Programsko rješenje web sustava na strani poslužitelja za personalizirano praćenje pacijenata oboljelih od zloćudnih bolesti zasnovano na funkcijskom programiranju

Antunović, Mato

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

Sveučilišni studij

PROGRAMSKO RJEŠENJE WEB SUSTAVA NA POSLUŽITELJA ZA PERSONALIZIRANO PRAĆENJE PACIJENATA OBOLJELIH OD ZLOĆUDNIH BOLESTI ZASNOVANO NA FUNKCIJSKOM PROGRAMIRANJU

Diplomski rad

Mato Antunović

Osijek, 2020.

SADRŽAJ

1. UVOD

Rak pluća jedan je od najčešćih oboljenja u današnjem društvu, a kada se uzme u obzir da je rak drugi najveći uzrok smrti, jasno je da je bitno osvijestiti okolinu, poboljšati prevenciju i poboljšati kvalitetu života pacijentima oboljelima od raka. Smanjenje obolijevanja ne može doći odjednom, nego je potrebna edukacija i prevencija uzroka bolesti koji su danas jako zastupljeni u društvu. Rana dijagnoza i testiranja najvažniji su čimbenici u otkrivanju raka, što znatno povećava vjerojatnost da će pacijent preživjeti i da će se uspjeti spriječiti širenje raka.

Zadatak ovog diplomskog rada je izraditi web sustav na strani poslužitelja za personalizirano praćenje pacijenata oboljelih od zloćudnih bolesti zasnovano na funkcijskom programiranju. Izrađeno je programsko rješenje na strani poslužitelja uz korištenje načela funkcijskog programiranja za praćenje oboljelih pacijenta koje obrađuje unesene podatke na strani klijenta. Liječnik unosi potrebne podatke, a sustav olakšava praćenje stanja pacijenata. Pacijent ima mogućnosti komunikacije s liječnikom, prikaz prilagođenih informacija o raku pluća i praćenje liječničkog tretmana.

U drugom poglavlju uspoređeni su na tržištu dostupni sustavi za praćenje pacijenata oboljelih od zloćudnih bolesti. Treće poglavlje detaljnije objašnjava problematiku raka pluća. U četvrtom poglavlju razrađeni su zahtjevi na strani poslužitelja, model baze podataka, upravljanje korisnicima i ulazni parametri u web sustav. Peto poglavlje prikazuj arhitekturu web sustava i način razmjene podataka između poslužitelja i klijenta. U šestom poglavlju prikazane su izabrane tehnologije, dok je u sedmom poglavlju prikazano programsko rješenje na strani poslužitelja.

1.1 Zadatak diplomskog rada

U diplomskom radu potrebno je s medicinskog gledišta opisati probleme i izazove s kojima se susreću pacijenti i liječnici kod zloćudnih bolesti. Nadalje, treba opisati postojeća web i mobilna rješenja koja omogućuju potporu pacijentu i liječniku, te razraditi model i programsku arhitekturu web sustava na strani poslužitelja i opisati potrebne programske metodologije, tehnologije, okoline i jezike. Za programsko rješenje na strani poslužitelja potrebno je predložiti građu baze podataka, te odgovarajući postupak analize podataka koji će omogućiti obradbu ulaznih podataka, procjenu stanja i rizika oboljelog i pripremu podataka za prikaz. Pri tome treba koristiti načela i programske paradigme i mogućnosti zasnovane na funkcijskom programiranju. Za programsko rješenje treba provesti nužno testiranje i ispitati ga na prikladnom skupu ulaznih podataka.

2. PREGLED STANJA U PODRUČJU I POSTOJEĆIH RJEŠENJE

Od sustava na tržištu koji pružaju pomoć i nadzor nad ljudima oboljelim od zloćudnih bolesti može se pronaći par aplikacija. Većina sustava omogućuje suradnju pacijenata sa stručnim osobljem, no ni jedna od tih aplikacija ne omogućava olakšani rad i liječnicima i pacijentima. Važnost kod sustava ovakve vrste koji nemaju konstantan nadzor od liječnika je mogućnost davanja samo preporuke i savjeta pacijentima, dok za najtočniju informaciju uvijek se najbolje posavjetovati s liječnicima.

2.1 Računalne tehnologije i rak pluća

Primjeri pronađenih aplikacija imaju mogućnost da pacijent unosi podatke kako se osjeća, praćenje iskustava i pristup informacijama koje su filtrirane kako bi korisnici dobili što točnije podatke. Većina pronađenih sustava i aplikacija temelje se na poboljšanju kvalitete života oboljelih pacijenata. Također, veću zastupljenost imaju sustavi s različitim tehnologijama računalnog učenja u želji za ranom detekcijom raka pluća. Problemi kod računalnih učenja su potreba za treniranjem računalnih mreža na ogromnim skupovima podataka i kako bi dobili najbolje rezultate za naš model je najbolje trenirati odvojeni set podataka. Svake godine zbog velike smrtnosti raka pluća i pokušaja rane detekcije organiziraju se različita natjecanja u svrhu poboljšanje ranog pronalaska raka pluća na CT snimkama kod pacijenata. Najpoznatije natjecanje je „Data Science Bowl“ koji je pod organizacijom Kaggle 2017. godine imao nagradni fond od 1 milijun dolara. Odazvalo se 2400 timova, a ponuđeno je više od 18 000 različitih algoritama. Rješenja na ovom natjecanju su ponudila modele koje su davali bolje rješenja čak do 10% od trenutno najboljih rješenja na tržištu. Jedan od pristupa detekcije ranog oblika raka pluća je preko ekspedicijske segmentacije CT slike. [4] Od tada vidimo značajan napredak u modelima te u budućnosti za slučajeve potrebe bi se mogla izvršiti lagana integracija takvog sustava u izrađenu aplikaciju u ovom diplomskom radu. Određene aplikacije omogućavaju kontakt između ljudi koji su oboljeli od sličnih bolesti. Prednosti tih aplikacija u odnosu na pacijenta koji ne koristi nikakav sustav za praćenje trenutnog stanja je pravovremena informacija i rana obavijest o trenutnom stanju ili u slučaju anomalija koje se mogu prepoznavati pomoću različitih specijaliziranih sustava. U slučaju nedoumica pacijent je u mogućnosti javiti se liječniku i postaviti mu pitanje vezano uz dijagnozu ili problematiku trenutnog stanja bez potrebe da samostalno istražuje i bude izložen u većini slučajeva netočnim informacijama[5,6].

2.2 Pregled postojećih rješenja

Ponuđena rješenja na tržištu najviše se usmjeravaju na pokušaje poboljšanja kvalitete života pacijenta, u smislu da im pružaju točne informacije i upute kako se ponašati u pojedinim stanjima.

- **Cancer Theraphy Advisior**

Aplikacija namjenjena onkološkim stručnjacima i ljudima koji žele pratiti zadnje novosti u proučavanim područjima vezanim za medicinu. Primjeri prezentacija koje se mogu pokazati pacijentima vezanim uz njihova stanja, dijete, lijekovi i medicinski kalkulatori.

- **Moovcare**

Najkompletniji sustav koji je pronađen, ali dostupan je samo u Francuskoj. Aplikacije su podijeljene u dva sustava: sustav za praćenje pušača i mogućnost javljanja simptoma, te sustav za praćenje pacijenata kojim je dijagnosticiran rak. Omogućava praćenje stanja pacijenata, simptoma te psihičku podršku pacijentima kako bi im olakšao kvalitetu života u tom teškom razdoblju[5].

- **CancerAid**

Pomaže pratiti trenutno stanje pacijenta, omogućava praćenje simptoma i trenutne terapije. Uz trenutnu terapiju imamo i sustav podrške u kojim pacijenti mogu pisati o svojim iskustvima. Informacije koje se prikazuju pacijentu provjerene su kako se ne bi širile dezinformacije.[6]

- **Cancer.Net Mobile**

Aplikacija je razvijena od strane ASCO (American Society of Clinical Oncology) u kojoj pacijent ima mogućnost praćenja trenutnog stanja, unošenje simptoma, prikaz razine boli u grafičkom sučelju. Praćenje količine lijekova koje se unose, unos pitanja za liječnika.

- **TNM Cancer Staging Calculator**

Kalkulator koji omogućuje računanje TNM-a za različite vrste raka. Rezultati i tablice potrebne za računanje se nalaze u AJCC Cancer Staging Manual koji se obnavlja svake godine.

2.3. Pregled stanja u odnosu na razvijeni web sustav

Zbog velike težine i jako brzog razvoja bolesti, pravovremena reakcija je jako bitna. Pacijenti oboljeli i dijagnosticirani od raka pluća imaju mogućnost od doživljavanja 10 godina od dijagnosticiranja u samo 6.8% slučajeva, a prvu godinu doživljavaju u samo 41.8% slučajeva. Prethodno navedeni podatci su ispitani na 2347 pacijenta.[13]. Zbog težine prethodno navedenih podataka aplikacije i sustavi za pomoć oboljelim pacijentima su jako važni. Zbog toga naša web

sustav pokušava spojiti liječnika i pacijenta kako bi ostvarili što bolju komunikaciju i omogućila im bolje standarde života i za same pacijente, a i njihove obitelji zbog točnosti informacija. Pacijent ima mogućnost pratiti sve dijagnoze u aplikaciji, komunicirati s liječnikom, pratiti razvoj razina raka pluća, provjeru dogovorenih termina.

3. RAK PLUĆA

Tumor se može podijeliti na maligni i benigni. Benigni tumori su „dobri“ tumori koji se mogu ukloniti i njihova veličina ostaje ista te se ne šire na druge dijelove tijela. Maligni tumori su agresivci u našem organizmu. Šire se i napadaju druga tkiva u tijelu, ulaze u limfni sustav i krvotok, a onda stvaraju nove tumore. Širenje i razvoj tumora se naziva metastaziranje. Rak pluća je jedan od najteže izlječivih oblika raka.

Osim sveopće poznatih uzroka cigara i duhana, na razvoj utječu i različiti genski čimbenici, izlaganje plinu radonu, radnici ili ljudi koji provode puno vremena pored azbestnih ploča. Sve češća pojava raka pluća je i kod ljudi koji žive u područjima s povećanom zagađenošću zraka ili kod tako zvanih pasivnih pušača to jest ljudi koji su izloženi dimu duhanskih proizvoda, iako ih sami ne konzumiraju. Simptomi raka pluća ovise o tipu, smještaju i načinu na koji se širi. Najčešći simptom je neprestani kašalj, kod ljudi koji imaju kronični bronhitis kašalj postaje jači. Kod iskašljavanja javlja se krv, dok u slučaju kada je rak smješten u nižim krvnim žilama može doći do težih oblika krvarenja[2,3].

Cilj izrade diplomskog rada i web sustava je pomoć doktorima koji prate i nadziru pacijente, te pomoć samim oboljelim od zloćudne bolesti. Pacijent imaju mogućnost unijeti svoje trenutno stanje i na osnovu odgovora na kratke upitnike dati važne informacije koje će sustav dalje obraditi. Rezultati omogućuju doktoru uvid u trenutno stanje pacijenta. Sustav onda prepoznaje da li je trenutno stanje pacijenta opasno ili ima li podudarnost s rizičnim situacijama kako bi moglo pozvati pomoć. Sustav se razvija na strani poslužitelja i strani korisnika, podatci se spremaju u bazi podataka. Kako bi osigurali što uniformnost koda i sustava slijeđena su načela funkcijskog programiranja. Funkcijsko programiranje zahtijeva kod pisan u funkcijama, rezultati za iste uvjete su uvijek jednaki i nema nepotrebnog ponavljanja koda.

