y);
    }
    for (let i = 0; i < pose.keypoints.length; i++) {
      let x = pose.keypoints[i].position.x;
      let y = pose.keypoints[i].position.y;
      p5.fill(0);
      p5.stroke(255);
      p5.ellipse(x, y, 16, 16);
    }
  }
};
```

Slika 5.6. *Draw funkcija unutar collectData funkcije*

Kako bi se sam postupak prikupljanja podataka pokrenuo, potrebno je pritisnuti neku tipku, nakon kojeg će se izvršiti niz radnji definiranih u *keyPressed* funkciji koja je prikazana na slici 5.7. Ukoliko se pritisne tipka „*Escape*“, snimanje se zaustavlja, ali u slučaju pritiska bilo koje druge tipke, nakon određenog vremenskog razmaka varijabla *state* poprimiti će vrijednost „*collecting*“, što će utjecati na prethodno objašnjenu *gotPoses* funkciju, te će se podaci početi prikupljati. Nakon drugog vremenskog odmaka, varijabla *state* poprimit će vrijednost „*waiting*“, te će se podaci prestati prikupljati.

```
p5.keyPressed = (_) => {
  if (p5.key == "Escape") {
    sendStopRecording();
  }
  sendKeyPressed(true);
  sendRecordingFinished(false);
  setTimeout(function() {
    state = 'collecting';
    sendKeyPressed(false);
    sendCurrentlyRecordingCheck(true);
    sendRecordingFinished(false);
    setTimeout(function() {
      sendData(neuralNetwork.neuralNetworkData.data.raw);
      sendCurrentlyRecordingCheck(false);
      sendRecordingFinished(true);
      state = 'waiting';
    }, settings.videoDuration);
  }, settings.videoDelay);
};
```

Slika 5.7. *KeyPressed funkcija unutar collectData funkcije*

Nakon što su svi podaci prikupljeni, prosljeđuju se funkciji *trainModel*, koja se sastoji od vlastite podfunkcije *setup* i *callback* funkcija *finished* i *dataReady*. Funkcija *setup*, prikazana na slici 5.8, slično kao i kod prikupljanja podataka, služi za inicijaliziranje svih potrebnih varijabli i ostalih

vrijednosti. Definira se HTML *canvas* element u kojem će se prikazivati graf koji prikazuje performanse modela, te se definira neuronska mreža sa istim opcijama definiranim kao i kod prikupljanja podataka. Osim toga, neuronskoj mreži predaju se prikupljeni podaci pomoću metode *loadData*, a kada se podaci učitaju, poziva se funkcija *dataReady*.

```
p5.setup = (_) => {
  canvas = p5.createCanvas(settings.videoWidth, settings.videoHeight);
  sendCanvas(canvas);
  let options = {
    inputs: 34,
    outputs: settings.numOfOutputs,
    task: 'classification',
    debug: true,
    learningRate: settings.learningRate,
  };
  neuralNetwork = ml5.neuralNetwork(options);
  neuralNetwork.loadData(settings.sampleData, dataReady);
};
```

Slika 5.8. *Setup funkcija unutar trainModel funkcije*

Funkcije *dataReady* i *finished*, prikazane na slici 5.9, služe za treniranje i spremanje modela. Unutar *dataReady* funkcije prikupljeni podaci se normaliziraju, te se neuronska mreža trenira s određenim brojem epoha i veličinom serije. Nakon što se model istrenira, poziva se funkcija *finished* koja generira četiri datoteke:

- model.json - sadrži informacije o topologiji modela, kao što su informacije o slojevima, verziji Tensorflowa i sl.
- model_meta.json – sadrži informacije o prikupljenim podacima, kao što su broj ulaza, broj izlaza, jesu li podaci normalizirani, minimalne i maksimalne vrijednosti pojedinih ulaza
- weights.bin – binarna datoteka koja sadrži informacije o težinskim vrijednostima neurona u istreniranom modelu neuronske mreže
- text_to_summarize.json – tekstovi koje je potrebno sažeti za pojedini izlaz

```
function dataReady() {
  neuralNetwork.normalizeData();
  neuralNetwork.train(
    {
      epochs: settings.epochs,
      batchSize: settings.batchSize,
    },
    finished,
  );
}
function finished() {
  sendFinishedStatus();
  neuralNetwork.save();
  const sampleDataStringified = JSON.stringify(settings.textToSummarize);
  const blob = new Blob([sampleDataStringified], { type: 'application/json' });
  FileSaver.saveAs(blob, "text_to_summarize.json");
}
```

Slika 5.9. *Callback funkcije dataReady i finished unutar trainModel funkcije*

Vlastiti model koji je dostupan na početnoj stranici aplikacije, i koji detektira medicinske slučajeve, istreniran je za tri potencijalna medicinska slučaja. Model detektira ako se osoba na slici drži za glavu, za prsa ili za trbuh, te se na temelju detektiranog položaja tijela korisniku prikazuju upute o medicinskom slučaju i kako postupiti u takvoj situaciji.

5.3. Pregled programskog rješenja – poslužiteljska strana

Poslužitelj je napisan na način da koristi REST arhitekturu, što znači da koristi HTTP metode na pojedinim krajnjim točkama, te se na temelju krajnje točke i HTTP metode izvrši neka *callback* funkcija definirana u kontrolerima. Popis svih dostupnih ruta nalazi se u tablici 5.2.

Tablica 5.2. *Pregled dostupnih ruta poslužitelja*

HTTP METODA	RUTA	OPIS RUTE
POST	/user/signup	Registracija korisnika
POST	/user/login	Prijava korisnika
POST	/trainData/new	Stvaranje novog skupa podataka za treniranje
GET	/trainData/list	Dohvaćanje svih korisničkih podataka za treniranje modela
PUT	/trainData/edit	Uređivanje postojećeg skupa podataka za treniranje
DELETE	/trainData/remove	Brisanje postojećeg skupa podataka za treniranje
GET	/model/list	Dohvaćanje svih korisničkih istreniranih modela
POST	/model/new	Stvaranje novog istreniranog modela
DELETE	/model/remove	Brisanje postojećeg istreniranog modela

Za stvaranje korisničkih računa i prijavu korisnika koriste se dvije rute: POST */signup* i POST */login* koje su prikazane na slici 5.10.

```
const express = require('express');
const router = express.Router();

const UserController = require('../controllers/UserController');
const UserControllerPolicy = require('../policies/UserControllerPolicy');

router.post('/login', UserController.login);
router.post('/signup', UserControllerPolicy.signup, UserController.signup);

module.exports = router;
```

Slika 5.10. *Rute za prijavu i registraciju korisnika*

Kada na neku od ruta stigne POST HTTP zahtjev, aktiviraju se metode definirane u *UserController* datoteci, ili u slučaju registracije, poziva se i posredna funkcija (engl. *middleware*) definirana u *UserControllerPolicy* datoteci.

U slučaju prijave, aktivira se *login* metoda prikazana na slici 5.11 koja provjerava postoji li korisnik u bazi podataka. U slučaju da ne postoji, poslužitelj vraća grešku, a u slučaju da postoji, provjerava se podudarnost lozinke. U slučaju da se lozinka u bazi i lozinka koju korisnik šalje podudaraju, poslužitelj vraća HTTP status kod 200 zajedno s JWT žetonom i korisničkim imenom korisnika, a u slučaju krive lozinke, poslužitelj vraća grešku.

```
async login (req, res) {
  try {
    const { username, password } = req.body;
    const user = await User.findOne({
      username
    });

    if (!user) {
      return res.status(403).send({
        error: 'Invalid login data.'
      });
    }
    const isPasswordValid = await user.comparePassword(password);
    if (!isPasswordValid) {
      return res.status(403).send({
        error: 'Invalid login data.'
      });
    }
    const { username: savedUsername } = user.toJSON();
    res.send({
      user: {username: savedUsername},
      token: jwtSignUser({ username: savedUsername })
    });
  } catch (err) {
    res.status(500).send({
      error: 'An error has occured. Try again later.'
    });
  }
},
```

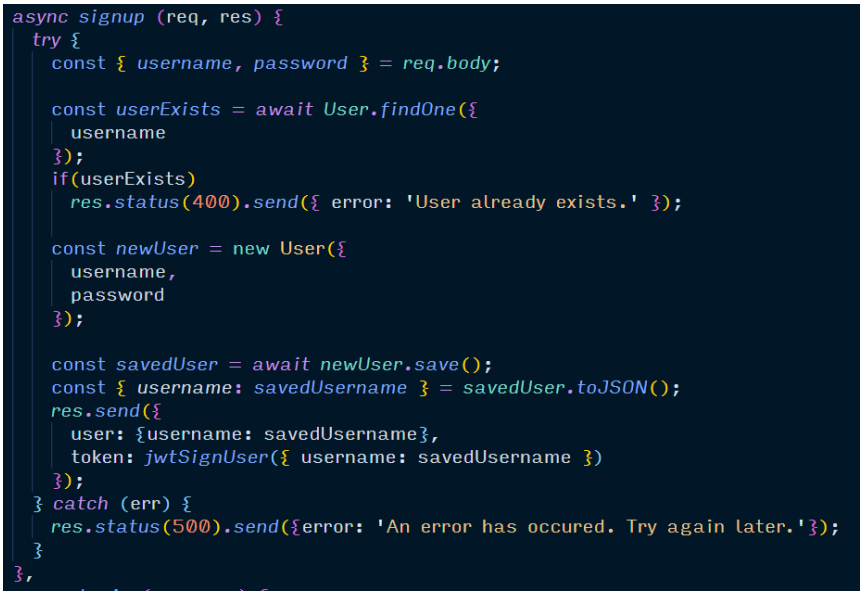
Slika 5.11. *Metoda za prijavu korisnika*

U slučaju registracije novog korisnika, prije poziva metode *signup* u *UserController* datoteci, poziva se *signup* metoda, čiji je dio programskog koda prikazan na slici 5.12, u *UserControllerPolicy* datoteci, gdje se pomoću *npm* paketa *Joi* provjeravaju uneseni podaci, kao što je duljina korisničkog imena i lozinke. Korisničko ime mora biti niz alfanumeričkih znakova, minimalne duljine 3, a maksimalne duljine 30 znakova, a za lozinku vrijede ista pravila, uz to da je duljina lozinke minimalno 6, a maksimalno 32 znaka.

```
signup (req, res, next) {
  const schema = Joi.object({
    username: Joi.string().alphanum().min(3).max(30).required(),
    password: Joi.string().regex(
      new RegExp('^[a-zA-z0-9]{6,32}$')
    ).min(6).max(32).required(),
  });
```

Slika 5.12. *Posredna funkcija za provjeru podataka pri registraciji*

U slučaju da pravila nisu zadovoljena, vraća se greška, a ako jesu, poziva se funkcija *next*, te se prelazi na *signup* metodu unutar kontrolera, prikazanu na slici 5.13.

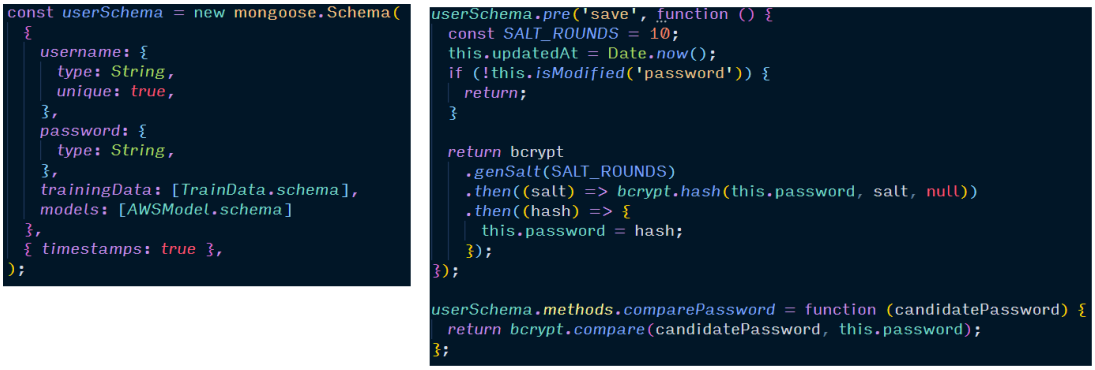


```
async signup (req, res) {
  try {
    const { username, password } = req.body;

    const userExists = await User.findOne({
      username
    });
    if(userExists)
      res.status(400).send({ error: 'User already exists.' });

    const newUser = new User({
      username,
      password
    });

    const savedUser = await newUser.save();
    const { username: savedUsername } = savedUser.toJSON();
    res.send({
      user: {username: savedUsername},
      token: jwtSignUser({ username: savedUsername })
    });
  } catch (err) {
    res.status(500).send({error: 'An error has occured. Try again later.'});
  }
},
```

Slika 5.13. *Metoda za registraciju korisnika*

U *signup* metodi unutar *UserController* kontrolera provjerava se postoji li u bazi podataka korisnik s unesenim korisničkim imenom. Ako postoji, poslužitelj vraća grešku, a ako ne, korisnika se sprema u bazu podataka.

Svaka interakcija s bazom podataka koristi *npm* paket *Mongoose* koji olakšava komunikaciju na način da pruža metode za spremanje, pretragu, brisanje i ažuriranje podataka u bazi. Kako bi se mogla vršiti interakcija na taj način, za svaku zbirku potrebno je napisati *schemu* po kojoj će se dokumenti unutar te zbirke spremati. Za potrebe spremanja i dohvaćanja korisnika definirana je *User schema* prikazana na slici 5.14. Prema *schemi*, svaki korisnik ima jedinstveno korisničko ime koje je tipa *String*, lozinku koja je također tipa *String*, polje *trainingData* koje je ugrađeni (engl. *embedded*) dokument sa vlastitom *schemom*, a koje predstavlja korisničke podatke za trening modela, te polje *models* koje je također ugrađeni dokument, a predstavlja istrenirane modele tog korisnika. Osim definicije *scheme*, za korisnika su definirane i pomoćne metode koje služe za kodiranje i dekodiranje lozinke koristeći *npm* paket *bcrypt* pomoću kojeg se lozinke pohranjuju na siguran način.



Slika 5.14. *Schema korisnika (lijevo) i pomoćne metode korisnika (desno)*

Osim spremanja i dohvaćanja informacija o samom korisniku, potrebno je i omogućiti metode za spremanje, dohvaćanje, brisanje i uređivanje trening podataka za buduće korisnikove modele, a rute koje to omogućuju prikazane su na slici 5.15.

```
router.get('/list', isAuthenticated, TrainDataController.list);
router.post('/new', isAuthenticated, TrainDataController.new);
router.put('/edit', isAuthenticated, TrainDataController.edit);
router.delete('/remove', isAuthenticated, TrainDataController.remove);
```

Slika 5.15. *Rute za trening podatke*

Svaka od ruta prije pokretanja metode u kontrolerima prvo poziva posrednu funkciju *isAuthenticated* kojom se provjerava valjanost JWT žetona u zaglavlju zahtjeva na temelju kojeg se utvrđuje autentičnost zahtjeva, tj. provjerava se šalje li zahtjev autorizirana osoba koja se nalazi u bazi podataka. U tu svrhu koristi se *npm* paket *passport*, koji dolazi sa preddefiniranim strategijama za provjeru nadolazećeg zahtjeva. U slučaju da je JWT nevaljan ili nepostojeći, zahtjev se odbija, a u slučaju da je JWT valjan, on se dekodira, informacija unutar JWT se stavi unutar zahtjevau obliku *user* objekta i aktivira se tražena metoda u *TrainDataController* kontroleru.

TrainDataController sadrži četiri metode: *list*, *new*, *edit* i *remove*. *List* metoda služi za pretragu trening podataka za nekog korisnika, a kod metode nalazi se na slici 5.16. Kao što je vidljivo na slici, prvo se iz zahtjeva uzme *user* objekt koji je dobiven preko *isAuthenticated* posredne funkcije, te se u bazi podataka pronađe korisnik s korisničkim imenom korisnika koji je zatražio dohvaćanje podataka za trening. Ako je korisnik pronađen, poslužitelj vraća HTTP status kod 200 zajedno s podacima za trening tog korisnika, a u slučaju da korisnik nije pronađen ili se dogodila neka neočekivana greška, vraća se HTTP status kod 400, odnosno 500 zajedno s porukom o grešci.

```
async list(req, res) {
  try {
    const { user } = req;
    const dbUser = await User.findOne({
      username: user.username,
    });

    if (dbUser) {
      res.status(200).send({ trainingData: dbUser.trainingData });
    } else res.status(400).send({ error: 'Error fetching data. Try again later.' });
  } catch (err) {
    res.status(500).send({ error: 'Error fetching data. Try again later.' });
  }
},
```

Slika 5.16. *List metoda unutar TrainDataController kontrolera*

Metoda *new*, čiji je programski kod prikazan na slici 5.17, odgovorna je za spremanje trening podataka u bazu podataka. Kao i kod *list* metode, prvo se pomoću *user* objekta u zahtjevu nađe korisnik s odgovarajućim korisničkim imenom. Svaki skup podataka za trening ima vlastiti naziv, stoga je prije spremanja potrebno provjeriti postoji li za pronađenog korisnika već skup podataka s nazivom unutar zahtjeva. Ukoliko ne postoji, pomoću *Mongoose* metoda se stvori novi *TrainData* objekt koji se doda u *trainingData* polje koje postoji u *schemi* korisnika, te poslužitelj kao odgovor vraća HTTP status kod 200 zajedno s novostvorenim trening podacima. U slučaju bilo kakve greške pri spremanju ili ukoliko skup podataka s nazivom već postoji, poslužitelj vraća odgovarajući HTTP status kod i poruku o grešci.

```
async new(req, res) {
  try {
    const { trainingData } = req.body;
    const { username } = req.user;

    const dbUser = await User.findOne({
      username,
    });

    try {
      const existingTrainingData = await User.findOne({ username, 'trainingData.name': trainingData.name });
      if (!existingTrainingData) {
        const trainData = new TrainData(trainingData);
        dbUser.trainingData.push(trainData);
      } else {
        res.status(400).send({ error: `Training data with the name ${trainingData.name} already exists!` });
        return;
      }
    } catch (error) {
      res.status(400).send({ error: `Training data with the name ${trainingData.name} already exists!` });
    }
    dbUser.save(function (error) {
      if (error) res.status(500).send({ error });
    });

    res.status(200).send({ trainingData: dbUser.trainingData, username: dbUser.username });
  } catch (error) {
    res.status(500).send({ error });
  }
},
```

Slika 5.17. *New metoda unutar TrainDataController kontrolera*

Metode *remove* i *edit*, prikazane na slici 5.18, služe za brisanje, odnosno uređivanje trening podataka nekog korisnika. Metode zahtijevaju slanje identifikacijskog broja skupa trening

podataka koji je potrebno ukloniti ili urediti. Metoda *remove* koristi MongoDB *pull* metodu kako bi se pronađen skup podataka uklonio iz *trainingData* polja korisnika, dok se *edit* metodom stari podaci zamijene novim vrijednostima. Obje metode kao rezultat vraćaju status kod 200 u slučaju uspješnog izvršetka zajedno s podacima o uklonjenom ili uređenom skupu, ili vraćaju status kod 400 ili status kod 500 zajedno s greškom u slučaju da je do nje došlo.

```
async remove(req, res) {
  try {
    const { user } = req;
    const { trainingDataId } = req.query;

    const { trainingData } = await User.findById(new ObjectID(user._id));
    const sample = trainingData.find((s) => s._id.toString() === trainingDataId);
    const dbUser = await User.findOneAndUpdate(
      {
        username: user.username,
      },
      {
        $pull: {
          trainingData: { _id: trainingDataId },
        },
      },
    );

    res.status(200).send({ trainingData: sample, user: dbUser });
  } catch (error) {
    res.status(500).send({ error: 'Error deleting data. Try again later.' });
  }
},
```

```
async edit(req, res) {
  try {
    const { trainingData } = req.body;
    const { username } = req.user;

    const dbUser = await User.findOne({
      username,
    });

    let updated;
    try {
      dbUser.trainingData = trainingData;
      updated = await dbUser.save();

    } catch (error) {
      res.status(400).send({ error: 'Error saving data. Try again later.' });
    }
    res.status(200).send({ trainingData: updated, username: dbUser.username });
  } catch (error) {
    res.status(500).send({ error });
  }
},
```

Slika 5.18. *Remove metoda (lijevo) i edit metoda (desno) unutar TrainDataController kontrolera*

Kako bi komunikacija s bazom podataka bila uspješna, bilo je potrebno definirati i model, tj. *schemu* po kojem će se trening podaci spremati u bazi podataka. Izrađeni model nalazi se na slici 5.19, a sastoji se od naziva podatkovnog skupa koji je tipa *String*, polja tekstova *texts* koji će se sažimati, te samog skupa podatka *data* prema kojem će se istrenirati neuronska mreža za detektiranje položaja tijela. Svaka stavka unutak polja *texts* je objekt koji je povezan s nekim izlazom neuronske mreže pomoću atributa *outputName*, te sadrži i tekst kojeg treba sažeti. Polje *data* sadrži podatke za trening. Svaki ulazni podatak sadrži dva objekta u obliku *xs* i *ys*, gdje *xs* predstavljaju ulazne vrijednosti kojih ukupno ima 34, a *ys* predstavlja oznaku izlaza za taj trening podatak.

```
const trainDataSchema = new mongoose.Schema({
  name: String,
  texts: [
    {
      outputName: String,
      text: String,
    },
  ],
  data: [
    {
      xs: {
        0: Number,
        1: Number,
        2: Number,
        // ...
        33: Number,
      },
      ys: {
        0: String,
      },
    },
  ],
});
```

Slika 5.19. *Schema skupa podataka za trening*

Osim spremanja korisnika i skupova trening podataka, omogućeno je i spremanje istreniranih modela. Svaki model sprema se u Amazon S3 *bucket*, a podaci o spremljenim datotekama, kao što su naziv modela, ključevi datoteka i njihova lokacija na S3 *bucketu* spremaju se u bazu podataka prema *schemi* koja se nalazi na slici 5.20.

```
const awsModelSchema = new mongoose.Schema({
  name: String,
  modelKey: String,
  metaKey: String,
  weightsKey: String,
  modelLocation: String,
  metaLocation: String,
  weightsLocation: String,
  textKey: String,
  textLocation: String,
});
```

Slika 5.20. *Schema korisničkih modela strojnog učenja*

Rute preko kojih se komunicira s bazom i Amazon S3 *bucketom* prikazane su na slici 5.21, a one su *list*, *new* i *remove*.

```
router.get('/list', isAuthenticated, ModelController.list);
router.post('/new', isAuthenticated, upload.array('files'), ModelController.new);
router.delete('/remove', isAuthenticated, ModelController.remove);
```

Slika 5.21. *Rute za modele strojnog učenja*

Metoda *list* u *ModelController* kontroleru vrlo je slična metodi *list* u *TrainDataController* kontroleru. Kao što je prikazano na slici 5.22, u bazi podataka se pronađu modeli onog korisnika koji šalje zahtjev za dohvaćanjem popisa modela, te se podaci o tim modelima vrate kao odgovor poslužitelja.

```
async list(req, res) {
  try {
    const { user } = req;
    const dbUser = await User.findOne({
      username: user.username,
    });