3.1 Tipovi raka pluća

Rak pluća se dijeli u dvije skupine: rak pluća malih stanica (SCLC) i rak pluća ne-malih stanica(NSCLC). Do ove klasifikacije je došlo promatranjem mikroskopskog izgleda stanica. Važno je razlikovati ove dvije vrste rakova zbog načina širenja i opasnosti za organizam.

SCLC je najagresivniji i najopasniji rak pluća. Širi se brže od svih drugih karcinoma pluća. U 99% posto slučajeva pronalazimo ga kod pušača, dok samo 1% ljudi obole od ovog tumora na neki drugi način. Kod ljudi kod kojih je dijagnosticiran SCLC tumor obično se dogodi prekasno i već se tada proširio na druge organe i tkiva[1,2].

NSCLC spada u najčešći tip raka pluća u 80% slučajeva.[1] . Može se podijeliti u 3 glavne skupine, a te skupine ubrajamo: karcinom malih stanica (mikrocelularni karcinom), karcinom pločastih stanica (planocelularni karcinom), adenokarcinom i karcinom velikih stanica[2].

U najvećem broju slučajeva rak pluća počinje u bronhima. Bronhi su dišni putevi koji omogućavaju opskrbu zraka u pluća. Takav oblik raka nazivamo bronhogeni karcinom ili karcinom bronha[2].

Bronhalni karcinoidi su obično mali(3-4cm) kada se prvi put uoče i pojavljuju se najčešće kod ljudi mlađih od 40 godina. Iako mogu metastazirati, ali u jako malom broju slučajeva. Karcinoidi se šire puno sporije od raka bronha, a iz tog razloga se najčešće otkriju dovoljno rano da se kirurški može ukloniti u cijelosti[1].

Iz zračnih mjehurića pluća može nastati karcinom alveolarnih stanica, koji iako se može pojaviti samo na jednom mjestu, često se u isto vrijeme razvija na više mjesta u plućima.

U rjeđe tumore pluća spadaju adenom bronha, sarkom i hrskavični hamartom. Limfom je rak limfnog sustava, koji može početi u plućima ili se u njih proširiti.

Metastazirani maligni tumori iz drugih dijelova tijela imaju tendenciju širenja na pluća. Prilikom metastaziranja su građene od iste vrste stanica kao i početni tumor, tako da prošireni rak ne spada u rak pluća, nego u metastazirani rak dijela tijela koji se proširio u pluća.

3.2 Simptomi raka pluća

Simptomi raka pluća mogu varirati ovisno o tome gdje je tumor smješten i iz kojeg dijela tijela se proširio. Problem se javlja u 25% slučajeva kada pacijenti nemaju nikakve simptome raka pluća nego se on slučajno otkrije prilikom rutinskog RTG snimka pluća ili CT[1].

Najčešći simptomi nastaju zbog rasta raka pluća i invazija plućnog i okolnog tkiva što može otežati disanje, dovodi do kašlja, pojavljuje se bol u prsima i iskašljavanje krvi. Zbog invazije raka pluća može doći do sužavanja bronha što za posljedicu dovodi do splasnuća dijela pluća koji taj bronh opskrbljuje. Posljedica također može biti i upala pluća, bolovi u prsima i neugodan osjećaj otežanog disanja te povišena temperatura.

Kasnije se mogu pojaviti i novi simptomi poput slabosti, gubitka apetita i težine. Zbog nakupljanja tekućine oko pluća može doći do zaduhe, a s povećanim širenjem može doći i do teške zaduhe, zatajenja srca i niske razine kisika.

„Ako su veliki dišni putovi opstruirani, može doći do kolapsa dijela pluća i posljedične infekcije (apscesi, upala pluća) u opstruiranom području.“ U slučajevima kada rak zahvati živac može doći do boli u ramenima koja se širi niz ruku, do promuklosti ili paralize glasnica. U slučaju da je rak zahvatio jednjak može doći do poteškoća u gutanju(disfagija) [1].

3.3 Stadiji raka pluća

Stadij je oznaka proširenosti raka u tijelu. Stadijevanje je procjena veličine i invazije raka u okolno tkivo. Također se provjerava prisutnost ili odsutnost metastaza u organima ili limfnim čvorovima. Pacijent koji ima viši stadij ima manje šanse za preživjeti i raširenost raka u njegovom organizmu je veća. Stadij odlučuje o vrsti tretmana koje će pacijent primiti.

Kako bi se odredio točan stadij bolesnika koriste se različite metode i tehnike među kojima su najpopularnije laboratorijski nalazi krvi, CT, RTG, MR, PET-CT i scintigrafija kostiju. Krvni testovi s alarmantnim vrijednostima nam mogu pokazati gdje se nalaze metastaze, ali to se dokumentira pomoću radioloških metoda.

Rak pluća kod SCLC tipa dijeli se u 2 stadija. U prvom stadiju rak je ograničen na jedno područje unutar pluća, dok u drugom stadiju rak pluća se raširio po tijelu i zahvatio druge organe, tkiva ili limfne čvorove.

NSCLC se dijeli u 4 stadija gdje u prvom stadiju imamo istu situaciju da je tumor ograničen samo na pluća. Drugi i treći stadij nam govore da se tumor nalazi u prsnom košu. Ovisno o količini invazivnih tumora i mogućnosti da bude pojave zahvaćenosti limfnih čvorova dodjeljuje mu se pripadajući stadij. Postoji mogućnost pojave samo zahvaćenosti limfnih čvorova ili invazivnih tumora, ne mora se pojaviti zajedno ili sve odjednom. Četvrti stadij je najopasniji jer se rak širi i na ostale organe[1,9].

3.4 Dijagnoza raka pluća

Liječnik počinje sumnjati i vrši ispitivanje raka pluća, ako pacijent ima konstantan ili jači kašalj ili neke druge simptome. Kod pacijenata koji su pušači vjerojatnost da ih liječnik pošalje na testiranje je puno veća. Inicijalna pretraga je RTG srca i pluća kojom se mogu vidjeti promjene u plućnom tkivu. Ponekad samo zasjenjenje na rendgenogramu prsnog koša može upozoriti na potrebu za daljnjim testiranjem i kod ljudi koji nemaju simptome. Rendgenogram može otkriti većinu plućnih tumora, no ima problema u prepoznavanju malih tumora.

U slučaju da pacijent ima klasične simptome, a uredan nalaz s RTG-a potrebno je izvršiti i daljnja testiranja što najčešće uključuje kompjutoriziranu tomografiju (CT). Osim CT-a može se još koristiti i magnetske rezonancije (MRI) ili pozitronske emisijske tomografije (PET). Ako postoji sumnja na rak pluća ponekad je dovoljno i samo iskašljane, ako možemo osigurati dovoljno materijala za pretragu, a naziva se još i citologija iskašljaja. U slučajevima kada to nije moguće, a ima sumnje i potrebna je mikroskopska pretraga uzoraka tkiva potrebno je izvršiti bronhoskopiju.

Bronhoskopija je tehnika kojom se tankim uređajem ulazi kroz usta ili nos u dišne puteve te se uzimaju uzorci pluća. Liječnik ovisno o potrebi može uzeti samo stanice za stijenki bronhi ili izrezati dijelove tkiva koje će promatrati mikroskopom kako bi se utvrdilo da li postoje stanice raka.

Za mjesta koja se teško mogu doseći bronhoskopom liječnik može pokušati doći pomoću igle. Napravi se rez na koži i igla se provlači između rebara. Ovaj se postupak mora vršiti pod kontrolom CT-a, a naziva se punkcijsko-aspiracijska biopsija. CT snimak glave ili trbuha nam može pomoći i ukazati, ako se rak pluća proširio na druge dijelove tijela poput nadbubrežne žlijezde ili mozga. Snimci kostiju mogu pokazati, ako se proširio na kosti.

Karcinomi malih stanica često imaju sklonost širenja u koštanu srž te iz tih razloga liječnik može napraviti biopsiju koštane srži.

Kod pacijenata kod kojih je dokazan rak pluća potrebno je procijeniti stupanj proširenosti bolesti jer će o tome ovisi odluka o liječenju. U današnje vrijeme najčešće je u upotrebi kompjuterizirana tomografija te PET-CT[2].

3.5 Liječenje raka pluća

Liječenje raka pluća ovisi o lokalizaciji i opsegu, histološkom tipu tumora te općenitom stanju bolesnika. Najčešći načini liječenja raka pluća su: zračenje, kemoterapija, kirurški zahvati i ciljano liječenje[2,3].

Terapije možemo podijeliti u dvije kategorije kurativnu koja ima za cilj iskorjenjivanje tumorske bolesti i palijativnu koja ne može izliječiti rak, ali pokušava bolesniku smanjiti bol i patnju. Obično se kombinira više vrsta terapija. Odabire se primarna terapija, a uz nju se obično dodaje još neka koja se naziva adjuvantna terapija. Najčešći primjer adjuvantne terapije je korištenje kemoterapije koje se provode nakon kirurških operacija kako bi ubili preostale stanice koje su ostale nakon operacije.

Kirurški zahvati obično se izvode kod tumora koji je u početnim stadijima (1 ili 2) NSCLC i u slučajevima kada se tumor nije raširio izvan pluća. U stvarnosti to predstavlja između 10% do 35% slučajeva kod ljudi koji izvrše kirurške zahvate 25 do 40% preživi barem 5 godina nakon dijagnoze. U slučajevima kada se tumor može ukloniti kirurški postoji mogućnost ponovno javljanja i povratka raka u 6 do 12% slučajeva, jer operacija ne mora rezultirati potpunim uklanjanjem. Postotak se drastično povećava za ljude koji nakon operacije nastave pušiti[1,9]. Kirurški zahvati se jako rijetko izvode na SCLC, jer je već pri otkrivanju raka on raširiti po tijelu nije lokaliziran.

Prije izvršavanja kirurških zahvata moraju se provesti testiranja kako bi se odredilo da li je pacijent u mogućnosti izdržati operaciju i kako je prestalo plućno krilo sposobno odraditi svoje funkcije. U procesu operacije određuje se koliko pluća treba ukloniti, a to može biti od jednog djelića pa do čitavog plućnog krila. Bolesti srca mogu biti jako veliki faktor u odlučivanju hoće li pacijent moći na resekciju. Operacije su jako zahtjevne, dugotrajne i komplicirane, a oporavak se vrši u bolnici uz konstantan nadzor zbog velikih mogućnosti komplikacija tijekom i nakon operacije. Najčešće komplikacije su krvarenja, infekcije i problemi s anestezijom zbog stanja srca ili pluća. Pacijenti mogu iskusiti mnoge nuspojave poput otežanog disanja, bol i slabosti nakon operacije, a sam oporavak i boravak u bolnici može trajati od nekoliko dana, tjedana pa mjeseci.

Moguće je i uklanjanje raka nastalog u drugom području koji se proširio u pluća, tako da se prvobitno ukloni rak s originalnog mjesta nastanka zatim iz pluća. Ovaj postupak se jako rijetko izvodi zbog zahtjevnosti zahvata na bolesnikov organizam. 90% slučajeva ne preživi u idućih 5 godina.

Liječenje pomoću radioterapije primjenjuje se na ljude koji nisu sposobni imati operaciju zbog svog trenutnog stanja, nekih drugih bolesti, rak se proširio izvan pluća ili rak preblizu dušnici. Može se primijeniti za liječenje i NSCLC i SCLC. Koriste se visoko-energetske zrake koje se usmjeravaju na područje zahvaćeno tumorom. Može se koristi kao kurativna, palijativna (koriste se niže razine zračenja nego kod kurativne terapije) ili kao adjuvantna terapija u kombinaciji sa zahvatom ili kemoterapijom.

Kemoterapija se koristi za liječenje NSCLC i SCLC. Podrazumijeva se uzimanje lijekova koji sprječavaju rast i razmnožavanje stanica tumora. Koristi se kao nadopuna kirurškim zahvatima ili u kombinacijama s radioterapijama. Najveća razlika između zahvata, zračenja i kemoterapije je što kemoterapija ne razliku zdrave od stanica raka nego utječe na sve stanice u organizmu. Najviše

stradavaju brzo dijelove stanice. Zbog stradavanja stanica koštane srži, probavnog sustava i vlasišta nuspojave su najčešće mučnina i povraćanje, umor, proljev, opadanje kose i mnogi drugi. U današnje vrijeme postoji velika količina kemoterapijskih lijekova koji se koriste za liječenje uznapredovanog raka, Najučinkovitiji su se pokazali lijekovi s dva spoja, od kojih je jedan najčešće sol platine. Postoji mogućnost uzimanja kemoterapije u obliku tableta, ali najčešće se daje u obliku injekcija ili infuzije. Bolesnik će primati terapiju ovisno o reakciji raka i stanju bolesnika. U slučaju da nema promjene nakon 4 ciklusa, provjerava se trenutno stanje pacijenta te mu se po tome ponovno određuje terapija, ali koristi se drugi citostatik[1,8,9].

Biološke terapije liječenja temelje se na liječenju primjenom antitijela. Za razliku od kemoterapije uništava samo negativne stanice. Naziva se još i ciljano liječenje, jer nema nuspojava kao kod kemoterapije. Antitijela prepoznaju molekule koje se nalaze na površini raka, vezuju se na nju te organizam prepoznaje te stanice kao nametnike i uništava.

Proces stvaranja novih krvnih žila naziva se angiogeneza. Kod zloćudnih tumora kada prerastu 1-2 milimetra, kako bi se nastavili širiti i dalje rasti moraju si osigurati dovoljno hranjivih sastojaka. Kada ponestane hrane i kisika iz okolnog tkiva počinju stvarati nove krvne žile koje dovode hranjive sastojke do tumora, a isto tako pomaže tumoru u metastaziranju. Proučavanje ovog procesa pomoglo je znanstvenicima i istraživačima u stvaranju monoklonskih antitijela koji blokiraju angiogene signale raka. Sprječavanjem angigenih signala sprječava se rast tumora i opskrba hranjivim tvarima. Kombiniranje s kemoterapijom usporava napredovanje raka i pospješuje liječenje[1,10].

EGFR je kratica za receptor epidermalnog faktora rasta(eng. Epidermal growth factor receptor), a možemo ga podijeliti na divlji i mutirani tip receptora. EGFR se nalazi na površini mnogih tumorskih stanica, a njegova zadaća je metastazirane tumorskih stanica. Proučavanje EGFR omogućilo je stvaranje lijekova i terapija koje sprječavaju rast i širenje tumorskih stanica. Najčešći lijekovi koji blokiraju EGFR su monoklonska antitijela i male molekule. Male molekule ulaze unutar stanica tumora i blokiraju metastaziranje stanica iznutra, dok monoklonska antitijela se vežu na dio EGFR koji se nalazi na površini. Prednost malih molekula je mogućnost liječenja pacijenta od kuće, jer dolaze u obliku tableta za razliku od monoklonskih antitijela koji se uzimaju u obliku infuzije. Testiranje za određivanje tipova EGFR omogućava primjenu terapija za bolesnike na koje će imati najbolji učinak. Testiranje receptora za epidermalni faktor rasta postaje

sve popularnije i dostupno je u većini zemalja. Kod pacijenata kod kojih će to biti moguće EGFR terapije će činiti primarnu metodu liječenja[1,10].

Potporno liječenje trudi se olakšati ili spriječiti simptome nastale primjenom terapija. Potporne terapije ne sprječavaju rast tumora, ali pomažu u ublažavanju simptoma bolesti. Najveću ulogu u liječenju tumora i zloćudnih bolesti imaju kemoterapije, no uz kemoterapije je vezano i najviše nuspojava. Zbog djelovanja i na dobre i na loše stanice, javljaju se mnoge nuspojave koje se pokušavaju bolesniku olakšati pomoću potpornog liječenja[12].

Stanice raka svakog tipa rastu i šire se na različite načine, te se stoga i različito liječe.

4. MODEL APLIKACIJE

U ovom poglavlju bavimo se razradom i problematikom idejnog rješenja za sustav na strani poslužitelja koje će pratiti prethodno spomenute funkcionalnosti i standarde. Sustav se projektirati i izrađuje uz praćenje zakona o prenosivosti i odgovornosti zdravstvenog osiguranja uz personalizirano praćenje pacijenata oboljelih od zloćudnih bolesti te koristeći se načelima funkcijskog programiranja.

4.1 Razrada zahtjeva na strani poslužitelja

Prije bilo kakve izrade programskog rješenja potrebno je definirati mogućnosti, korisnička sučelja i razine pristupa pojedinih korisnika. Cilj diplomskog rada je izrada sučelja koje će pomoći liječniku u praćenju pacijenata, a pacijentima olakšati i poboljšati svakodnevan život. Kako bi im omogućili različite funkcionalnosti moramo odvojiti korisnike i definirati zahtjeve za svaku od tih kategorija.

1. Neprijavljeni liječnik ima mogućnosti:
    - kreiranja korisničkog računa (registracija)
    - korisnik se može prijaviti kao doktor u aplikaciji, ako je već registriran i unese točne podatke za elektroničku poštu i šifru
2. Prijavljenim liječnicima na strani poslužitelja omogućeno je:
    - registracija pacijenata uz generiranje sigurne šifre koja dolazi na elektroničku adresu pacijenta
    - doktor se mora moći odjaviti iz sustava
    - doktor ima mogućnost uređivanja svojih podataka koji su uneseni u web sučelje
    - mogućnost uređivanja i dodavanja podataka za pacijente:
        - podatci o lijekovima, količini i vrsti upotrebe
        - Termini i sastanci
        - Podaci o pacijentu
        - Promjena stanja bolesti
    - Doktoru mora biti omogućen pregled svih pacijenata
    - Pregled informacija o pojedinim pacijentima
    - Dodavanja dijagnoza i dokumentacija o pacijentu u sustav

- Mogućnost komunikacije i slanja obavijesti pacijentu

3. Neprijavljeni pacijent ima mogućnosti:
    - prijave u sustav, u slučaju kada ga liječnik prijavi u sustav te pacijent dobije šifru na adresu svoje elektroničke pošte
4. Prijavljenim pacijentima na strani poslužitelja omogućeno je:
    - Pacijent se mora moći odjaviti iz sustava
    - Pacijentu je omogućeno uređivanje svojih podataka koji su uneseni u web sučelje, ali nije u mogućnosti mijenjati dijagnoze doktora
    - Pregled informacije o sebi:
        - Podatci o lijekovima, količini i vrsti upotrebe
        - Zakazani termini kod doktora (prethodni i budući)
        - Općeniti podatci o pacijentu
        - Praćenje bolesti
    - Kontaktiranje doktora koji je zadužen za njegovo liječenje
        - Praćenje obavijesti i informacija o bolesti
        - Pregled i dodavanje bilješki

4.2 Upravljanje korisnicima

Aplikaciju mogu koristiti i liječnici i pacijenti. Liječnici i pacijenti ne mogu imati uvid u iste podatke te iz tog razloga su odvojeni i koriste različita korisnička sučelja. Doktori imaju mogućnosti kreiranja novih pacijenata, upravljanja terapijama, kreiranjem termina i još dosta mogućnosti koje se navode kasnije, a detaljnije će biti pokazane mogućnosti na klijentskoj strani. Pacijenti imaju mogućnosti slanja poruka i obavijesti doktoru, uređivanja svog profila, dodavanja potrebnih dokumenata i postavljanje sadržaja na stranicu. Za pristup stranicama na kojim se nalaze bilo kakve informacije zahtjeva se prijava korisnika u sustav.

4.3 Internacionalni sustav za određivanje stadija karcinoma pluća

Na poslužiteljskoj strani prikazuje se način kako se određuje stadij karcinoma raka pluća. Opis vrijednosti za parametre koje ćemo koristiti T,N i M je preuzet iz priloga [9], dok dodani su parametri koji se ne nalaze u prilogu [9], a koriste se za računanje u najnovijoj verziji. Najnovija verzija se može pronaći u prilogu broj[13], gdje su detaljno raspisani svi slučajevi i načini određivanja parametara. Svake godina se izdaje nova verzija priloga [13].

Tablica 4.1 Prikaz kategoriziranja primarnog tumora

Primarni tumor (T)

Tis	Karcinom u situ
T1	Tumor ≤ 3 cm bez prodora proksimalnije od lobarnog bronha (tj. nije u glavnom bronhu)
T2	Tumor sa bilo kojom od navedenih karakteristika: • >3 cm • Zahvaća glavni bronh ≥2 cm distalno od karine • Zahvaća visceralnu pleuru • Atelektaza ili opstruktivni pneumonitis segmenta ili lobusa ali ne zahvaća čitavo pluće
T3	Tumor bilo koje veličine sa jednom od navedenih karakteristika: • Zahvaća stijenku toraksa (uključujući i tumore gornjeg sulkusa), ošit, medijastinalnu pleuru ili parijetalni perikard • Zahvaća glavni bronh <2 cm distalno od karine ali ne zahvaća karinu • Atelektaza ili opstruktivni pneumonitis čitavog pluća
T4	Tumor bilo koje veličine sa jednom od navedenih karakteristika: • Zahvaća medijastinum, srce, velike krvne žile, traheju, jednjak, trup kralješka, karinu • Maligni pleuralni ili perikardni izljev Satelitski tumorski čvor (čvorovi) u istom režnju gdje se nalazi i primarni tumor

Tablica 4.2 Prikaz kategoriziranja regionalnih limfnih čvorova

Regionalni limfni čvorovi(N)	
NX	Nedostatak informacija
N0	Nema metastaza u regionalnim limfnim čvorovima
N1	Metastaze u istostranim peribronhalnim i/ili istostranim hilusnim limfnim čvorovima, te zahvaćeni intrapulmonalni čvorovi izravnim širenjem iz primarnog tumora
N2	Metastaze u istostrane medijastinalne i/ili subkarinalne limfne čvorove
N3	Metastaze u medijastinalne i/ili hilusne limfne čvorove suprotne strane, istostrane ili kontralateralne skalenske ili supraklavikularne limfne čvorove

Tablica 4.3 Prikaz kategoriziranja udaljenih metastaza

Udaljene metastaze(M)	
M0	Nema udaljenih metastaza
M1	Udaljene metastaze postoje [uključujući metastatske tumorske čvorove u lobusu ili

	lobusima na istoj strani ali na suprotnoj od primarnog tumora]

Tablica 4.4 TNM numeracija stadija

Numerirani stadij	
Okultni karcinom	Nedostatak informacija
0 stadij	Prisutnost raka u plućnoj tekućini ili sluzni, ali odsutnost primarnog raka i odsutnost raka u ostalim područjima tijela.
Stadij 1-3	Prisutnost tumora u tijelu, ali nije metastazirao u druga područja.
Stadij 4	Uznapredovali stadij raka, metastazirao od primarnog tumora

4.4 Ulazni parametri u web sustav

Ulazni parametri ovise o ruti na kojoj se korisni nalazi, ali na strani poslužitelja se nalazi upravitelj prijavama koji za svakog prijavljenog korisnika u sustavu prati njegove podatke, kako bi u slučaju upita mogli jednostavno odgovoriti, ako korisnik ima dozvoljenu razinu pristupa.