    if (dbUser) {
      res.status(200).send({ models: dbUser.models });
    } else res.status(400).send({ error: 'Error fetching data. Try again later.' });
  } catch (err) {
    res.status(500).send({ error: 'Error fetching data. Try again later.' });
  }
},
```

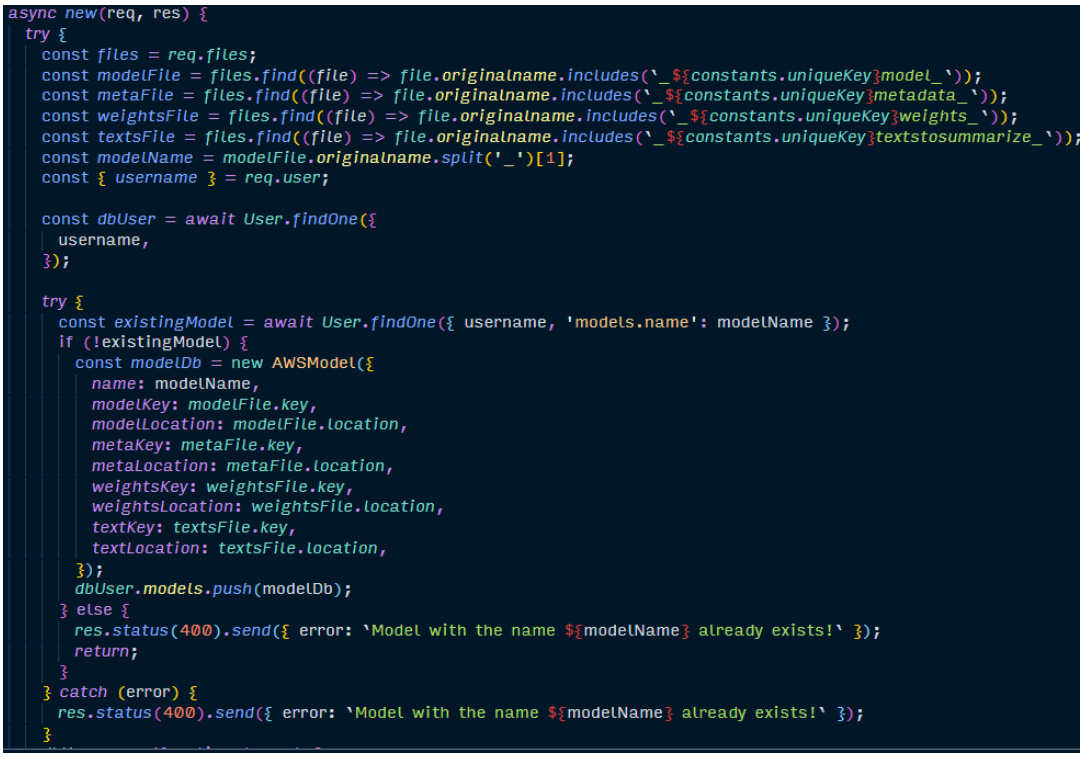
Slika 5.22. *List metoda unutar ModelController kontrolera*

Kod spremanja novog modela, prije spremanja podataka o modelu u MongoDB bazu podataka metodom *new*, osim posredne funkcije *isAuthenticated* dodana je i druga posredna funkcija *upload*, čiji je programski kod prikazan na slici 5.23, a služi za prijenos datoteka na Amazon S3.

Korištenjem AWS SDK (*Amazon Web Services Software Development Kit*) definiraju se konfiguracijske postavke kojima se omogućava pristup AWS korisničkom računu. Zatim se instancira *aws.S3* objekt koji se, zajedno s postavkama o nazivu S3 *bucketa*, tipom podataka datoteke i jedinstvenim ključem datoteke, predaje funkciji *multerS3*, koja je dodana kao *npm* paket, a služi za prijenos datoteka na Amazon S3.

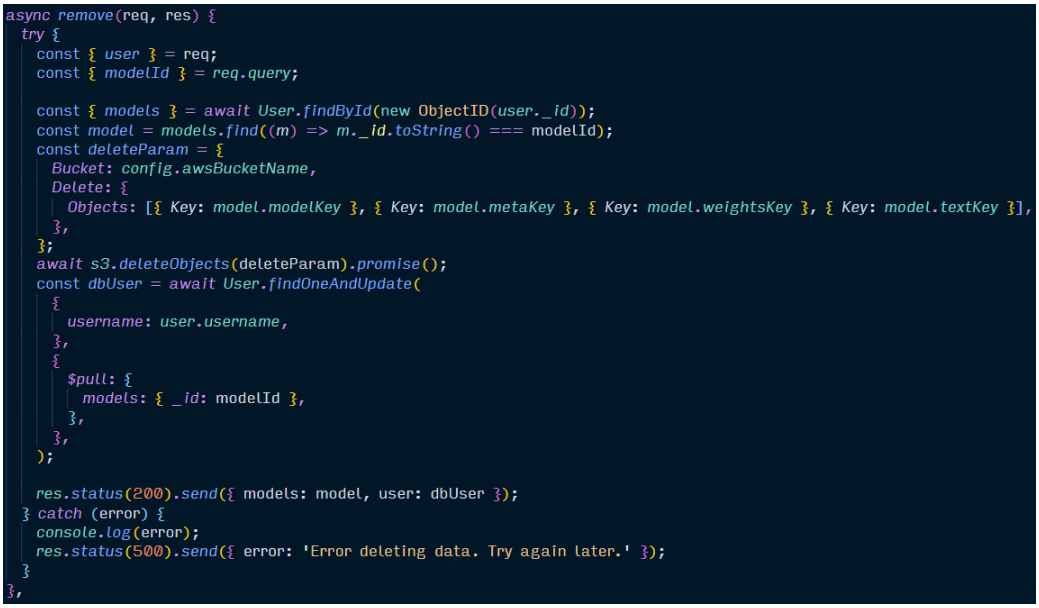
Slika 5.23. *Posredna funkcija upload*

Nakon prijenosa datoteka, u *new* funkciji, prikazanoj na slici 5.24, u zahtjevu se mogu dobiti informacije o prenesenim datotekama preko *req.files* polja podataka. U polju se identificiraju različite datoteke pomoću JavaScript *find* metode na način da se provjeri sadržava li naziv datoteke određeni niz znakova. Tako će, npr. svaka model datoteka u nazivu sadržavati poseban ključ nakon kojeg slijedi riječ „*model*“, ili će datoteka s težinama sadržavati poseban ključ nakon kojeg slijedi riječ „*weights*“ itd. Nakon identificiranja svake datoteke, iz datoteke u kojoj je spremljen model očita se i naziv samog modela. Nakon toga, potrebno je provjeriti postoji li u bazi podataka, za korisnika koji šalje datoteke, već taj naziv modela. Ukoliko taj model ne postoji, informacije o svim datotekama se spremaju u bazu podataka, a poslužitelj kao odgovor vraća spremljeni model. U slučaju da je došlo do bilo kakve greške, poslužitelj vraća odgovarajući HTTP status kod zajedno s informacijama o grešci.



Slika 5.24. *New metoda unutar ModelController kontrolera*

Osim dodavanja novih i dohvaćanja postojećih modela, postoji i mogućnost brisanja modela, za što služi *remove* metoda prikazana na slici 5.25. Metoda kao parametar prima identifikacijski broj modela kojeg je potrebno obrisati. Nakon pronalaska korisnika koji šalje zahtjev za brisanjem, identifikacijski broj modela se koristi kako bi se taj model pronašao u bazi podataka, nakon čega se prvo datoteke modela izbrišu u Amazon S3 *bucketu*, a zatim se model izbriše i u bazi podataka.

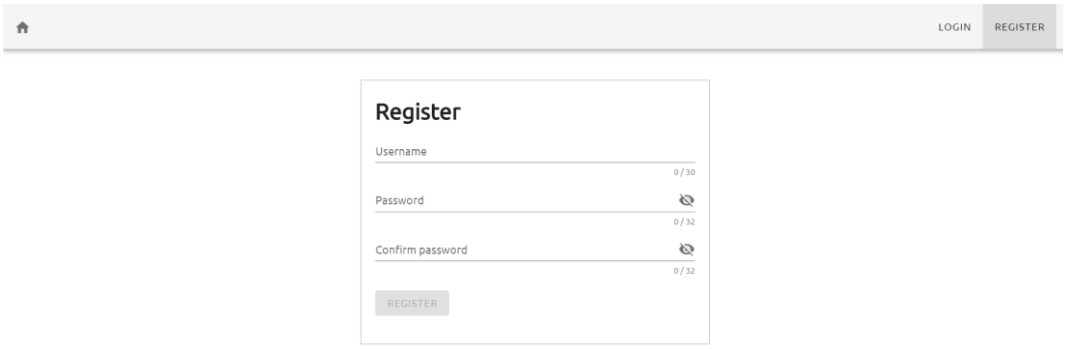


Slika 5.25. *Remove metoda unutar ModelController kontrolera*

5.4. Pregled programskog rješenja – klijentska strana

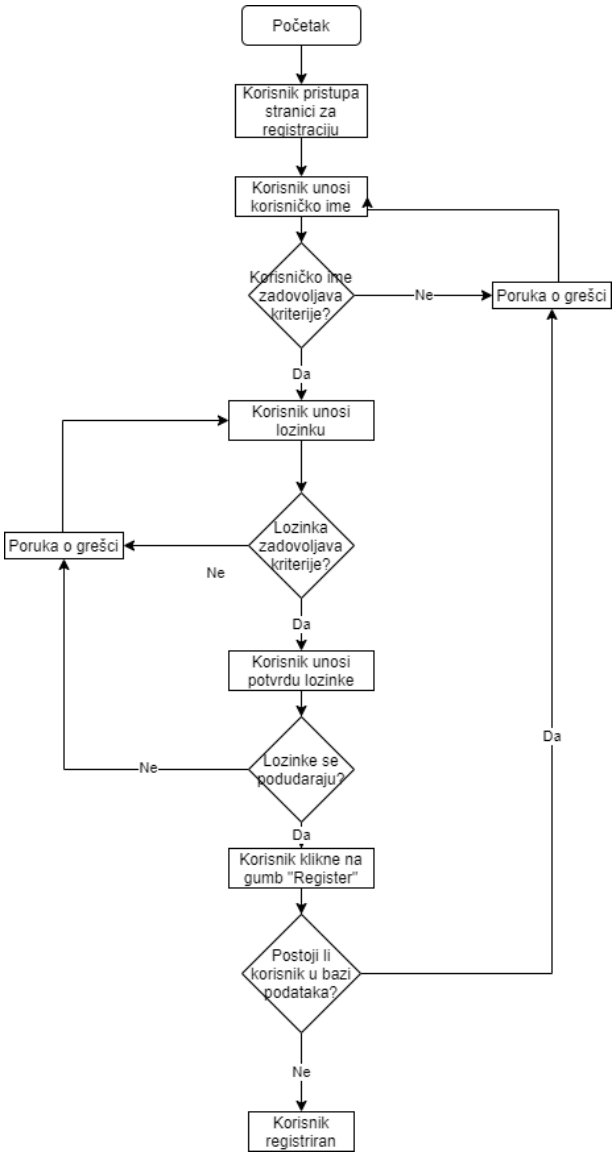
5.4.1. Prijava i registracija u sustav

Prije korištenja naprednih mogućnosti aplikacije, kao što je treniranje modela na vlastitom skupu podataka, spremanje vlastitih modela i spremanje skupa podataka za trening, potrebno je napraviti korisnički račun. Kako bi se korisnički račun napravio, potrebno je kliknuti na gumb „*Register*“ u navigacijskom oknu aplikacije, te se tada otvori stranica prikazana na slici 5.26.



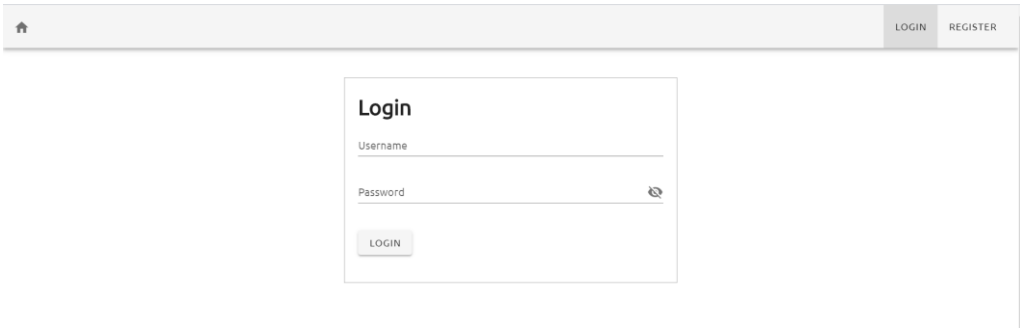
Slika 5.26. *Registracija korisnika*

Od korisnika se zahtijeva unos korisničkog imena, lozinke i potvrde lozinke. Forma sadrži validaciju kojom se provjerava duljina korisničkog imena, duljina lozinke i podudarnost lozinke s njenom potvrdom. Kada su sva potrebna polja pravilno ispunjena, gumb „*Register*“ ispod forme postane dostupan, te se korisnika spremi u sustav, kako je prikazano u dijagramu toka na slici 5.27. Nakon registracije, aplikacija korisnika automatski prijavljuje i šalje na početnu stranicu.



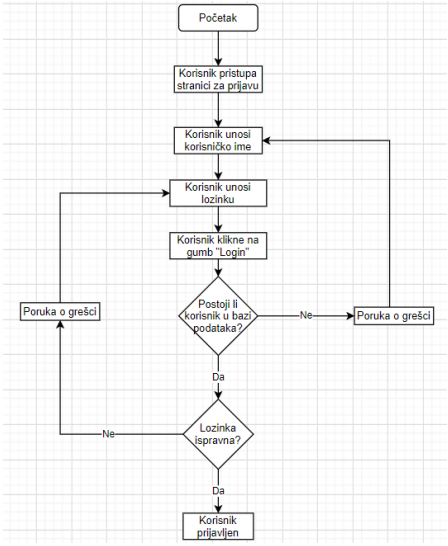
Slika 5.27. *Dijagram toka stranice za registraciju*

Registrirani korisnici imaju mogućnost prijave u sustav klikom na gumb „*Login*“ u navigacijskom oknu, kojim se otvara stranica prikazana na slici 5.28.



Slika 5.28. *Prijava korisnika*

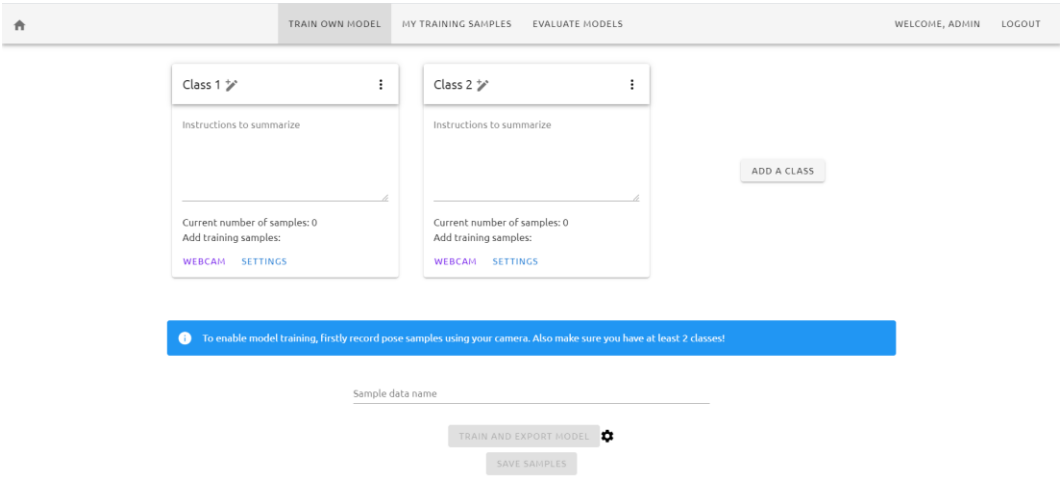
Korisnik se prijavljuje unosom korisničkog imena i lozinke koji su uneseni prilikom registracije. Nakon uspješne prijave, korisnika se preusmjerava na početnu stranicu aplikacije. Cijeli proces prijave korisnika prikazan je na dijagramu toka na slici 5.29.



Slika 5.29. *Dijagram toka stranice za prijavu*

5.4.2. Treniranje vlastitog modela

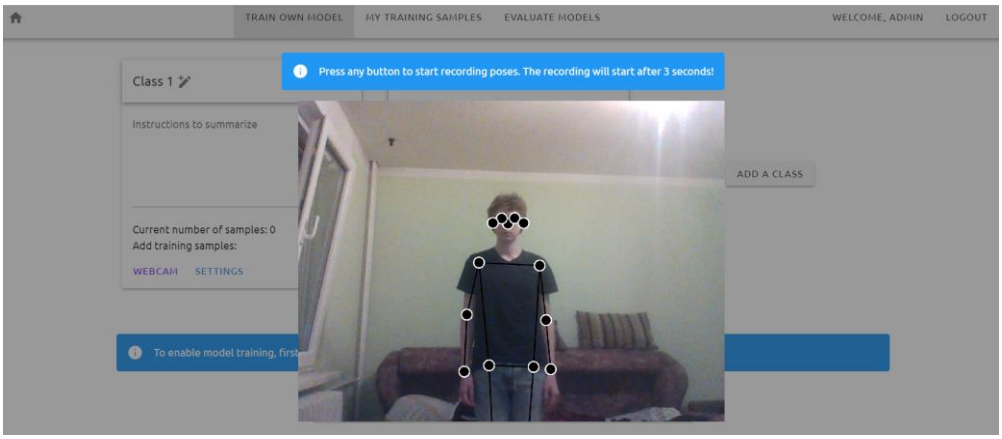
Prijavom u sustav, korisniku se u navigacijskom oknu pružaju dodatne mogućnosti prikazane na slici 5.30, a to su treniranje vlastitog modela na nekom skupu podataka, pregled spremljenih skupova podataka, te evaluacija i spremanje vlastitih modela.



Slika 5.30. *Sučelje stranice za treniranje vlastitog modela*

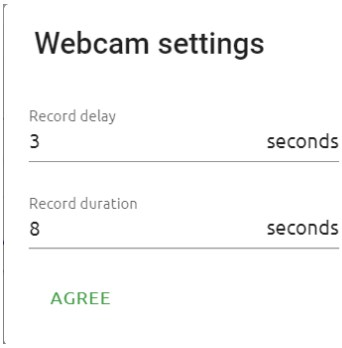
Stranica za treniranje vlastitog modela daje mogućnost korisniku da doda željeni broj klasa kojima može urediti naziv, te dodati tekst koji je potrebno sažeti za svaku klasu. Nakon definiranja naziva klasa, unutar kartice koje vizualno predstavljaju pojedine klase, korisnik može odabrati opciju „*Webcam*“, kojom se aktivira sučelje prikazano na slici 5.31. Nakon što se sučelje potpuno učita,

korisnik može pritisnuti tipku „*Esc*“ za prestanak snimanja, ili bilo koju drugu tipku kako bi se snimanje započelo.