Dodatna sigurna razina kod ulaznih parametara je registracija pacijenta se odvija preko doktora. Doktor popuni podatke za pacijenta, u kojim se traži ime, prezime i povijest bolesti. Unesena email adresa se provjerava da li je dobrog formata i da li već postoji u bazi podataka. Nije moguće da postoji dva puta ista adresa e-pošte u bazi podataka neovisno da li se odnosi na doktore ili pacijente. Osmišljen je siguran način unosa šifre u bazu podataka, a isto vrijedi i za doktora koji dodaje pacijenta te osigurati pacijentu siguran način da dobije šifru koju doktor ne zna.

Jedan od zadovoljavajućih načina i rješenja koje zadovoljava zakon o prenosivosti i odgovornosti zdravstvenog osiguranja je upisivanje lozinke u bazu podataka koja je hashirana (eng. Hash), a pacijentu šaljemo tu lozinku na mail. Lozinku smo hashirana pomoću bcrypt koji je namijenjen za radu s lozinkama. Iako je algoritam kreiran 1999, godine jedan je od najboljih algoritama za hashirane temeljen na Blowfishu koji podnosi napade rječnicima, napade sirove snage (eng. Brute force) i napade u kojima se odvija pogađanje lozinki (eng. Rainbow table attacks) koji danas pomoću grafičkih kartica mogu odraditi jako veliki broj kombinacija. Trenutni standard je PBKDF2, iako uz njega se još koriste i Scrypt,Argon2 i bcrypt. Scrypt i Argon2 su jako memorijski

zahtjevni za korištenje, ali kako bi to mogli iskoristiti u našu prednost moramo imati jako veliku memoriju servera, ili prebacivati dio napada i posla na klijentska računala. U slučaju nedostatka memorije jako su osjetljivi na distribuirane napade uskraćivanja resursa (eng. Distribution denial of service).

Korisnici pri pokušaju ulaska u sustav unosi adresu svoje elektroničke pošte i lozinku. Lozinka se provjera tako da se usporedi lozinka u bazi i unesena lozinka pomoću Bcrypta. Ako su unesene lozinke poklapaju korisnik može pristupiti sustavu.

Ulazni parametri koje korisni može unijeti pri promjeni profila provjeravaju se prije unosa u bazu podataka. Omogućena je promjena svih početnih postavki, a priložene datoteke postavljamo u Cloud preko osigurane komunikacije. Korisnik može spremiti svoju profilnu sliku koja će biti vidljiva, ili doktor može postaviti dokumente vezane uz bolest koji su vidljivi samo doktoru i tom pacijentu.

Pacijentu su ograničeni unosi podataka i izmjene bilo kakvih osjetljivih sadržaja koji su vezani uz liječenje, dijagnozu ili bolest.

4.5 Model baze podataka

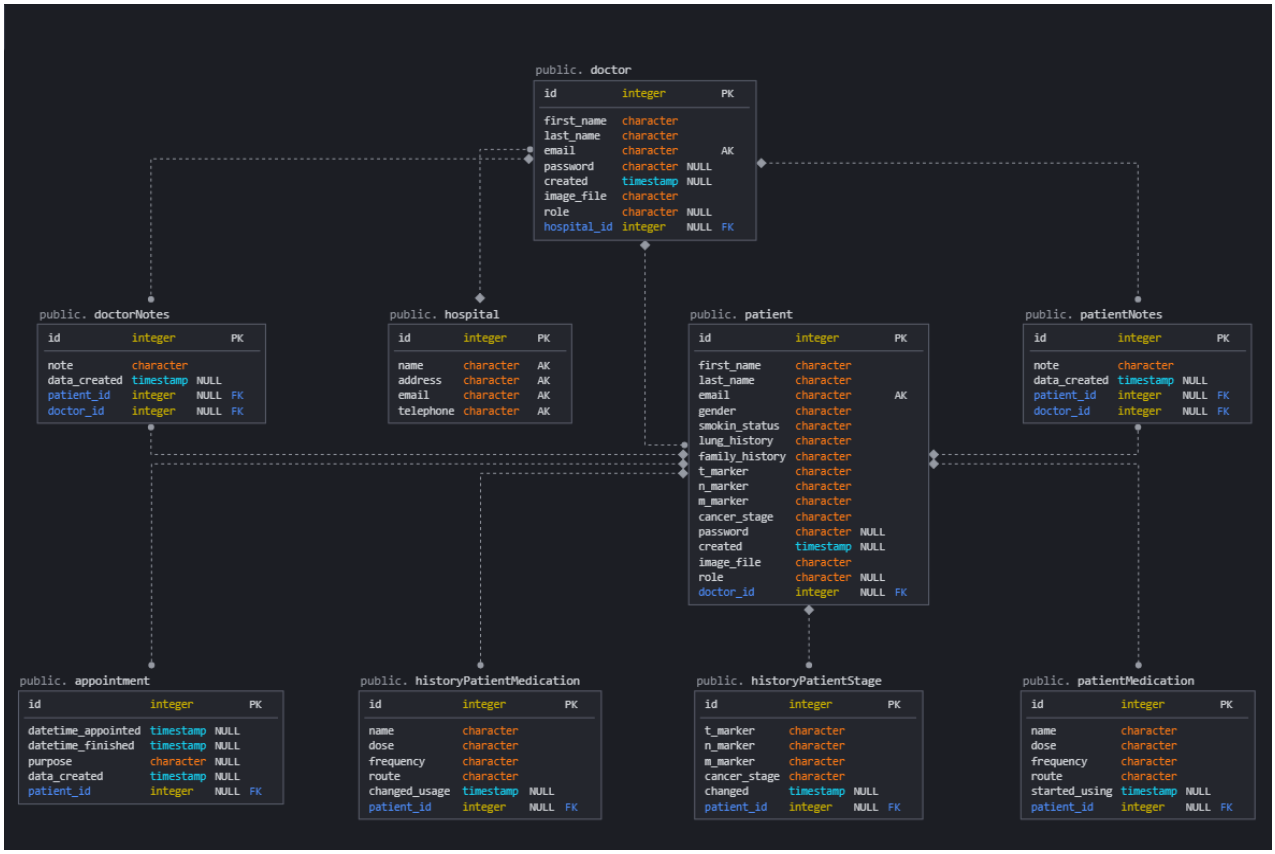
Zbog osjetljivosti kreiranog sustava i rada s osjetljivim informacijama većinu podataka i promjena se automatizirano sprema. Baze su kreirane da budu lako održive i da se mogu vrlo jednostavno proširiti dodavanjem migracija i za podatke i za shemu.

Pri izradi baze podataka kreće se od relacije bolnica-doktor. Svaki doktor može raditi samo u jednoj bolnici stoga se poslužuje relacijom N:1.

Svakom pacijentu se dodjeljuje liječnik, tako da uz poznavanje pacijenta lagano se dolazi do bilo koje vrijednosti vezano uz liječnika ili bilo koju drugu informaciju o pacijentu. Liječnik je povezan i tablicom *Notes* u kojoj svakom pacijentu možemo pokazati prilagođene obavijesti ili informacije koje im doktor pošalje. Jedan doktor može imati više obavijesti koje šalje jednom ili više pacijenata.

Jedan pacijent može imati više obavijesti koje može poslati doktoru. Relacija je ista kao kod 1:N. Pacijent može imati više sastanaka koje možemo pretražiti po pacijentima, a povezani su s tablicom doktora. Svaki pacijent za terapiju ima mogućnost primanja N lijekova. Pri svakoj izmjeni trenutne terapije iz sigurnosnih razloga i kako bi zadovoljavalo standarde spremam se trenutnu terapiju u prošlu, a nova terapija postaje trenutna. Slična situacija se događa i s promatranjem napredovanja bolesti kod pacijenta. Stadij karcinoma pluća se pamti pri svakoj

promjeni bilo koje od T,N,M vrijednosti. Svaki pacijent može imati više stadija karcinom u povijesti.



Slika 4.4 Model kreirane baze podataka izvezene iz PgAdmin-a

5. ARHITEKTURA WEB SUSTAVA

Izrada web sustava najčešće se sastoji od 3 komponente: baze podataka, poslužiteljskih aplikacija i web sučelje. Baza nam služi za pohranu i pretragu podataka. Poslužiteljska aplikacija je program koji odgovara na upite web sučelja. U najčešćoj primjeni web sučelje zatraži određene podatke, poslužiteljski dio naše aplikacije ode u bazu podataka i provjeri te informacije i u ovisnosti o traženom upiti ti podatci se mogu dohvatiti, urediti i odgovoriti na upit. Najčešće eksploatacije sustava se događaju kada korisnik pokušava poslati nedozvoljene komade koda i naredbi prema našoj bazi preko web sučelja. Web sučelje je skripta koja se izvršava u Internet pregledniku. U slučaju da nemamo primjenu programskih jezika poput JavaScripta, nego samo HTML, onda to više nije skripta nego markup. Glavni fokus web sučelja je interakcija s korisnikom, a to se može odvijati na način prikupljanja i prikaza podataka na korisniku ugodan načina. Kako bi baza podataka i aplikacija mogli komunicirati koristimo TCP/IP protokol, a za izdavanja naredbi prema bazi najčešće se koristi SQL. Kako bi poslužiteljska aplikacija i web sučelje moglo komunicirati moramo koristiti HTTP protokol. Pri radu s njima je preporučeno koristiti REST metodologiju i razmjenjivati podatke u JSON formatu. Važna komponenta koja nam olakšava svakodnevni život je i jedinstveni identifikator resursa URL (engl. Uniform Locator Resource). Svaka stranica ima jedinstvenu adresu koja se naziva URL. Olakšava nam spajanje i pamćenje samih adresa, jer u suprotnom bi morali raditi i pamtiti IP adrese stranica

5.1M HTTP

HTTP (engl. *Hyper Text Transfer Protocol)* je protokol za prijenos podataka. Koristi se na aplikacijskom sloju i omogućava web aplikacijama komunikaciju i razmjenu podataka. Baziran je na TCP/IP protokolu i omogućava preuzimanje hipertekstualnih dokumenata različitih formata i sadržaja (slike, tekst, video,..). Kako bi uopće bila omogućena komunikacija između računala i poslužitelj, obje strane moraju imati mogućnost rada s http protokolom. Sustav koji šalje zahtjev (eng. *Request)* se naziva klijent, dok poslužitelj daje odgovor (eng. *Response*). Postoje 3 važne stvari vezane za HTTP u aplikacijskom sloju:

- Prvo što moramo znati da HTTP protokol ostvaruje konekciju i klijent pravi zahtjev na poslužitelj. Nakon što je kreiran zahtjev klijent se odspaja s poslužitelja, te se ponovno spaja kada poslužitelj ima spreman odgovor. Na transportnom sloju
- HTTP može dostaviti bilo koju vrstu podataka dok obje strane pri komunikaciji mogu pročitati taj podatak

- Klijent i poslužitelj kod HTTP protokola znaju se međusobno samo za vrijeme trenutnog zahtjeva. Zatvaranje ili prekidanje trenutnog zahtjeva i želja za ponovnim spajanjem izaziva potrebu za ponovnim slanjem informacija, te se povezivanje radi kao prvi puta.

HTTP će nam biti posebno važan kada budemo kreirali rute, primali i slali podatke na sustav. Metode koje se koriste u HTTP-u su:

- GET
- POST
- PUT
- DELETE
- PATCH
- OPTIONS
- HEAD

Najčešće metode koje se koriste pri radu su GET i POST. GET metoda se nikad ne bi trebala koristiti kada se radi s osjetljivim podatcima jer se prikazuje u URL-u.

	GET	POST
Gumb Natrag / Osvježi	Bezopasno	Podatci će se ponovno predati(preglednik bi trebao obavijestiti korisnika)
Oznake	Može se koristiti za oznake	Ne može se koristiti za oznake
Predmemoriranja	Može se spremiti u predmemoriju	Ne može se spremiti u predmemoriju
Vrsta kodiranja	aplikacija/x-www-obrazci-poveznicakodirana	aplikacija/x-www-obrazci-poveznicakodirana ili višedijelni/obrasci-podataka (višedijelni obrasci se koriste kod binarnih podataka ili podataka koji nisu alfanumerički)
Povijest	Parametri ostaju u povijesti preglednika	Parametri se ne spremaju
Ograničenja na veličinu podataka	Kod slanja podataka GET ima restrikcije, jer najveća dozvoljena dužina URL-a je 2048 znakova	Nema restrikcija
Ograničenja na tip podataka	Samo ASCII znakovi dozvoljeni	Nema ograničenja svi tipovi su dozvoljeni (binarni također)
Sigurnost	Nikad koristiti pri slanju osjetljivih informacija	POST je sigurniji od GET jer se podatci ne spremaju u povijest preglednika ili u web poslužiteljima
Vidljivost	Podatci su vidljivi svima u URL-u	Podatci se ne vide u URL-u

Tablica 4.1 Usporedba GET i POST metode

Svaka HTTP poruka se sastoji od zaglavlja i tijela poruke. Kod zahtjeva šalje se metoda, putanja, verzija protokola koju koristimo, zaglavlja i tijela poruke. Metoda može biti bilo koja od gore navedenih. Najčešće klijent dohvaća podatke pomoću GET, a šalje podatke pomoću POST metoda. Putanja predstavlja URL. Verzija protokola je koju verziju HTTP-a koristimo. Prva verzija HTTP-a 1.1 je izašla 1997. godine, dok njegov nasljednik HTTP/2 izlazi 2015. godine. HTTP/3 se počeo primjenjivati 2019 godine, a specifičan je po tome što koristi UDP protokol umjesto TCP protokola u transportnom sloju. U zaglavlju se mogu nalaziti dodatke informacije o poslužitelju. Odgovor podrazumijeva verziju protokola, status kod, poruku statusa, zaglavlje i po potrebi tijelo poruke. Verzija protokola koje se pridržava, status code je obično predstavljen standardiziranim vrijednostima. Najčešći kodovi koje korisnik primjećuje su 200,400,404. 200 znači da je sve u redu, 400 da je zahtjev loš te da ga poslužitelj ne može obraditi. Iako ova prethodna 2 koda su česta korisnik ih vjerojatno ne vidi toliko često kao 404 koji predstavlja da trenutni zahtjev ne može biti ispunjen jer podatci o njemu trenutno ne postoje.

5.2 Strana klijenta i strana poslužitelja

Prilikom izrade web stranica i sustava postoje dvije metodologije prema mjestu obrade i učitavanja podataka. Podatci se mogu obrađivati na strani klijenta ili na strani posluživača.

Poslužitelj (engl. *Server*) je program koji dostavlja informacije i usluge drugim programima koji su spojeni na njega preko komunikacijskog kanala.

Klijent je program koji zatražuje pristup od poslužitelja (moguće i više njih) i traži određene zahtjeve. Poslužitelju istovremeno može pristupiti više klijenata[18].

Kod obrade podataka na strani posluživača, poslužiteljska aplikacija (u našem slučaju Flask aplikacija) nakon što dobije zahtjev na valjani jedinstveni identifikator resursa izrađuje cijeli HTML dokument i šalje ga natrag. Web preglednik dobije običan HTML dokument i prikaže ga.

Kod obrade podataka na strani klijenta, umjesto gotovo HTML-a, pošalje se JavaScript aplikacija. Aplikacija se izvršava u pregledniku na strani korisnika i ta aplikacija izradi HTML kod, koji se onda prikazuje korisniku.

Iz razlika upotrebe gore navedenih postupaka i potječu imena aplikacije na strani poslužitelja i na strani klijenta.

Oba pristupa imaju prednosti i nedostatke, a ovisno o vrsti aplikacije ili sustava kojeg izrađujemo ovisit će i odabir metoda i arhitekture. Najznačajnija je razlika količina podataka koja se šalje

između klijenta i servera. Kod pristupa na strani poslužitelja šalje se kompletna stranica i svi podatci ukomponirani unutra. Kada otvorimo novu stranicu na istom portalu, poslužitelj mora poslati sve podatke ponovno kao novu stranicu. Kod obrade podataka na strani klijenta, poslužitelj šalje JavaScript aplikaciju kada korisnik prvi puta dođe na stranicu. Prednosti rada u poslužitelj klijent arhitekturi je da se posao može rasporediti na više računala. Više klijenata može udaljeno pristupiti poslužitelju, a time se pojednostavljuje izrada radi podijele poslova između entiteta. Lakša kontrola toka programa, sigurnost spremanja podataka jer se podatci spremaju samo na strani poslužitelja, a svi klijenti mogu poslati zahtjev na te podatke i u ovisnosti o zadovoljavanju uvjeta dobiti će filtrirane podatke od strane poslužitelja. Kako bi poslužitelj i klijent uspješno mogli komunicirati potrebno je da postoje definirani jezici kojim oni komuniciraju, a mi ih nazivamo protokol. Obrada podataka na strani poslužitelja dominirala je tržištem no s povećavanjem kompleksnosti stranica, a potrebno je da se stranica svaki put cijela šalje postalo je jako zahtjevno. S izlaskom Angulara, Ember,js i Googleovom podrškom da može čitati JavaScript obrada podataka na klijentskoj strani je preuzela veliki dio tržišta. Sa sve boljom Internet infrastrukturom koja pridonosi stabilnosti i brzinama koje rastu, obrada na strani klijenta je radila dosta dobro.

Glavni problem obrade podataka na strani klijenta je potreba za slanjem cijele stranice kada se promijeni željena ruta. Zbog toga razloga i obrada na strani klijenta je dobila na popularnosti. Taj problem je riješen tako da se stranica učita prvi puta, a nakon toga radi se s programskim okvirima koji podržavaju AJAX pozive i dohvaćaju samo potrebne podatke na postojećoj stranici. Zbog tog rješenja React postaje jedan od glavnih standarda na tržištu i u samo par godina uspostavlja svoju dominaciju. Za ovakve sustave često još koristimo nazive i izomorfne ili univerzalne aplikacije. Prednosti ovakvih aplikacija su da omogućavaju kreiranje poslužiteljske i klijentske strane i omogućiti korisnicima rad s proizvoljnim tipovima podataka. Najveći nedostatak je da nam je potreban server kako bi mogli vrtjeti našu aplikaciju.

5.3 JSON

JSON (eng. JavaScript Object Notationion) je jedan od najlakših tekstualnih formata dizajniran za čitanje i razmjenu podataka razumljiv i ljudima i strojevima. Datoteke s podatcima koji se nalaze u JSON formatu imaju oznaku .js, a njihova meta oznaka application/json.

Osnovni tipovi podataka u JSON formatu su:

- Broj (eng. *Number*) – cijeli i decimalni brojevi (decimalna točka)
- Tekst (eng. *String)* – Niz od nula ili više Unicode znakova. Tekst je razdvojen s

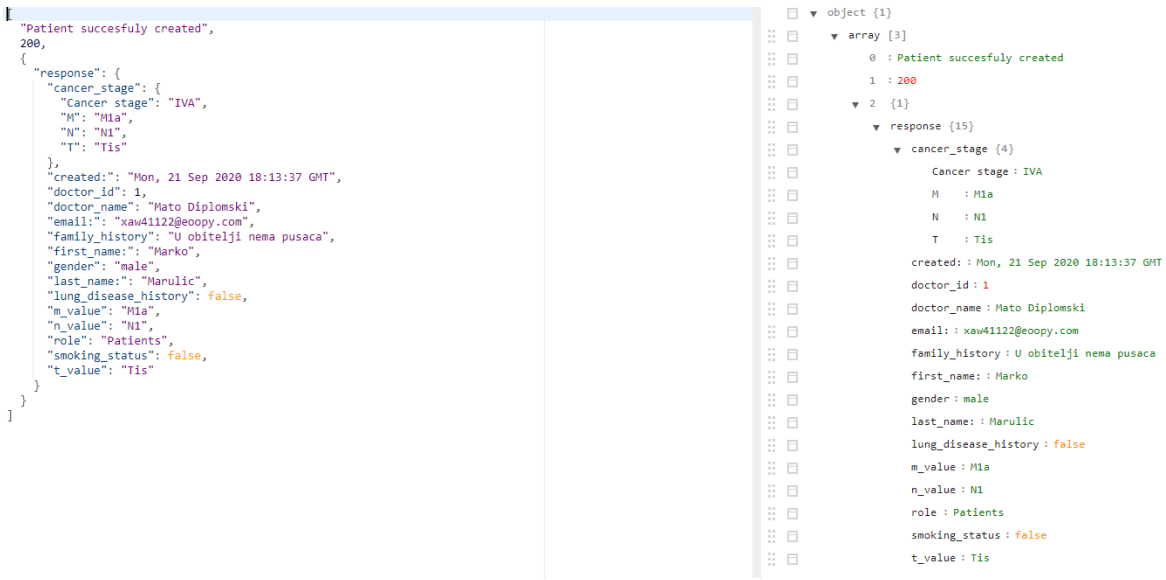
- Bool logički tip podatka (eng. *Boolean)*
- Polje (eng. *Array)*
- Objekt (eng. *Object)*
- NULL vrijednost (eng. *Null)*

JSON je izgrađen od dvije strukture objekata i polja. Objekti predstavljaju parove koji imaju ime i vrijednost, dok su polja liste vrijednosti. Zbog popularnosti gore navedenih struktura podataka i učestalosti u svim današnjim programskim jezicima ima smisla zašto je JSON izgrađen na njima. JSON format sve više zamjenjuje XML format zbog jednostavnosti u pisanju i čitanju. XML format ima oznake (eng. *Tags*) i ne može se predvoditi kroz standardne JavaScript funkcije nego je potreban zaseban prevoditelj.