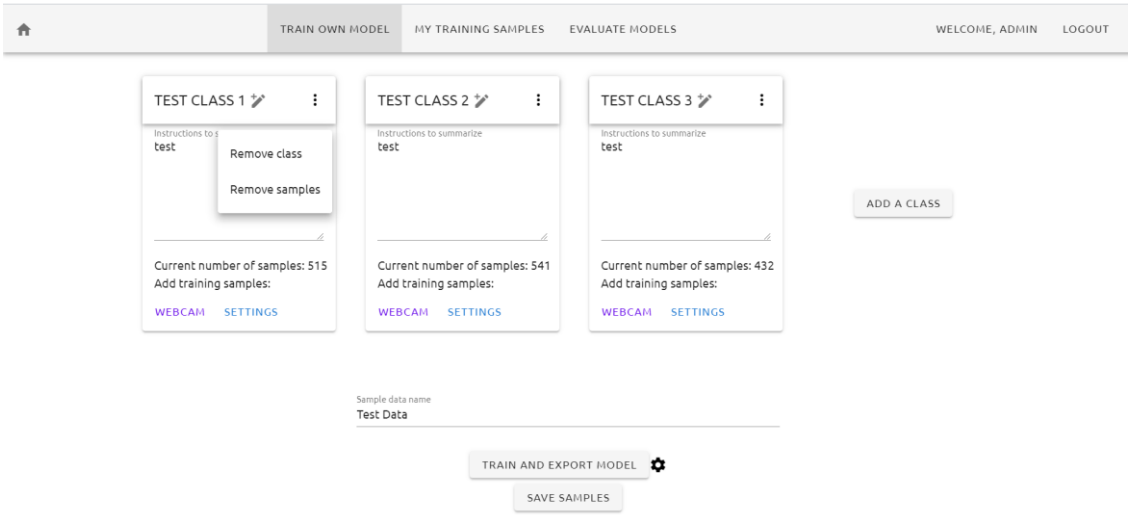
Slika 5.31. *Sučelje za snimanje položaja tijela*

Korisnik također može odabrati opciju „*Settings*“, kojom se otvara dijalog prikazan na slici 5.32, a u kojem je moguće podesiti duljinu odgode snimanja nakon pritiska tipke, te ukupno trajanje snimanja nekog položaja.



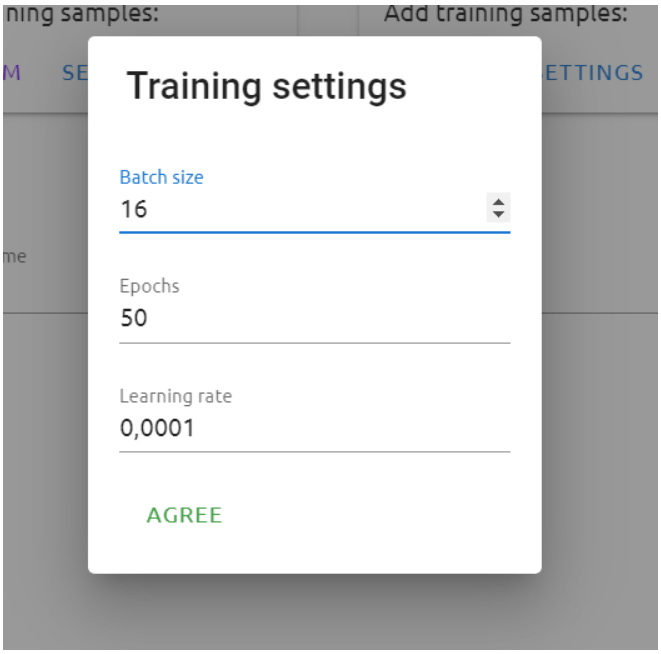
Slika 5.32. *Postavke snimanja*

Nakon snimanja uzoraka za svaku klasu, kao što je prikazano slikom 5.33, za svaku klasu se ažurira trenutni broj uzoraka. Klase je moguće preimenovati klikom na ikonu olovke, a klikom na ikonu tri točke pored naziva klase moguće je ukloniti klasu ili ukloniti uzorke te klase. Prije spremanja skupa uzoraka, potrebno je skupu dodati ime, a klikom na gumb „*Save samples*“, uzorci se spremaju u bazu podataka, te ih je moguće brisati iz same baze podataka ili naknadno uređivati.



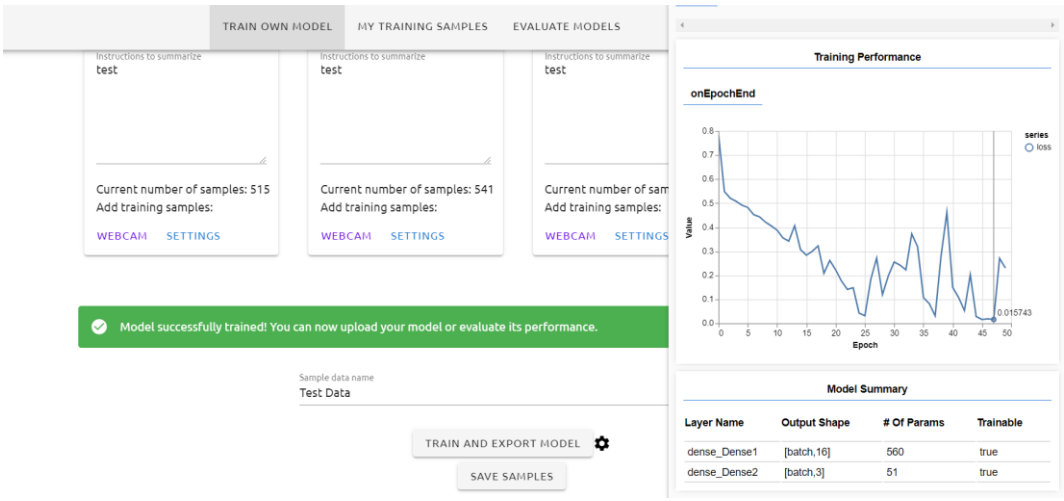
Slika 5.33. *Prikupljeni podaci za trening modela*

Nakon prikupljanja dovoljnog broja uzoraka, moguće je istrenirati model klikom na gumb „*Train and export model*“. Prije samog treninga modela, moguće je i postaviti parametre prikazane na slici 5.34, koji su veličina serije, broj epoha i stopa učenja modela.



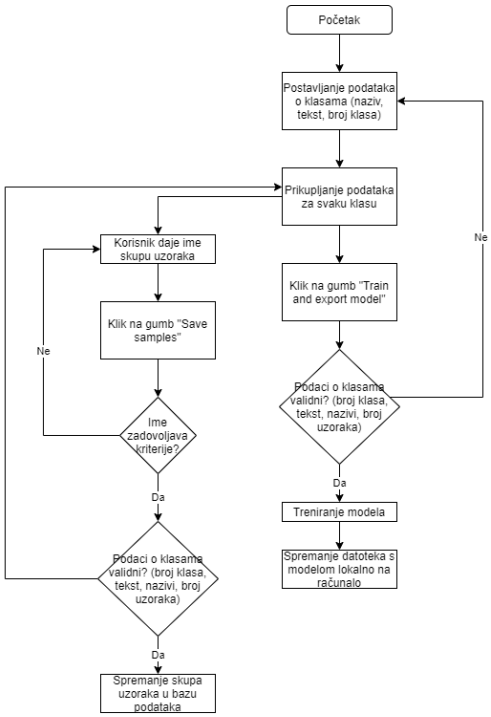
Slika 5.34. *Postavke za trening modela*

Nakon klika na gumb „*Train and export model*“, na stranici se pojavljuje graf prikazan na slici 5.35, na kojem je moguće vidjeti broj epoha i vrijednost funkcije gubitka. Osim grafa, prikazani su i slojevi neuronske mreže zajedno s brojem parametara za neki sloj.



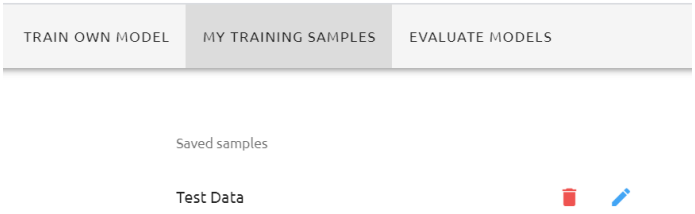
Slika 5.35. *Postavke za trening modela*

Nakon što se završi treniranje modela, generiraju i datoteke potrebne za uspješno učitavanje modela. Općeniti pregled postupka treniranja modela, spremanja modela lokalno na računalo i spremanja skupa podataka dan je na slici 5.36.



Slika 5.36. *Dijagram toka stranice za treniranje vlastitog modela*

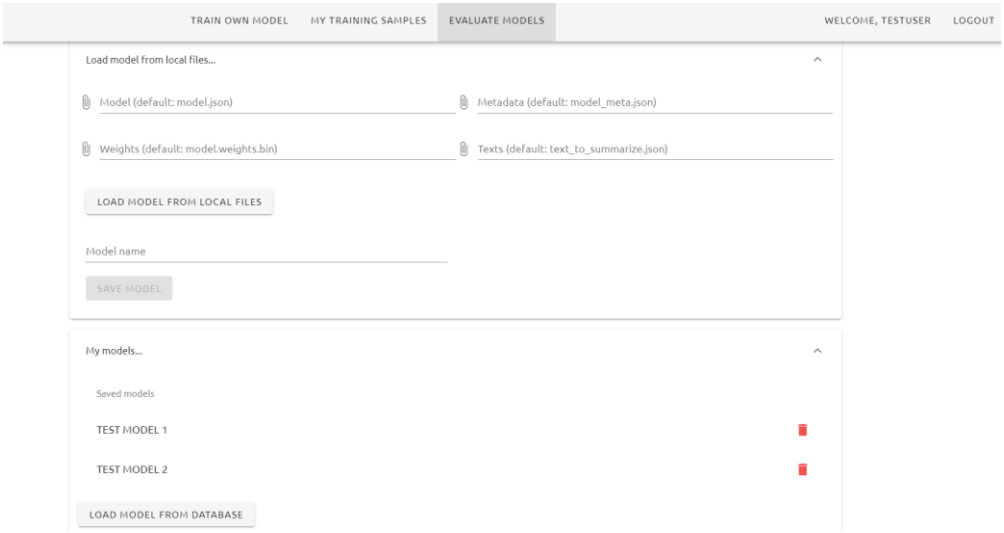
Sve skupove podataka za trening moguće je pogledati klikom na gumb „*My training samples*“, kojim se korisnika navigira na stranicu prikazanu na slici 5.37. Korisnik klikom na odgovarajuće ikone može uređivati ili brisati spremljene skupove podataka.



Slika 5.37. *Pregled skupova podataka za trening*

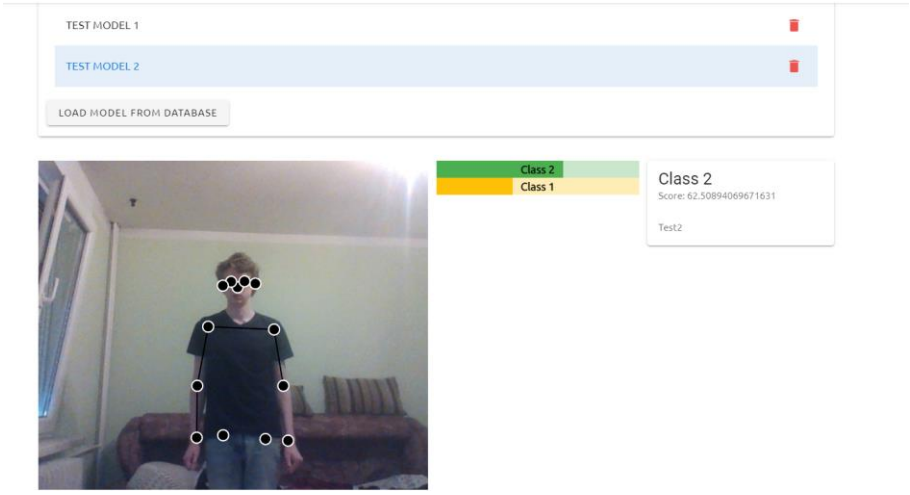
5.4.3. Ispitivanje vlastitih modela i početna stranica

Mogućnost ispitivanja i testiranja modela spremljenih na vlastitom računalu ili modela spremljenih u bazi podataka dostupna je klikom na „*Evaluate models*“ u navigacijskom oknu korisničkog sučelja, nakon čega se otvara stranica prikazana na slici 5.38. Stranica nudi mogućnost prijenosa datoteka generiranih treniranjem modela. Svaku datoteku potrebno je posebno prenijeti, a posebno je važno da se datoteke nazivaju svojim početnim vrijednostima, jer inače učitavanje modela nije moguće, i sustav će javljati grešku. Osim učitavanja lokalno spremljenih datoteka, njih je moguće i pohraniti u bazu podataka i na Amazon S3, a to je moguće unosom imena modela u predviđeni tekstualni okvir, nakon čega se omogućava funkcionalnost gumba „*Save model*“, klikom na koji se model pohranjuje.



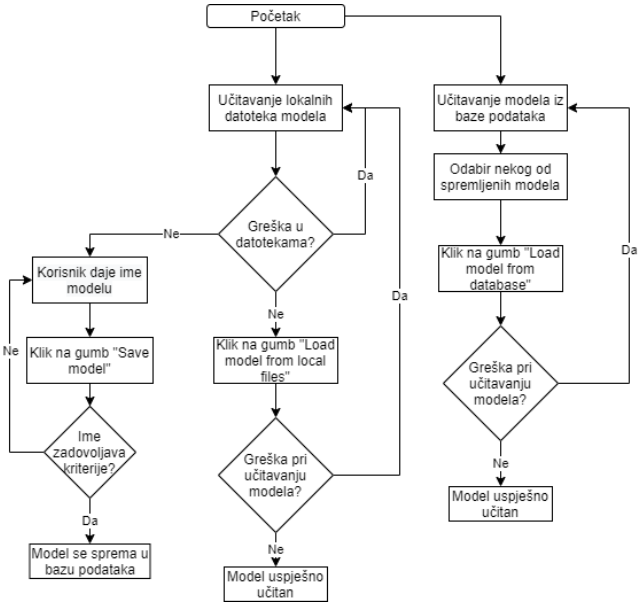
Slika 5.38. *Sučelje stranice „Evaluate models“*

Svi pohranjeni modeli prikazuju se unutar „*My models*“ panela, gdje ih je moguće brisati, ili je moguće označiti neki od modela te ga učitati. Nakon učitavanja svih datoteka ili odabira modela, moguće je kliknuti na gumb „*Load model from local files*“, odnosno gumb „*Load model from database*“, nakon čega će se na sučelju prikazati elementi prikazani na slici 5.39.



Slika 5.39. *Detektiranje i klasifikacija položaja tijela*

Na sučelju se prikazuje video osobe na slici zajedno s iscrtanim ključnim točkama i kosturom, a sa strane se nalaze sve moguće klase modela, sa vizualnim prikazom uvjerenosti modela u neki položaj u obliku ispunjenosti trake, gdje zelena traka predstavlja klasu koja ima najveću uvjerenost modela, a žute trake predstavljaju sve druge klase. Pored traka nalazi se i panel koji prikazuje trenutnu klasu s najboljim rezultatom, zajedno s sažetim tekstom unesenim za tu klasu. Korištenje stranice opisano je dijagramom toka na slici 5.40.



Slika 5.40. *Dijagram toka stranice za ispitivanje vlastitih modela*

Učitavanje modela i klasifikacija rezultata vrši se pomoću funkcije *loadModel*, koja je po strukturi vrlo slična funkciji za prikupljanje podataka *collectData* i funkciji za treniranje modela *trainModel*. Funkcija *loadModel* također je napisana koristeći ml5.js i p5.js, a sastoji se od p5.js

funkcija *setup* i *draw*, *callback* funkcija *neuralNetworkLoaded*, *gotResult*, *gotPoses* i *modelLoaded*, te pomoćne funkcije *classifyPose*.

U *setup* funkciji, čiji je programski kod prikazan na slici 5.41, kao i programski kod funkcije *collectData*, stvara se HTML *canvas* element u kojem će se prikazivati video, te se stvara HTML *Media Element* metodom *createCapture* koja će pokrenuti video. Učitava se PoseNet model, te se postavljaju *callback* funkcije koje će se izvršiti kada se PoseNet model učita i kada se detektira položaj tijela. Inicijalizira se neuronska mreža, te se, prema postavkama, učitavaju datoteke prenesene s osobnog računala ili se učitavaju datoteke pohranjene na Amazon S3 usluzi.