Pravila sintakse kod JSON formata:

- Podatci dolaze u parovima „ime“ : „vrijednost“
- Podatci su odvojeni zarezom
- Objekti se nalaze unutar vitičastih zagrada
- Nizovi se nalaze unutar uglatih zagrada

Slika 4.1 i slika 4.2 prikazuju izgled i razliku u izgledu podataka zapisanih pomoću JSON i XML formata, koje bi dobili kao odgovor na zahtjev doktora za registracijom korisnika u aplikaciji. Primjećujemo da je struktura ostala ista, ali u JSON formatu imamo oblik ime vrijednost, dok se u XML-u koriste oznake.



Slika 5.1 Odgovora na upit za kreiranje pacijenta u JSON formatu u izrađenoj aplikaciji

Slika 5.2 Odgovora na upit za kreiranje pacijenta u XML formatu u izrađenoj aplikaciji]

6. RAZVOJNA PROGRAMSKA OKOLINA

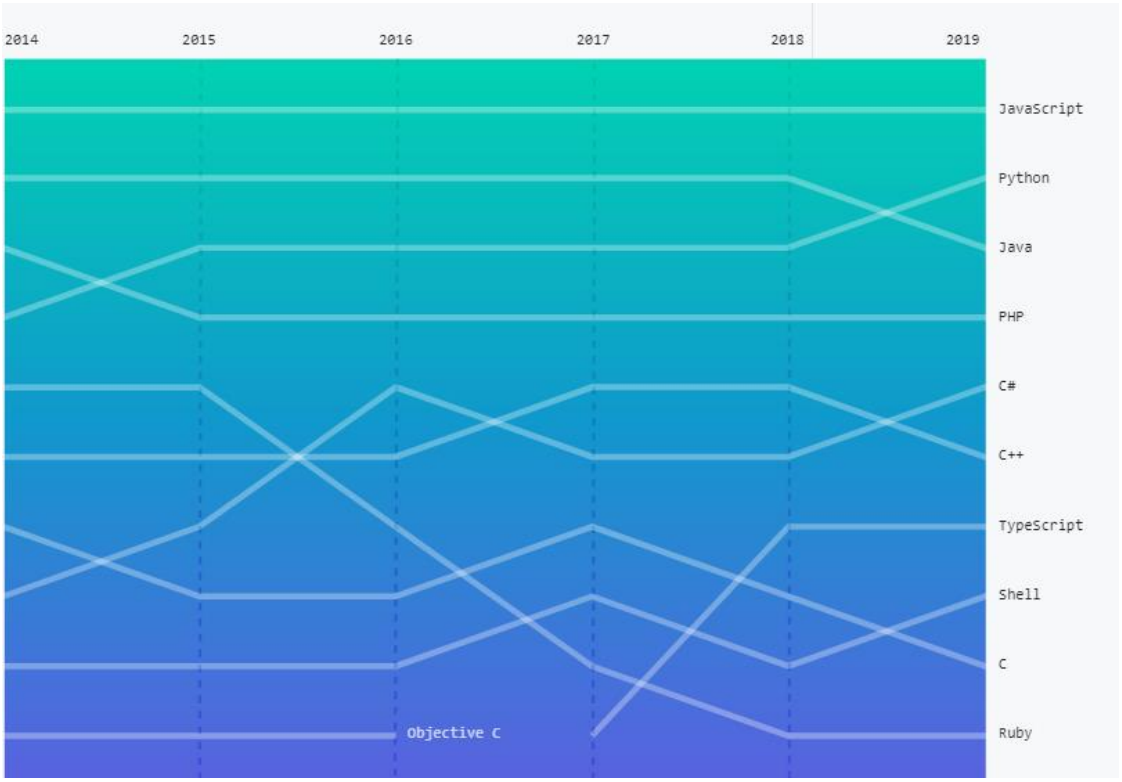
U ovom poglavlju će biti opisani osnovni pojmovi i alati potrebi kako bi mogli razumjeti programski kod, upravljanje korisnicima, sustave za autentifikaciju i korištene programske paradigme.

6.1 Python

Guido van Rossum je razvio Python 1990. godine kao projekt sa strane. Python nije doživio svoj skokoviti rast kao neki drugi programski jezici, ali se od početka trudio kako bi smanjio potrebe kako bi svi mogli početi programirati. Najveća promjena se dogodila 2008. godine kada izlazi Python 3 i donosi veliku količinu promjena.

Izvrsna integracija s C/C++ i može vrlo lagano prebacivati dijelove posla i koda iz Pythona u C, a isto tako podatci iz C-a se mogu puno brže i lakše obraditi u Python-u. Najveća razlika između Pythona i C/C++ je da za Python trebamo interpreter, dok za C/C++ je potreban kompajler.

Prednosti su jedan od najboljih podrška i količine literature zbog velike količine korisnika koje rade u Python-u. U zadnjem rangiranju od stranice *StackOverflow* Python je proglašen četvrtom najkorištenijom tehnologijom nakon JavaScripta,HTML/CSS-a,SQL-a.[14]



Slika 6.1 Zastupljenosti programskih jezika na GitHub-u [16]

Za izradu dijela na strani poslužitelja našeg sustava koristit ćemo se Pythonom. Za Python smo se odlučili iz više razloga, među njima su zastupljenost na tržištu, čitljivost, jednostavnost i lagana mogućnost nadogradnje. Veliku zastupljenost vidimo u tome da i sami tehnološki divovi koriste Python u pozadini svojih web aplikacija zbog lagane mogućnosti skaliranja. Python doživljava nevjerojatan rast zbog vrlo jednostavnog baratanja s virtualnim okolinama, postavljanjem modula, a većinska primjena u umjetnoj inteligenciji, obraditi velikih podataka, virtualnoj sigurnosti i robotici samo ubrzavaju taj rast. Velika korištenost u znanstvenim područjima i lagana implementacija postojećih rješenja na web sustave omogućava da Python bude sve više zastupljen i na web tržištu.Jedna od najvažnijih stvari pri izradi medicinskih aplikacija i sustava je i sama sigurnost , a Python i njegovi dodatci su trenutno među najboljih za sigurne razmjene podataka [15].

Jedna od velikih prednosti Pythona je njegova dokumentacija, a kada pogledamo koje sve kompanije koriste Python nije ni čudo. Među njima se nalaze: Google, Dropbox, Netflix, Instagram, Spotify, Facebook, Reddit,… Google je počeo raditi većinu stvari u Pythonu od njegovog samog početka, a jedno vrijeme im je uzrečica bila „Python gdje možemo, C++ gdje moramo.“ Što je značilo da su koristili C++ kod zahtjevnih rukovanja memorijama. Također zanimljiva činjenica je da kreator Pythona trenutno radi u Dropbox-u.

6.2 Funkcijsko programiranje

Većina računalnih kodova izvodi promjene stanja i sadržaja memorije. Kada se neki programski kod napiše, prevede u strojni kod teško će bilo što napraviti, ako ne mijenja vrijednosti u memoriji računala ili registrima procesora. Apstrakcije samih modela mijenjanja vrijednosti na sklopovlju računala dovela su do razvoja programskih kodova koji se sastoje od niza instrukcija, a rješenje problema dobiva se slijedom tih naredbi koje mijenjanju sadržaje unutar memorije ili tijek kojim se program izvodi. Takav stil pisanja programskog koda naziva se imperativni i dominira u programskim jezicima poput C-a.

Funkcija imaju važnu ulogu u matematici. Svakoj vrijednosti ulaznog skupa pridružuje vrijednosti izlaznog skupa. Pronalaze i izražavaju vezu između dva skupa podataka. Funkcije su osnovni dio funkcijskog programiranja. Programski kod u funkcijskom programiranju sastoji se od barem jedne ili više funkcija, a rezultat funkcija se dobiva izračunavanjem izraza unutar funkcije. Želja kako bi se što više određeni problem opisao matematički, umjesto da se traži sredina između računala i matematike udaljilo je funkcijsko programiranje od željenog stupnja performansi. Većinom se pričalo i spominjalo funkcijsko programiranje u akademskim krugovima. Prednost

funkcijskog programiranja je da je bliži načinu razmišljanja čovjeka nego načinu rada računala. S padom cijena računalne snage, a rastom cijene programera funkcijsko programiranje počinje biti ponovno puno više istraživano[23].

Neke od osnovnih svojstava funkcijskog programiranja su:

- Nepromjenjivost (eng. *Immutability)*- ne može se mijenjati vrijednost varijabli nakon što su inicijalizirane. Sve vrijednosti su konstantne, a jedini način da se promjeni neka vrijednost je promjenom unutar funkcije
- Čistoća (eng. *Purity)* – funkcije ne smiju utjecati na vrijednosti koje joj nisu direktno proslijeđene. Rezultat ove funkcije je da pri svakom pokretanju te iste funkcije za isti set podataka moramo dobiti isti rezultat.
- Funkcije višeg reda (eng. *Higher-order functions)* – funkcije koje primaju druge funkcije kao parametre, često povlači za sobom da funkcije dominiraju kodom te budu osnovni objekt u upotrebi

Ovo su samo neki od uvjeta, ali isto tako nisu pravilo. Postoje jezici koji ne zadovoljavaju sve od ovih uvjeta, a opet se ubraja u funkcijske jezike. Također mnogi jezici koji nisu funkcijski podržavaju rad s gore navedenim svojstvima. Primjer jezika koji zadovoljava sva tri gore navedena uvjet je Haskell.

Funkcijski jezici su deklarativni jezici koji govore računalu koji rezultat žele. U ovoj obradi funkcijskog programiranja pozabavit ćemo se s funkcijskim programiranjem u Pythonom i da li je to moguće, koliko je primjenjivo i korisno.

Određeni dijelovi Pythona zasnovani su na Haskell-u kojeg smo gore naveli kao čisti funkcijski programski jezik. Funkcijsko programiranje je jedno od često spominjanih pojmova u današnje vrijeme kada se priča o programiranju. U slučaju da se pokuša pretražiti funkcijsko programiranje na internetu pronaći će se hrpa članaka i rasprava vezana uz koji način programiranja je najbolji. U svim će se spominjati Haskell, Skala, F# i mnogi drugi gdje će se voditi rasprava da li je taj jezik čisto funkcijski ili „prljav“. U funkcijskom programiranju važna razlika je da funkcija ne mora značiti isto što i u programiranju. Funkcija se gleda s matematičkog stajališta, dok funkcija kakve većina programera poznaje smatra se procedurom. Iz tog razloga Python u kojem dominiraju promjenjivi skupovi podataka, nije baš prikladan jezik za to. Kreator Pythona Guido je sam rekao da ne voli funkcijsko programiranje i da Python nije kreiran tako da ga podržava. Određeni dijelova su se promijenili kroz povijest i izlaske novih verzija Pythona u kojem funkcija se može i nazvati „građaninom prvog reda“. Kada se spominje kreator Pythona i funkcijsko programiranje,

lambda, mape i filteri predstavljaju zanimljivosti koje on nije želio ugraditi u Python i smatra nepotrebnim iz razloga da smanjuju čitljivost, otežavaju otklanjanje greški iz koda i smanjuju produktivnost pri održavanju. Još jedna od zamjerki na račun Pythona i borbe njegovog kreatora protiv funkcijskog programiranja je problem optimizacija poziva repa (engl. *Tail-Call-Optimization*). Python je ograničen na 1000 rekurzivnih operacija, o čemu je i sam kreator pisao na svom blogu. Još neki od uvjeta za funkcijsko programiranje bi bili da se petlje se smatraju nepotrebnim, jer se sve može pomoću rekurzija. Globalne varijable nisu čiste, već se sve treba raditi s lokalnim vrijednostima. Umjesto definiranja funkcija možemo koristi lambda izraze.

Kako bi se upoznali s Pythonom i funkcijskim programiranjem u njemu i vidjeli da li je to uopće moguće pokušat ćemo slijediti pravila funkcijskog programiranja kako bi riješili gore navedene probleme, a u ovom radu pokušat ćemo naći određenu sredinu između funkcionalnosti, čitljivosti, efikasnosti i funkcijskog programiranja u Pythonu. Naravno u Pythonu nećemo moći izbjeći pridruživanja vrijednosti varijablama i nećemo moći biti uvijek u mogućnosti raditi s nepromjenjivim objektima, ali potrudim ćemo se pri izradi web sustava.

Prvo ćemo spomenuti neke od značajnih ugrađene funkcije u Python koje se koriste u funkcijskom programiranju i pokazati određena svojstva funkcijskog programiranja.

Iteratori su objekti koji predstavljaju tok podataka, možemo se kretati s jednog objekta na sljedeći pomoću *next(objekt)*.

```
test_iteratora = [1,2,3]
item = iter(test_iteratora)
item
next(item)
Konzola:
<list_iterator object at 0x0367E628>
1
```

Programski kod 6.2 Prikaz rada s iteratorima i ispisa u konzolu

Korisnost iteratora se može vidjeti i pri radu s drugim tipovima podataka koji ne spadaju u čisto funkcijsko programiranje. U Pythonu 3.7 vrijednosti u rječniku su uvijek poredane redoslijedom ubacivanja, dok u starijim verzijama to ponašanje nije bilo definirano i nismo ga mogli tako lagano kontrolirati. Prednosti korištenja vrijednosti koje su vraćene određenim vremenskim redoslijedom jako olakšava rad u određenim situacijama.

Spomenuti ćemo i ugrađene funkcije *map* i *filter* koje su funkcije višeg reda. *map(funkcija, iteratorA, iteratorB)* – Vraća iteracije od sekvenci, a u primjeru možemo vidjeti da *map* za svaki element liste primjenjuje funkciju na njega.

```
def mala(rijec):
    return rijec.lower()
list(map(mala,["DIPLOMSKI","RAD"]))
Konzola: ['diplomski', 'rad']
```

Programski kod 6.3 Prikaz rada ugrađenom s funkcijom višeg reda map

filter(uvjet,iterator)- vraća iterator nakon što je prošao kroz sve elemente koji zadovoljavaju određeni uvjet

```
def is_even(x):
    return (x % 2) == 0
list(filter(is_even, range(10)))
Konzola: [0, 2, 4, 6, 8]
```

Programski kod 6.4 Prikaz rada ugrađenom s funkcijom višeg reda filter

Python podržava neke od nepromjenjivih tipova podataka poput nizova(eng. *String),* numeričkog tipa podatka (eng. *Numeric),* parova(eng. *Tuples),…* Ako pokušavamo promijeniti vrijednost niza tako da postavimo prvi element niza na neki drugi znak dobili bi grešku u kojoj bi nam program javio *TypeError* grešku.

Jako veliki dio funkcijskog programiranja zauzimaju rekurzije, iako se većina susrela s rekurzijama u životu u svakodnevnom radu svi radije koriste petlje zbog jednostavnosti čitanja i razumijevanja samog koda. U primjeru smo prikazali jednostavnu rekurziju za računanje sume predane liste. U prvom koraku korisnik ulazi u rekurziju provjerava se i veći od duljine liste, ako je to je rubni uvijte na kojem se rekurzija prestaje izvršavati i vraća vrijednost zbroja u našem slučaju 25. U slučaju da je vrijednost varijable i manja od duljine *count* se pribroji vrijednost elementa iz liste koji se nalazi na indeksu i te se ponovno ulazi u funkciju rekurzivnim pozivom, dok se vrijednost i povećava.

```
def Sum(L, i, lenght, count):
    if lenght <= i:
        return count
    count += L[i]
    count = Sum(L, i + 1, lenght, count)
    return count


L = [1, 3, 5, 7, 9]
count = 0
lenght = len(L)
```

Programski kod 6.4 Prikaz rada s rekurzijama u Pythonu

Iako većina shvaća što znači čistoća elemenata, veliki broj ljudi se vjerojatno nije susretao s funkcijama višeg reda. Neka osnovna svojstva funkcije višeg reda je da ju moramo moći spremiti u varijablu, funkciju možemo proslijediti kao parametar drugoj funkciji i možemo vratiti funkciju iz funkcije. Funkcije s kojim radimo moramo moći spremiti u neki od tipova podataka poput liste.

```
def velika(tekst):
    return tekst.upper()


def mala(tekst):
    return tekst.lower()


def pozdrav(func):
    tekst = func("Prikaz funkcija viseg reda.")
```

Programski kod 6.5 Korištenje funkcija višeg reda

U gore navedenoj funkciji pozivamo prvo funkciju pozdrav() kojoj za parametar prosljeđujemo funkciju velika(). Iako u ovom primjeru koristimo print što se krši s načelima funkcijskog programiranja, jer ne bi trebali imati ispite unutar funkcija koristimo ga radi demonstrature.

```
PRIKAZ FUNKCIJA VISEG REDA.
prikaz funkcija viseg reda.
```

Programski kod 6.6 Prikaz ispisa za Programski kod 6.5

Jedno od najznačajnijih elemenata funkcijskog programiranja su *lambda* funkcije u kojima lambda može zamijeniti *def*. Lambda funkcija u Pythonu znači da je to funkcija bez imena, dok ako želimo kreirati funkciju s imenom moramo koristiti def. U donja dva primjera prikazat ćemo dvije najjednostavnije lambda funkcije, ali i iz njih se može shvatiti zašto ih velika većina ljudi izbjegava i jako rijetko piše u Pythonu.

```
zbrajalo = lambda x, y: x+y
def zbrajalo  (x, y):
    return x + y
ispisi = lambda naziv, vrijednost: naziv + '=' + str(vrijednost)
def ispisi  (naziv, vrijednost):
    return naziv + '=' + str(vrijednost)
```

Programski kod 6.7 Lamba operator u Pythonu

6.3 Flask

Flask za razliku od većine drugih programskih okvira, omogućava svojim korisnicima da preuzmu kontrolu nad onim žele dodati u svoju virtualnu okolinu.

Flask se izdvaja od sličnih programski okvir zato što omogućava da korisnik preuzme kontrolu. Sam odlučuje o kreativnosti i načinu izrade aplikacije. Često se naziva i mikro programski okvir (eng. *micro framework*) zato što dolazi s jako malo uključenih mogućnosti, no za većinu stvari što nam može pasti na pamet već postoje gotovi moduli koji nam mogu olakšati izradu naše aplikacije ili sustava.

Programeri se danas često moraju boriti sa samim alatima kada žele napraviti nešto izvan samih okvira alata i nemaju slobodu i kreativnost u izražavanju svojih sposobnosti. Flask je totalno suprotan. Minimalno okruženje s kojim dolazi omogućava da osoba koja radi na projektu odlučuje o fleksibilnosti, robusnosti i alatima koje će primjenjivati s njima. Odlučili se za relacijske, ORM, NoSQL baze podataka ili možda čak ne želimo imati bazu podataka izbor je na nama i Flask će nas u tome podržavati i nećemo biti ničime ograničeni. Flask je od početka svog dizajna zamišljen da bude proširiv. Naravno s proširivosti se mogu pojaviti i problemi, ako pri izradi aplikacije koristimo nestandardne module u kojima se pojavi neka ranjivost, a taj modul se više ne održava.

Za našu aplikaciju razmišljali smo između Flaska i Djanga, no na kraju smo odlučili sve izradi u Flasku zbog mogućnosti prilagodbe našim potrebama i jednostavnije izrade prototipova. Flask poput Djanga dostavlja vrhunske performanse, ali nekad ljudi znaju bježati od Flaska za kompleksnije stvari zbog potreba za konfiguracijom.

Zbog korisnika koji sam odlučuje što će ubaciti u Flask kod ostaje relativno čist i nema nepotrebnih dodataka. Uredan i strukturiran kod olakšava ispravke grešaka i testiranje, a isto tako i samu brzinu pisanja koda. Uvijeti koje smo tražili da naš programski okvir zadovoljava su apstrakcija baze podataka (rad s ORM u ovom slučaju koristimo SQLAlchemy), kompatibilan s bazom podataka koju želimo koristiti, upravljanje sa sesijama, kolačićima i siguran sustav za autentifikaciju.

Pri izboru programski okvir za medicinske svrhe možda je i važnije da je sustav jednostavniji bez puno dodataka, jer u slučaju problema lakše je pronaći kvar i zakrpati prijetnje. Također sustav mora zadovoljavati zakon o prenosivosti i odgovornosti zdravstvenog osiguranja (krat. HIPPA eng. Health Insurance Portability and Accountability Act) standarde. Pri kojima je važno osigurati privatnost pacijenata, sigurnost spremanje podataka i još veliku količinu uputnika koje sustav mora zadovoljiti ukupno negdje oko 100 pitanja.

6.4 Upravljanje podatcima i baza podataka

Python ima pakete za skoro sve baze podataka, a pošto sam Flask koji smo koristili ne pruža nikakve zabrane koju vrstu baze podataka koristi na nama preostaje odabir. Također moguće je koristiti i pakete koji uvode dodatne dijelove apstrakcije u našu bazu podataka poput PeeWee, SQLAlchemy ili Mongo Engine. Omogućavaju nam rad na većoj razini s Python objektima umjesto entitetima baze podataka poput tablica,redaka i stupaca. U ovom projektu smo se odlučili raditi s bazom podataka pomoću SQLAlchemy-a, a za bazu podataka koristiti PostgreSQL.

Pri odlučivanju koju bazu podataka koristiti imamo na izbor veliki broj uvjeta. Jednostavnost korištenja i izrade jedan je od početnih uvjeta od koji većina ljudi kreće. Apstrakcijski sloj olakšava upotrebu i rad s bazama podataka zbog pretvorbe objekata u naredbe baze podataka, a često se još naziva i ORM (eng. Object-relational mappers) ili ODM (eng. Object document mappers).

Performanse kod velikih baza podataka mogu biti značajni problemi, pa iz toga razloga se može voditi debata da li je apstrakcijski sloj toliko dobar, jer može biti nešto sporiji od niže razinskih naredbi. U većini slučajeva nećemo ni primijetiti razliku između ove dvije vrste upita, iako postoje metode optimizacija.

Također moramo paziti da naš sustav na kojem razvijamo, a kasnije postavljamo u pogon podržava rad sa željenom bazom podataka. Određeni apstrakcijski softverski okviri podržavaju rad sa samo jednom vrstom bazom podataka, ali u našem slučaju korišteni SQLAlchemy ORM omogućava rad s većinom popularnih baza podataka poput MySQL, PostgreSQL i SQLite.