```
p5.setup = (_) => {
  canvas = p5.createCanvas(settings.videoWidth, settings.videoHeight);
  video = p5.createCapture(p5.VIDEO);
  video.hide();
  poseNet = ml5.poseNet(video, modelLoaded);
  poseNet.on('pose', gotPoses);

  let options = {
    inputs: 34,
    outputs: settings.numOfOutputs,
    task: 'classification',
    debug: true,
  };
  neuralNetwork = ml5.neuralNetwork(options);
  if(settings.loadFromLocalFiles) {
    neuralNetwork.load(settings.files, neuralNetworkLoaded);
  } else if (settings.loadFromDatabase) {
    const modelDetails = {
      model: settings.awsModel,
      metadata: settings.awsMetadata,
      weights: settings.awsWeights
    }
    neuralNetwork.load(modelDetails, neuralNetworkLoaded);
  }
};
```

Slika 5.41. *Setup funkcija unutar loadModel funkcije*

Unutar *draw* funkcije prikazanoj na slici 5.42, iscrtavaju se sve ključne točke zajedno s kosturom osobe na slici.

```
p5.draw = (_) => {
  p5.push();
  p5.translate(video.width, 0);
  p5.scale(-1, 1);
  p5.image(video, 0, 0, video.width, video.height);

  if (pose) {
    for (let i = 0; i < skeleton.length; i++) {
      let a = skeleton[i][0];
      let b = skeleton[i][1];
      p5.strokeWeight(2);
      p5.stroke(0);

      p5.line(a.position.x, a.position.y, b.position.x, b.position.y);
    }
    for (let i = 0; i < pose.keypoints.length; i++) {
      let x = pose.keypoints[i].position.x;
      let y = pose.keypoints[i].position.y;
      p5.fill(0);
      p5.stroke(255);
      p5.ellipse(x, y, 16, 16);
    }
  }
  p5.pop();
};
```

Slika 5.42. *Draw funkcija unutar loadModel funkcije*

Callback funkcija *neuralNetworkLoaded* daje poruku da je klasifikacija spremna, te poziva pomoćnu funkciju *classifyPose* koja služi za klasificiranje položaja tijela, kako je i prikazano na slici 5.43. Funkcija *classifyPose* provjerava postoji li neki položaj tijela osobe, te u tom slučaju koristi ključne točke detektirane na slici za klasifikaciju metodom *classify*. Nakon uspješne klasifikacije, poziva se *callback* funkcija *gotResult*. Ukoliko nije detektiran položaj tijela, funkcija za se klasifikaciju se ponovno poziva nakon 100 ms.

```
function neuralNetworkLoaded() {
  console.log('pose classification ready!');
  classifyPose();
}

function classifyPose() {
  if (pose) {
    let inputs = [];
    for (let i = 0; i < pose.keypoints.length; i++) {
      let x = pose.keypoints[i].position.x;
      let y = pose.keypoints[i].position.y;
      inputs.push(x);
      inputs.push(y);
    }
    neuralNetwork.classify(inputs, gotResult);
  } else {
    setTimeout(classifyPose, 100);
  }
}
```

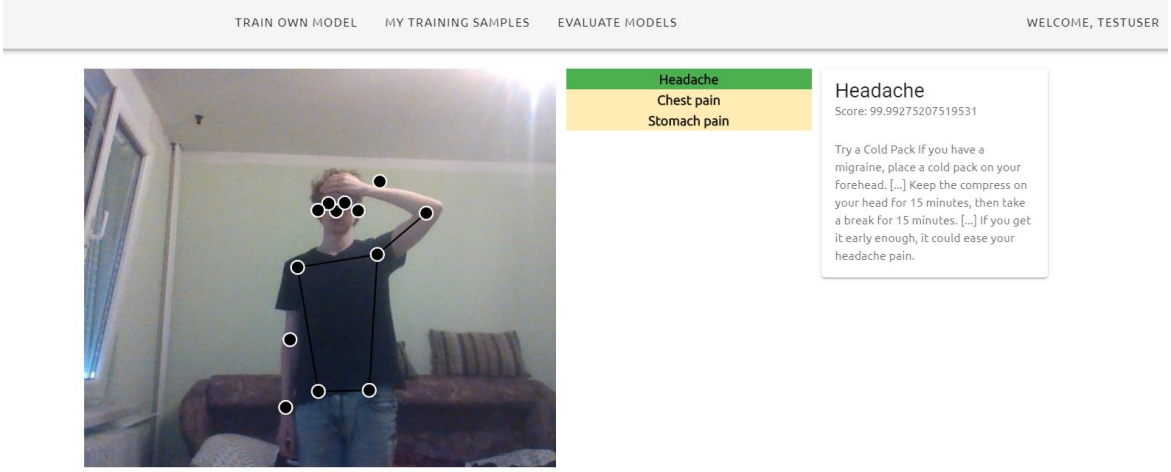
Slika 5.43. *Programski kod za klasifikaciju položaja osobe*

Callback funkcija *modelLoaded* ispisuje poruku o spremnosti PoseNet modela, *callback* funkcija *gotPoses* pohranjuje trenutne vrijednosti koordinata ključnih točaka i kostura tijela u varijable *pose* i *skeleton*, a *callback* funkcija *gotResult* množi vrijednosti uvjerenosti modela šalje rezultate klasifikacije Vue aplikaciji, kako je prikazano na slici 5.44.

```
function gotResult(error, results) {
  for (const result of results) {
    result.confidence *= 100;
  }
  sendResults(results);
  classifyPose();
}

function gotPoses(poses) {
  if (poses.length > 0) {
    pose = poses[0].pose;
    skeleton = poses[0].skeleton;
  }
}

function modelLoaded() {
  console.log('poseNet ready');
}
};
```

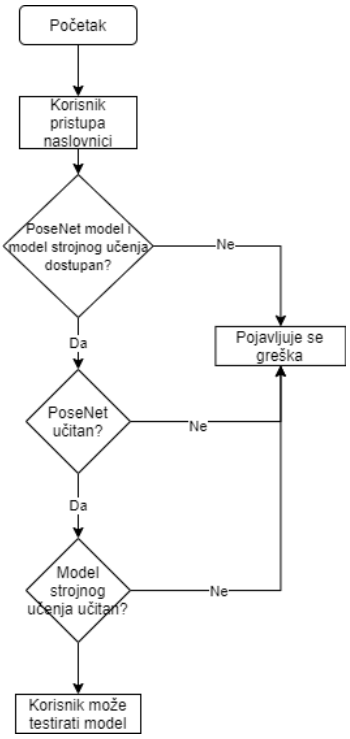
Slika 5.44. *Callback funkcije gotResult, gotPoses i modelLoaded unutar funkcije loadModel*

Naslovnica aplikacije, koja koristi preddefinirani model koji detektira hitne medicinske slučajeve, radi na vrlo sličan način kao i stranica za evaluaciju vlastitih modela, uz to da nije potrebna registracija i prijava za pristup stranici, ali i nije ponuđen izbor spremanja i učitavanja drugih modela, kao što je prikazano na slici 5.45.



Slika 5.45. *Naslovnica aplikacije*

Dijagram toka s funkcionalnostima koje su ponuđene na naslovnoj stranici prikazan je na slici 5.46.



Slika 5.46. *Dijagram toka naslovne stranice*

6. TESTIRANJE APLIKACIJE I MODELA STROJNOG UČENJA

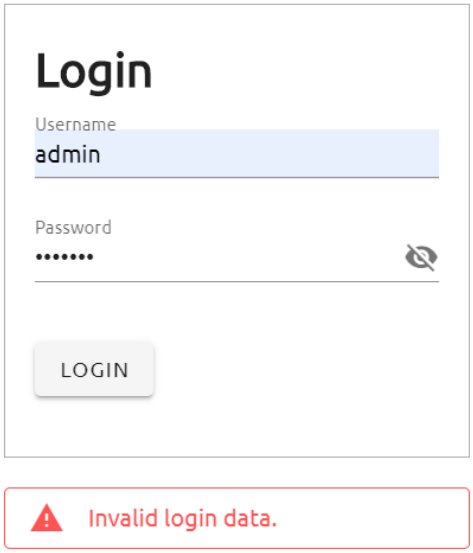
6.1. Testiranje aplikacije

Nad aplikacijom provedeno je funkcionalno testiranje korisničkog sučelja kojim su testirane različite mogućnosti korisničkog sučelja, te su razne iznimke pri korištenju opisane u sljedećim primjerima.

Kod prijave korisnika postoje nekoliko različitih odgovora poslužitelja, prema kojima se diktira tok aplikacije. U slučaju da korisnik unese pravilne podatke za prijavu, poslužitelj odgovara s HTTP status kodom 200, te je prijava uspješna, no ako korisnik unese pogrešne podatke, poslužitelj odgovara s HTTP status kodom 403 koji označava zabranu pristupa resursu, kao što je prikazano u tablici 6.1, te se ispisuje poruka prikazana na slici 6.1.

Tablica 6.1. *Testirani scenariji na stranici za prijavu*

Scenarij	**HTTP zahtjev**	**HTTP odgovor**
Prijava ako su podaci za prijavu pogrešni	POST	403 Forbidden
Prijava ako su podaci za prijavu pravilni	POST	200 Success



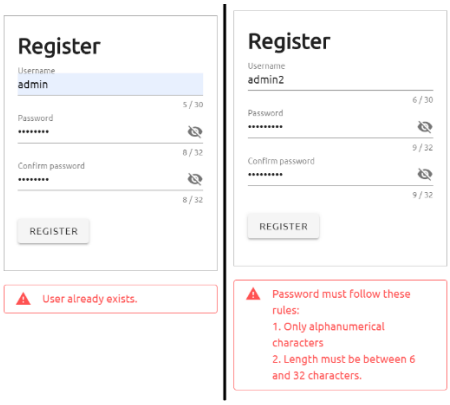
Slika 6.1. *Poruka o grešci pri prijavi*

Pri registraciji su odgovori poslužitelja prikazani u tablici 6.2, a sastoje se od tri različita scenarija: ukoliko korisničko ime ili lozinka ne zadovoljavaju uvjete poslužitelj kao odgovor vraća status kod 400 koji označava pogrešne podatke pri zahtjevu, a ukoliko su uneseni podaci valjani, poslužitelj odgovara sa status kodom 200, te se korisnički račun uspješno stvara.

Tablica 6.2. *Testirani scenariji na stranici za registraciju*

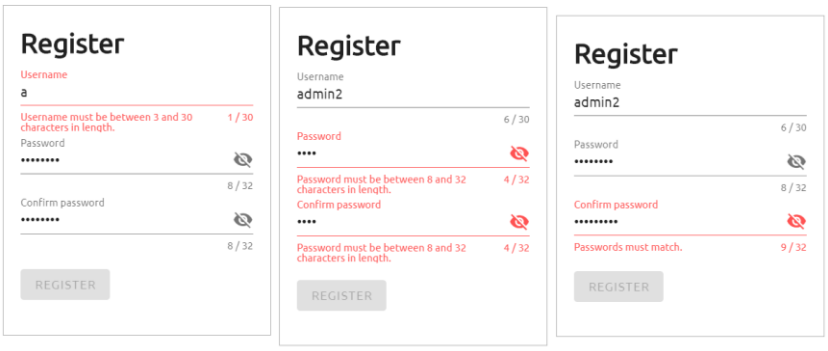
Scenarij	HTTP zahtjev	HTTP odgovor
Registracija ako korisničko ime postoji	POST	400 Bad Request
Registracija ako lozinka ne zadovoljava uvjete	POST	400 Bad Request
Registracija ako su podaci za registraciju pravilni	POST	200 Success

Na slici 6.2 su prikazani slučajevi grešaka kada korisničko ime već postoji i kada lozinka ne zadovoljava uvjete.



Slika 6.2. *Greške na stranici za registraciju pri komunikaciji s poslužiteljem*

Osim provjere podataka na poslužiteljskoj strani, pri registraciji postoje i provjere podataka na klijentskoj strani, prikazane na slici 6.3, a tiču se duljine korisničkog imena, podudaranja lozinke i potvrde lozinke, te duljine lozinke.

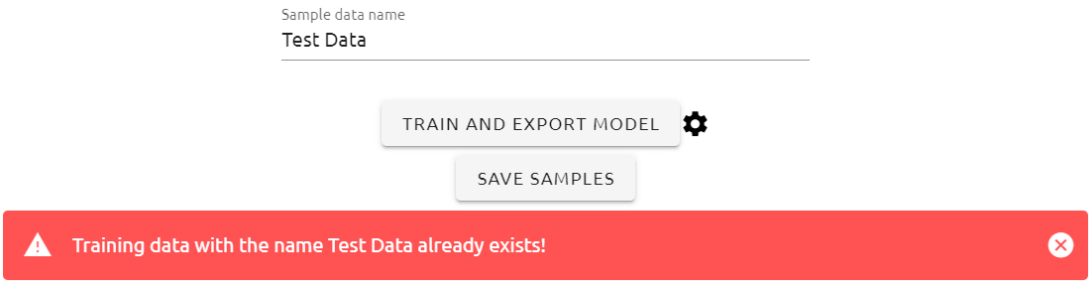


Slika 6.3. *Provjera podataka na klijentskoj strani stranice za registraciju*

Na stranici za treniranje vlastitog modela postoje samo dva moguća odgovora poslužitelja za scenarije navedene u tablici 6.3. U slučaju spremanja skupa podatka s nazivom koji već postoji poslužitelj odgovara s HTTP status kodom 403, te se ispisuje poruka kao na slici 6.4, a u slučaju da naziv skupa podataka ne postoji, podaci se uspješno spremaju.

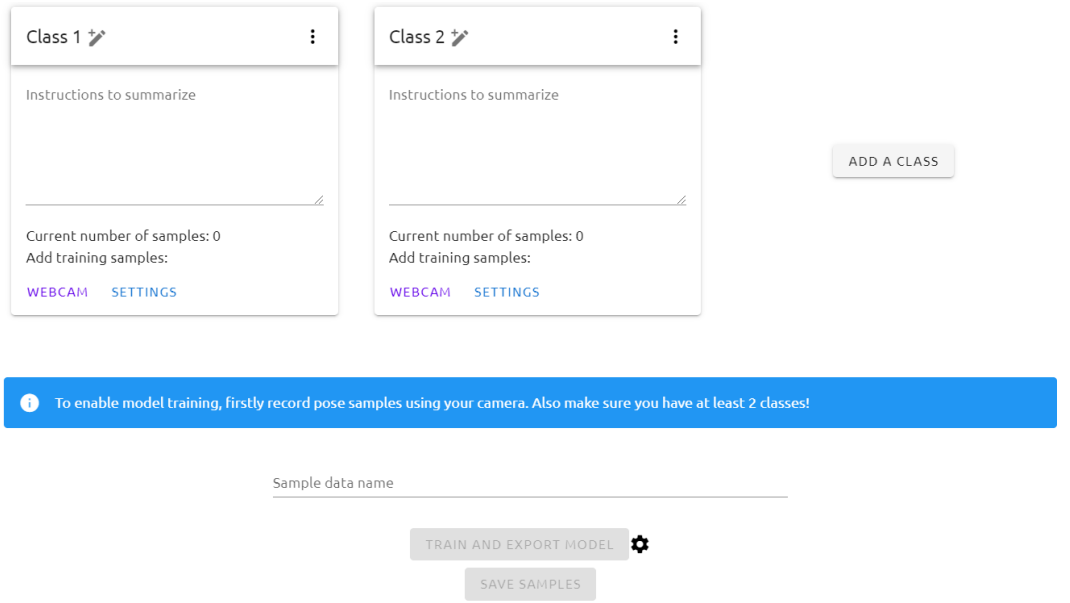
Tablica 6.3. *Testirani scenariji na stranici za treniranje vlastitog modela*

Scenarij	HTTP zahtjev	HTTP odgovor
Spremanje skupa podataka ako naziv skupa podataka postoji u bazi podataka	POST	403 Forbidden
Spremanje skupa podataka ako naziv skupa podataka ne postoji u bazi podataka	POST	200 Success



Slika 6.4. *Greška na stranici za treniranje vlastitog modela pri komunikaciji s poslužiteljem*

Osim mogućih grešaka pri spremanju podatka, postoje razne provjere podataka na klijentskoj strani, a koje uključuju: provjeru da svaka klasa ima naziv, provjeru da svaka klasa ima tekst za sažimanje, provjeru da svaka klasa ima barem jedan uzorak, te provjeru da postoje barem dvije klase. U slučaju da neki od tih uvjeta nije zadovoljen, gumbovi za treniranje i spremanje skupa podataka bit će onemogućeni, kao što je prikazano na slici 6.5.

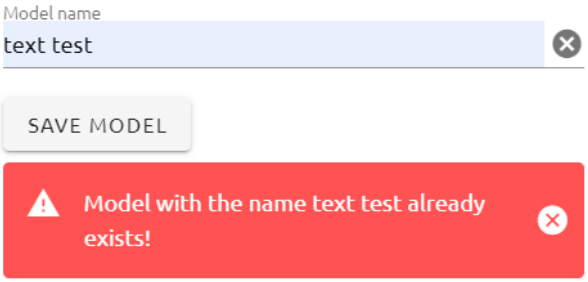


Slika 6.5. *Provjera podataka na klijentskoj strani stranice za treniranje vlastitog modela*

Na stranici za ispitivanje vlastitih modela postoje nekoliko mogućih odgovora poslužitelja za scenarije navedene u tablici 6.4. Pri dohvaćanju svih korisničkih modela poslužitelj odgovara HTTP status kodom 200, osim u slučaju da postoji greška na poslužitelju, kada se odgovara s HTTP status kodom 500. U slučaju spremanja modela s nazivom koji već postoji poslužitelj odgovara s HTTP status kodom 403, te se ispisuje poruka kao na slici 6.6, a u slučaju da naziv modela ne postoji, podaci se uspješno spremaju.

Tablica 6.4. *Testirani scenariji na stranici za ispitivanje vlastitog modela*

Scenarij	**HTTP zahtjev**	**HTTP odgovor**
Dohvaćanje svih korisničkih modela	GET	200 Success 500 Internal Server Error
Spremanje modela ako naziv modela postoji u bazi podataka	POST	403 Forbidden
Spremanje modela ako naziv modela ne postoji u bazi podataka	POST	200 Success



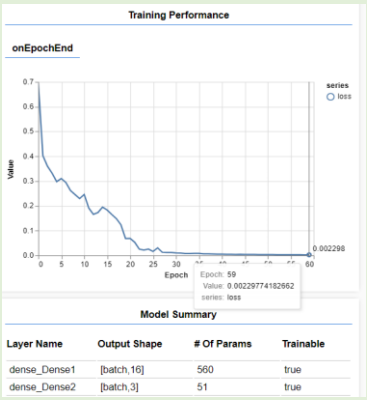
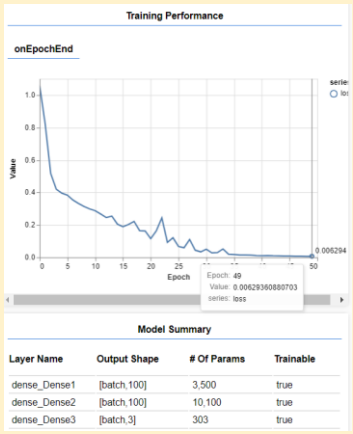
Slika 6.6. *Greška na stranici za ispitivanje vlastitog modela pri komunikaciji s poslužiteljem*

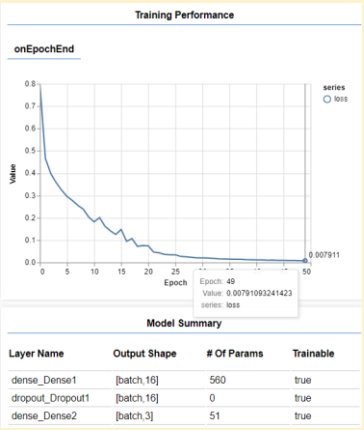
6.2. Testiranje modela strojnog učenja

Za potrebe treniranja modela prikupljeno je 515 različitih uzoraka za slučaj glavobolje, 541 uzoraka za slučaj boli u trbuhu i 432 uzoraka za slučaj boli u prsima. Snimanje položaja provedeno je nad dvije osobe, uz različite pozadine, različitim stupnjevima osvjetljenja i udaljenosti od kamere. Eksperimentirano je s različitim postavkama modela, od dodavanja dodatnih skrivenih slojeva, promjenu broja epoha, veličine serije i stope uzoraka. Najbolji modeli i njihove performanse uspoređeni su u tablici 6.5, gdje je prikazano kako je gubitak najmanji kod sekvencijalnog modela koji je istreniran na 60 epoha, uz stopu učenja od 0.2, sa dva sloja neuronske mreže, gdje prvi skriveni sloj koristi *ReLu* aktivacijsku funkciju, a drugi *Softmax* aktivacijsku funkciju, te uz veličinu serije od 16 uzoraka. Model uspješno radi ukoliko je samo jedna osoba na slici; u slučaju većeg broja osoba model daje netočne rezultate, što je i za očekivati s obzirom da je za potrebe testiranja korišten skup podataka u kojem se samo jedna osoba nalazila

na slici i korišten je PoseNet model koji prepoznaje samo jednu osobu. Ostali opisani modeli su davali približno slične grafove, no u praksi su se pokazali relativno lošijima, gdje bi ili krivo detektirali položaj, ili bi dali vrlo nisku ocjenu uvjerenosti za sve klase, ili neku od klasa nikad ne bi detektirali.

Tablica 6.5. *Usporedbe različitih modela*

GRAF	POSTAVKE MODELA
 **MODEL 1**	Broj epoha: 60 Stopa učenja: 0.2 Slojevi: • *Dense* sloj sa 16 skrivenih jedinica i *ReLu* aktivacijskom funkcijom • *Dense* sloj sa 3 skrivene jedinice i *Softmax* aktivacijskom funkcijom Veličina serije: 16 uzoraka Vrijednost gubitka na kraju treninga: 0.0023
 **MODEL 2**	Broj epoha: 50 Stopa učenja: 0.1 Slojevi: • *Dense* sloj sa 100 skrivenih jedinica i *ReLu* aktivacijskom funkcijom • *Dense* sloj sa 100 skrivenih jedinica i *Sigmoid* aktivacijskom funkcijom • *Dense* sloj sa 3 skrivene jedinice i *Softmax* aktivacijskom funkcijom Veličina serije: 16 uzoraka Vrijednost gubitka na kraju treninga: 0.0063

 **MODEL 3**	Broj epoha: 50 Stopa učenja: 0.1 Slojevi: • *Dense* sloj sa 100 skrivenih jedinica i *ReLu* aktivacijskom funkcijom • *Dropout* sloj • *Dense* sloj sa 3 skrivene jedinice i *Softmax* aktivacijskom funkcijom Veličina serije: 16 uzoraka Vrijednost gubitka na kraju treninga: 0.0079

Za svaki model iz tablice 6.5 prikazana je i usporedba točnosti detekcije nekog od slučaja u tablici 6.6. Iz rezultata je vidljivo kako su modeli 2 i 3 davali dobre rezultate pri detektiranju glavobolje, no imali su značajno manju točnost detektirajući bol u prsima i trbuhu, što vodi do zaključka kako je povećanje broja slojeva ili skrivenih jedinica negativno utjecalo na performanse modela. Svaki model je testiran na način da se simulirao neki medicinski slučaj pred kamerom u trajanju od 30 sekundi. Za vrijeme simulacije za svaki detektirani slučaj računao bi se prosjek uvjerenosti modela u klasu, te točnost modela kao omjer broja točno detektiranih slučajeva i ukupno detektiranih slučajeva na videu.

Tablica 6.6. *Usporedbe rezultata testiranja različitih modela*

SLUČAJ	MODEL	TOČNOST	PROSJEK UVJERENOSTI
Glavobolja	MODEL 1	99%	93%
	MODEL 2	96%	99%
	MODEL 3	91%	81%
Bol u trbuhu	MODEL 1	92%	89%
	MODEL 2	63%	35%
	MODEL 3	36%	48%
Bol u prsima	MODEL 1	84%	72%
	MODEL 2	8%	4%
	MODEL 3	11%	16%

Za sažimanje teksta koristila se usluga i algoritam MeaningCloud. Za potrebe sažimanja teksta usluzi je potrebno proslijediti tekst koji se želi sažeti, kao i ciljani broj rečenica u sažetom tekstu, koji za svaki slučaj iznosi 8. Za slučaj glavobolje, usluzi se prosljeđuje 50 rečenica s ukupno 592 riječi, od kojih su 292 jedinstvene riječi, za slučaj boli u prsima 33 rečenice s ukupno 487 riječi,

od kojih su 227 jedinstvene, a za slučaj boli u trbuhu 27 rečenica s ukupno 378 riječi, od kojih su 204 jedinstvene. Za potrebe ispitivanja rezultata sažimanja teksta, korišten je skup metrika i programski paket ROUGE [41] (*Recall-Oriented Understudy for Gisting Evaluation*), koji se koristi za ispitivanje rezultata sažimanja i prevođenja teksta u kontekstu obrade prirodnog jezika. Skup metrika ROUGE sadržava nekoliko metrika, a one relevantne su:

- ROUGE-1 – odnosi se na preklapanja unigrama (svake riječi) između računalno generiranog sažetog i referentnog sažetog teksta (kojeg u većini slučajeva piše ljudska osoba)
- ROUGE-2 – odnosi se na preklapanja bigrama između računalno generiranog sažetog i referentnog sažetog teksta
- ROUGE-L – odnosi se na najduži zajednički podniz riječi između između računalno generiranog sažetog i referentnog sažetog teksta, te uzima u obzir sličnost u strukturi rečenica

U tablici su prikazani rezultati ROUGE metrika za tekstove različitih slučajeva.

Tablica 6.7. *Rezultati ROUGE metrike sažetaka tekstova*

SLUČAJ	METRIKA	PRECIZNOST	ODZIV	F1 MJERA
Glavobolja	ROUGE-1	0.93	0.24	0.38
	ROUGE-2	0.88	0.22	0.36
	ROUGE-L	0.95	0.28	0.43
Bol u trbuhu	ROUGE-1	0.94	0.34	0.50
	ROUGE-2	0.91	0.33	0.48
	ROUGE-L	0.98	0.40	0.56
Bol u prsima	ROUGE-1	0.95	0.38	0.54
	ROUGE-2	0.92	0.37	0.53
	ROUGE-L	0.98	0.44	0.61

U kontekstu ROUGE metrika, mjera odziva znači koliki postotak referentnog sažetka računalno generirani tekst proizvodi, mjera preciznosti označava koliko je riječi generiranom sažetku relevantno, dok F1 mjera označava ravnotežu između preciznosti i odziva. Iz rezultata testiranja, gledajući sve ROUGE mjere, najbolji rezultat ima tekst koji označava bol u prsima, dok je tekst u slučaju glavobolje onaj s najlošijim rezultatima odziva i F1 mjere.

7. ZAKLJUČAK

U ovom diplomskom radu izrađena je web aplikacija za detekciju hitnih medicinskih slučaja i izdavanja informacija o detektiranom medicinskom slučaju i davanja uputa kako postupiti. Unutar aplikacije implementiran je gotov model strojnog učenja koji koristi neuronske mreže za detektiranje položaja tijela osobe na kameri, koji položaj klasificira u neku od preddefiniranih klasa koje predstavljaju hitne medicinske slučajeve. Ostale funkcionalnosti aplikacije su mogućnost registracije, prijave i odjave korisnika, te mogućnost treniranja i evaluacije vlastitog modela strojnog učenja koji koristi neuronske mreže, a koja je dostupna registriranim korisnicima. U radu je dan osvrt na postojeća tehnološka rješenja za detekciju položaja tijela i sažimanje teksta, te njihovu praktičnu upotrebu u komercijalnim proizvodima. Opisan je koncept strojnog učenja i računalnog vida, kao i relevantne primjene u obliku detektiranja poza tijela i obradbe teksta. Osim toga, objašnjene su mnogobrojne tehnologije koje su korištene za izradu aplikacije, kao što su Vue.js programski okvir za izradu korisničkih sučelja, Node.js izvršno okruženje, MongoDB baza podataka, ml5.js biblioteka za strojno učenje unutar internetskog preglednika, p5.js za crtanje i animaciju, MeaningCloud usluga za analizu teksta i druge. Prikazan je način organiziranja aplikacije, te su objašnjeni i prikazani važni dijelovi aplikacije, kako u obliku programskog koda, tako i u obliku korisničkog sučelja.

Iz rezultata funkcionalnog testiranja aplikacije može se vidjeti da aplikacija radi ispravno, s očekivanim rezultatima pri krivim i pravilnim korisničkim unosima. Rezultati ispitivanja različitih modela za detekciju hitnih medicinskih slučajeva pokazuju da model pravilno detektira medicinske slučajeve, te za njih generira sažetak uputa koje, testirane ROUGE metrikama, imaju visoku preciznost, no relativno nizak odziv, što je aspekt aplikacije u kojem je moguć napredak. Daljnjim treniranjem, korišteni model bi se mogao proširiti tako da detektira veći broj slučajeva, veći broj osoba na slici, kao i da bude točniji u različitim uvjetima, s obzirom da je za treniranje modela korištena tek jedna razlučivost kamere, a za potrebe treniranja modela samo dvije osobe. Prikupljanjem većeg broja uzoraka s osobama različitih građa tijela, s različitim razlučivostima kamere, te različitim udaljenostima od kamere mogao bi se istrenirati još pouzdaniji i točniji model za detekciju položaja tijela.

LITERATURA

[1] T. M. Mitchell, „Machine Learning“, McGraw-Hill Science/Engineering/Math (1997), pp. 1-2 (pristup ostvaren 26.9.2020)

[2] M. Hrga, „Računalni vid“, Zbornik radova Veleučilišta u Šibeniku, No. 1-2/2018, 2018., str. 207 do 216, dostupno na https://hrcak.srce.hr/198597 (pristup ostvaren 08.07.2020.)

[3] E. Alpaydın, „Introduction to Machine Learning“, The MIT Press Cambridge, Massachusetts London, Engleska (2004), pp. 31-32 (pristup ostvaren 26.9.2020)

[4] C. Taylor, „Structured vs Unstructured Data“, Datamation (2018), dostupno na https://www.datamation.com/big-data/structured-vs-unstructured-data.html (pristup ostvaren 29.06.2020.)

[5] Q. Liu, Y. Wu, „Supervised Learning“, Encyclopedia of the Sciences of Learning, Springer, Boston, MA., siječanj 2012, str. od 2 do 3, dostupno na https://www.researchgate.net/publication/229031588_Supervised_Learning (pristup ostvaren 29.06.2020.)

[6] S. Mishra, „Unsupervised Learning and Data Clustering“, Towards Data Science (2017), dostupno na https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a (pristup ostvaren 29.06.2020.)

[7] Anonimno, „Introduction To Machine Learning: Is AutoML Replacing Data Scientists?“, E Cloud Valley (2019), dostupno na https://www.ecloudvalley.com/mlintroduction/ (pristup ostvaren 30.06.2020.)

[8] Anonimno, „Background: Machine Learning“, Github (2020), dostupno na https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Background-Machine-Learning.md (pristup ostvaren 30.06.2020.)

[9] Anonimno, „A Beginner's Guide to Neural Networks and Deep Learning“, SkyMind (2019), dostupno na https://web.archive.org/web/20190724055606/https://skymind.ai/wiki/neural-network (pristup ostvaren (06.07.2020)

[10] A. Jain, G. Kulkarni, V. Shah, “Natural Language Processing,” International Journal of Computer Sciences and Engineering, Vol.6, Issue.1, pp.161-167, siječanj 2018., dostupno na https://www.researchgate.net/publication/325772303_Natural_Language_Processing (pristup ostvaren 05.07.2020.)

[11] Y. van Loon, „ChatBots are coming….but how is NLU used?“, Ywan van Loon (2016), dostupno na https://ywanvanloon.com/chatbots-are-coming-but-how-is-nlu-used/ (pristup ostvaren 05.07.2020.)

[12] S.K. Bharti , K. Sathya Babu, S.K. Jena, „Automatic Keyword Extraction for Text Summarization: A Survey“, ArXiv, Vol. abs/1704.03242, travanj 2017., str. od 4 do 6, dostupno na https://arxiv.org/ftp/arxiv/papers/1704/1704.03242.pdf (pristup ostvaren 08.07.2020.)

[13] J. Brownlee, „A Gentle Introduction to Computer Vision“, Machine Learning Mastery (2019), dostupno na https://machinelearningmastery.com/what-is-computer-vision (pristup ostvaren 08.07.2020.)

[14] A. Garcia-Garcia, S. Orts-Escolano, S.O. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, „A Review on Deep Learning Techniques Applied to Semantic Segmentation“, arXiv, travanj 2017, str 1 do 3, dostupno na https://arxiv.org/pdf/1704.06857.pdf (pristup ostvaren 11.07.2020.)

[15] P. L. Liu, „Single Stage Instance Segmentation — A Review“, Towards Data Science (2020), dostupno na https://towardsdatascience.com/single-stage-instance-segmentation-a-review-1eeb66e0cc49 (pristup ostvaren 11.07.2020.)

[16] G. Papandreou, T. Zhu, L.C. Chen, S. Gidaris, J. Tompson, K. Murphy, „PersonLab: Person Pose Estimation and Instance Segmentation with a Bottom-Up, Part-Based, Geometric Embedding Model“, arXiv, ožujak 2018., str. od 1 do 3, od 14 do 15, dostupno na https://arxiv.org/pdf/1803.08225.pdf (pristup ostvaren 12.07.2020.)

[17] J. Zhang, Y. Zhao, M. Saleh, P. J. Liu, „PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization“, arXiv, prosinac 2019., str. od 1 do 4, dostupno na https://arxiv.org/abs/1912.08777 (pristup ostvaren 24.09.2020.)

[18] „MeaningCloud“, dostupno na https://www.meaningcloud.com/ (pristup ostvaren 05.09.2020.)

[19] A. Toshev, C. Szegedy, “DeepPose: Human Pose Estimation via Deep Neural Networks.” 2014 IEEE Conference on Computer Vision and Pattern Recognition (2014), str. od 1 do 4, dostupno na https://arxiv.org/abs/1312.4659 (pristup ostvaren 24.09.2020.)

[20] Z. Cao, G. Hidalgo, T. Simon, S.Wei, Y. Sheikh, „OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields“, arXiv, prosinac 2018., str. od 1 do 3, dostupno na https://arxiv.org/abs/1812.08008 (pristup ostvaren 24.09.2020.)

[21] „PoseNet“ model, dostupno na https://www.tensorflow.org/lite/models/pose_estimation/overview (pristup ostvaren 24.08.2020.)

[22] „TensorFlow“, dostupno na https://www.tensorflow.org/ (pristup ostvaren 27.08.2020.)

[23] „SwimEye“, dostupno na https://swimeye.com/ (pristup ostvaren 12.07.2020.)

[24] „Coral Detection Systems“, dostupno na https://coraldrowningdetection.com/ (pristup ostvaren 12.07.2020.)

[25] I. Anić, „Vozački umor“, Istraži Me (2013) - http://www.istrazime.com/prometna-psihologija/vozacki-umor/ (pristup ostvaren 08.07.2020.)

[26] A. Hashemi, V. Saba, S. N. Resalat, „Real Time Driver’s Drowsiness Detection by Processing the EEG Signals Stimulated with External Flickering Light“, Basic and clinical neuroscience vol. 5,1 (2014): pp. 22-27., dostupno na https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4202601/ (pristup ostvaren 12.07.2020.)

[27] N. Alioua, A. Amine, M. Rziza, D. Aboutajdine, „Driver's Fatigue and Drowsiness Detection to Reduce Traffic Accidents on Road“, Computer Analysis of Images and Patterns. CAIP 2011. Lecture Notes in Computer Science, vol 6855. Springer, Berlin, Heidelberg, dostupno na https://link.springer.com/chapter/10.1007/978-3-642-23678-5_47 (pristup ostvaren 12.07.2020.)

[28] T.C. Sottek, „Denso's driver drowsiness prototype monitors your facial expressions“, The Verge (2011), dostupno na https://www.theverge.com/2011/12/5/2614164/denso-driver-drowsiness-monitor (pristup ostvaren 12.07.2020.)

[29] B. Pueo, J.M. Jimenez-Olmedo, „Application of motion capture technology for sport performance analysis“, Retos: nuevas tendencias en educación física, deporte y recreación. 2017. pp. 241-247, dostupno na https://www.researchgate.net/publication/314879089_Application_of_motion_capture_technology_for_sport_performance_analysis (pristup ostvaren 13.07.2020.)

[30] M. Gregor, P. Horejsi, Š. Michal, „Case study: Motion capture for ergonomics“, 2015., pp. 468-476, dostupno na https://www.researchgate.net/publication/285221539_Case_study_Motion_capture_for_ergonomics (pristup ostvaren 13.07.2020.)

[31] M. Field, D.A. Stirling, F. Naghdy, Z. Pan, „Motion capture in robotics review“, 2009 IEEE International Conference on Control and Automation (2009): 1697-1702, dostupno na https://www.researchgate.net/publication/224113113_Motion_capture_in_robotics_review (pristup ostvaren 14.07.2020.)

[32] D. Cardinal, „Making gesture recognition work: Lessons from Microsoft Kinect and Leap“, Extreme Tech (2013), dostupno na https://www.extremetech.com/extreme/160162-making-gesture-recognition-work-lessons-from-microsoft-kinect-and-leap (pristup ostvaren 14.07.2020.)

[33] „Mozilla Developer Network“, dostupno na https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide (pristup ostvaren 12.09.2020.)

[34] „Vue“, dostupno na https://vuejs.org/ (pristup ostvaren 12.09.2020.)

[35] „w3schools“, dostupno na https://www.w3schools.com/js/js_htmldom.asp (pristup ostvaren 12.09.2020.)

[36] „p5.js“, dostupno na https://p5js.org/ (pristup ostvaren 08.08.2020.)

[37] „ml5.js“, dostupno na https://ml5js.org/ (pristup ostvaren 08.08.2020.)

[38] „MongoDB“, dostupno na https://www.mongodb.com/ (pristup ostvaren 10.08.2020.)

[39] „Node.js“, dostupno na https://nodejs.org/en/ (pristup ostvaren 10.08.2020.)

[40] „Amazon S3“, dostupno na https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html (pristup ostvaren 10.08.2020.)

[41] Anonimno, „What Is ROUGE And How It Works For Evaluation Of Summarization Tasks?“, RxNLP (2018), dostupno na https://rxnlp.com/how-rouge-works-for-evaluation-of-summarization-tasks/ (pristup ostvaren 27.9.2020)

SAŽETAK

U ovom diplomskom radu modelirana je i izrađena web aplikacija za detekciju hitnih medicinskih stanja na temelju analize slike koristeći PoseNet model za detekciju položaja tijela, a korišteno je i načelo generiranja prirodnog jezika za sažimanje veće količine teksta koji predstavlja opis slučaja i upute za postupanje pri pojavi tog slučaja korištenjem web usluge MeaningCloud. Ostvareno je korisničko sučelje koje omogućuje prijavu korisnika, treniranje vlastitog modela strojnog učenja, spremanje vlastitog modela na računalo ili u bazu podataka, spremanje i uređivanje vlastitog skupa podataka za učenje, te evaluaciju vlastitih modela, kao i preddefiniranog modela koji detektira hitne medicinske slučajeve. Izrađena aplikacija ostvarena je pomoću Vue.js programskog okvira za izradu korisničkog sučelja, ml5.js biblioteke za strojno učenje, MeaningCloud usluge za sažimanje teksta, MongoDB baze podataka, te Node.js izvršnog okruženja kojim se ostvaruje programska podrška na strani poslužitelja.

Ključne riječi: hitni medicinski slučajevi, neuronska mreža, obrada prirodnog jezika, sažimanje teksta, strojno učenje, web aplikacija.

ABSTRACT

WEB SYSTEM FOR MONITORING EMERGENCY MEDICAL CASES BASED ON IMAGE ANALYSIS AND NATURAL LANGUAGE GENERATION

In this paper, a web application for detecting medical emergencies was modeled and developed, using the PoseNet model for pose detection, as well as using the concept of natural language generation to summarize large bodies of text which contain medical case descriptions and instructions on what to do in the case of a described emergency using the MeaningCloud web service. A user interface which enables user registration, training an own machine learning model, saving that model locally on the computer or storing it in the database, saving and editing own training datasets, evaluating the trained models, and evaluating the pretrained model which detects emergency medical cases was created. The application was built using the Vue.js frontend framework, ml5.js machine learning library, MeaningCloud service for text summarization, MongoDB database, and Node.js as the backend runtime environment.

Keywords: medical emergencies, neural network, natural language processing, text summarization, machine learning, web application.

ŽIVOTOPIS

Filip Česnek rođen je 7. listopada 1996. u Osijeku. Nakon završene osnovne škole u Magadenovcu, 2011. godine upisuje se u Elektrotehničku i prometnu školu u Osijeku, gdje stječe zvanje tehničar za mehatroniku. Nakon završetka srednje škole, 2015. godine upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. 2018. godine upisuje diplomski smjer Programsko inženjerstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

PRILOZI

Prilog 1. Dokument završnog rada u .docx formatu

Prilog 2. Dokument završnog rada u .pdf formatu

Prilog 3. Programski kod izrađene web aplikacije