U ovom slučaju morali smo paziti da imamo i omogućenu integraciju s Flaskom, ali to nam nije bio problem pošto SQLAlchemy ima posebno razvijen omotač (eng. Wrapper) Flask-SQLAlchemy.

Tablica 6.8 Usporedba PostgreSQL i MySQL-a

	PostgreSQL	MySQL
Poznata kao	Najnaprednija baza podataka na svijetu	Najpopularnija baza podataka na svijetu
Tip projekta	Projekti otvorenog tipa (eng. *Open source projects*)	
Jezik implementacije	C	C/C++
GUI alati	PgAdmin	MySql Workbench
Atomičnost, dosljednost, izolacija i trajnost (eng. Atomicity, Consistency, Isolation, and Durability)	Da	Da
Mehanizmi spremanja	Jedan	Višestruki(InnoDB i MyISAM podrška za ACID)
Analitičke funkcije	Da	Ne
Stupac indentiteta (kreiranje jedinstvenih vrijednosti u stupcu)	Da	Ne
Vrste podataka	Podržava jedinstvene tipove podataka kreirane od strane korisnika, hstore (ključ-vrijednost), polja	Klasični SQL tipovi podataka
Unsigned cijeli brojevi	Ne	Da
Bool vrijednosti	Da	Ne (TINYINT(1) se koristi za Bool tip podataka)
IP adrese	Da	Ne
Nasljeđivanje tablica	Da	Ne
Podupiti	Da	Problemi kod podupita
JSON podrška	Da	Ne

7. PROGRAMSKO RJEŠENJE

U ovom poglavlju pokazat ćemo programsko rješenje web sustava na strani poslužitelja za personalizirano praćenje pacijenata oboljelih od zloćudnih bolesti zasnovano na funkcijskom programiranju. Ovaj rad će činiti kompletnu cjelinu s radom Programsko rješenje web sustava na strani poslužitelja za personalizirano praćenje pacijenata oboljelih od zloćudnih bolesti[23]. Cijelo programsko riješenje pisano je unutar funkcija, te funkcija predstavlja građana prvog reda u ovom radu.

7.1 Konfiguracija sustava

Za početak rada prvobitno je bilo potrebno instalirati Python na računalni sustav koji smo koristili. Nakon instalacije Pythona krenuli smo s postavljanjem ostalih potrebnih dodataka. Za bazu podataka i lakšu kontrolu koristili smo PostgreSQL-ov PgAdmin4. Kolegica koja je u drugom diplomskom radu radila stranu klijenta morala je isto moći pokrenuti server i isprobati stranu poslužitelja kod sebe, pa smo morali smisliti neko rješenje kako bi rad i pokretanje bilo što jednostavnije. Koristili smo određene specifičnosti Pythona kako bi pri prenošenju modula instalacija samih bila što jednostavnija. Veliku pomoć u tome nam je radila naša virtualna okolina (eng. *Virtual enviroment*) koja u Pythonu omogućava da korisnik za pojedini projekt u svojoj virtualnoj okolini ima samo module potrebe za rad na tom projektu. Potrebne module možemo iznijeti iz Pythona, pomoću i spremiti u tekstualnu datoteku prikazanu u programskom kodu 7.1, a isto tako je prikazana instalacija svih modula iz requirements.txt.

```
pip freeze > requirements.txt
pip install -r requirements.txt
```

Programski kod 7.1. Prikaz izvoza potrebnih modula u requirements.txt i instalacije

Kako bi aplikacija radila kako treba morali smo postaviti i određene stvari vezane za inicijalizaciju naše poslužiteljske strane. Inicijalizacija aplikacije i baze podataka događa se u __init__.py. Kreiramo aplikaciju (u kodu eng. *app*) koju koristimo daljnjem radu kako bi omogućili da se samo jednom kreira i ne moramo ju postavljati više puta, te nemamo problema u slučaju da će se negdje ponavljati dva puta. Također povezali smo aplikaciju s bazom podataka koju ćemo dalje u kodu korsititi kao *db*, postavili trajanje sesije (eng. *Session life* time), bcrypt za hašanje šifri korisnika kao što smo razradili u modelu i na kraju postavili JWT (eng. *JASON Web Token*). Kako bi mogli pratiti naše korisnike odlučili smo koristiti upravitelja prijavama (eng. *Login Manager*). Za njega smo definirali početnu stranicu koju neprijavljeni korisnici nisu mogu vidjeti, te što će se

dogoditi, ako je korisnik udaljen od sustava i nema nikakvih aktivnosti na stranici. Registrirane su putanje za svaku od mogućih stranica koju korisnik može posjetiti.

Kako ne bi svaki put bilo potrebno postavljati koju aplikaciju će Flask morati pokrenuti i gdje ju pronaći dosjetili smo se kreirati *.flaskenv* u kojem smo postavili o kojoj se aplikaciji radi i postavili smo način rada te aplikacije. Aplikacija je postaviti u development način rada kako pri svakoj promjeni ne bi morali gasiti server i ponovno ga pokretati.

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_cors import CORS
from flask_bcrypt import Bcrypt
from flask_login import current_user, LoginManager
from datetime import timedelta
from flask_jwt_extended import JWTManager
app = Flask(__name__)
app.config["SECRET_KEY"] = "secretkey"
app.config["SQLALCHEMY_DATABASE_URI"] = 'postgresql://username:password@loc
alhost/postgres'
app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(minutes=20)
app.config['JWT_SECRET_KEY'] = 'secretkeyJWT'

CORS(app)
db = SQLAlchemy(app)
bcrypt = Bcrypt(app)
jwt = JWTManager(app)

login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = "/login"
login_manager.refresh_view = "/login"
login_manager.needs_refresh_message = (u"Session timedout, please re-
login")
login_manager.needs_refresh_message_category = "info"

from .routes import routes as routes_blueprint
app.register_blueprint(routes_blueprint)
```

Programski kod 7.2. Konfiguracija aplikacije unutar __init__.py

Doktorima je omogućeno postavljanje dokumenata svih formata, a svima koju su prijavljeni na sustav postavljanje svoje profile slike. Kako ne bi morali sve to spremati lokalno na našem serveru

odlučili smo riješiti taj problem pomoću vanjskih usluga poslužitelja. Razmišljali smo o dva načina postavljanja između Amazon S3 i Google servisa. U slučaju da se koriste usluge poslužitelja na Amazonu vjerojatno bi koristili S3, ali odlučili smo se za Google zbog jednostavnosti kreiranja računa i mogućnosti besplatne usluga. Morali smo definirati opseg usluga (eng. *Scope*) koje želimo omogućiti da strana poslužitelja ima i definirati gdje će se nalaziti ti podatci. Podatci se spremaju u *token.pickle* kako to više ne bi morali raditi pri sljedećem pokretanju.

```
from __future__ import print_function
import pickle
import os.path
import webbrowser
from google.protobuf import service
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
from googleapiclient.http import MediaFileUpload, MediaIoBaseDownload
from pathlib import Path, WindowsPath
import io
SCOPES = ['https://www.googleapis.com/auth/drive.file', 'https://www.googleapi
s.com/auth/drive',
          'https://www.googleapis.com/auth/drive.appdata']
```

```
def services():
    """Creating credentials and returning service
    """
    creds = None
    if os.path.exists('token.pickle'):
        with open('token.pickle', 'rb') as token:
            creds = pickle.load(token)
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                'C:/Users/Mato/Desktop/Diplomski_2/flasksite/credentials.json'
, SCOPES)
            creds = flow.run_local_server(port=0)
        with open('token.pickle', 'wb') as token:
            pickle.dump(creds, token)

    service = build('drive', 'v3', credentials=creds)
    return service
```

Programski kod 7.3. Prikaz kreiranja akredatacije za Google Drive

7.2 Postavljanje baze

Nakon definirane strukture i konfiguracije kreirana je baza podataka. Kako bi smo povezali bazu s našom aplikacijom morali smo u *models.py* proslijediti *db* kako bi mogli kreirati bazu za željenu aplikaciju. Pošto nam je kreacija tablice pacijenata najvažnija i najviše stvari o njoj objasnit ćemo i pokazati kako izgleda kod za nju. Zbog duljine koda za kreiranje tablice pacijenti podijelit ćemo je u dva dijela.

```
@dataclass
class Patient(db.Model, UserMixin):
    __tablename__ = 'patient'

    id: int
    first_name: str
    last_name: str
    email: str
    gender: str
    smokin_status: str
    lung_history: str
    family_history: str
    t_marker: str
    n_marker: str
    m_marker: str
    password: str
    created: str
    image_file: str
    role: str
```

Programski kod 7.4. Kreiranja tablice pacijenata (1.dio)

Vrijednosti unutar tablica definirane su kao klasa podataka(eng. *Dataclass)*. Parametar *db.Model* u klasi *Patient* pruža informaciju da se radi modelu baze podataka, dok drugi parametra *UserMixin* omogućava standardnu implementaciju za *flask-login* pri čemu se izbjegava potreba za definiranjem svih metoda koje *flask-login* očekuje. U njemu se nalaze funkcije *is_active(), is_authenticated(), is_anonymous(), get_id(), __eq__() i __ne__(). __eq__() i __ne__()* služe kako bi mogao provjeriti jednakost ili nejednakost dva *UserMixin* objekta pomoću *get_id()* funkcije. Koristi se jer Python2 i Python 3 ne provjeravaju jednako.

```
    id = db.Column(db.Integer(), primary_key=True)
    first_name = db.Column(db.String(45), nullable=False)
    last_name = db.Column(db.String(45), nullable=False)
    email = db.Column(db.String(75), nullable=False, unique=True)
    gender = db.Column(db.String(15), nullable=False)
    smokin_status = db.Column(db.String(15), nullable=False)
    lung_history = db.Column(db.String(250) nullable=False)
    family_history = db.Column(db.String(250), nullable=False)
    t_marker = db.Column(db.String(10), nullable=False)
    n_marker = db.Column(db.String(10), nullable=False)
    m_marker = db.Column(db.String(10), nullable=False)
    cancer_stage = db.Column(db.String(100), nullable=False)
    password = db.Column(db.String(128))
    created = db.Column(DateTime(timezone=True), server_default=func.now())
    image_file = db.Column(db.String(150), nullable=False,
                           default="default.jpg")
    role = db.Column(db.String(12), default="Patient")

    doctor_id = db.Column(db.Integer(), db.ForeignKey("doctor.id"))
    child_patienthistory = db.relationship(
        'HistoryPatient', lazy=True, cascade="save-update, merge, delete")
    child_patientmedication = db.relationship(
        "PatientMedication", lazy=True, cascade="save-update, merge, delete")
    child_patientmedication = db.relationship(
        "Appointment", lazy=True, cascade="save-update, merge, delete")
    relationship_doctorNotes = db.relationship(
        'PatientNotes', backref='doctor', lazy=True, cascade="save-
update, merge, delete")
    def __init__(self, first_name, last_name, email, gender, smokin_status, lun
g_history, family_history, t_marker, n_marker, m_marker, cancer_stage, password
, doctor_id):
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.gender = gender
        self.smokin_status = smokin_status
        self.lung_history = lung_history
        self.family_history = family_history
        self.t_marker = t_marker
        self.n_marker = n_marker
        self.m_marker = m_marker
        self.cancer_stage = cancer_stage
        self.password = password
        self.doctor_id = doctor_id
```

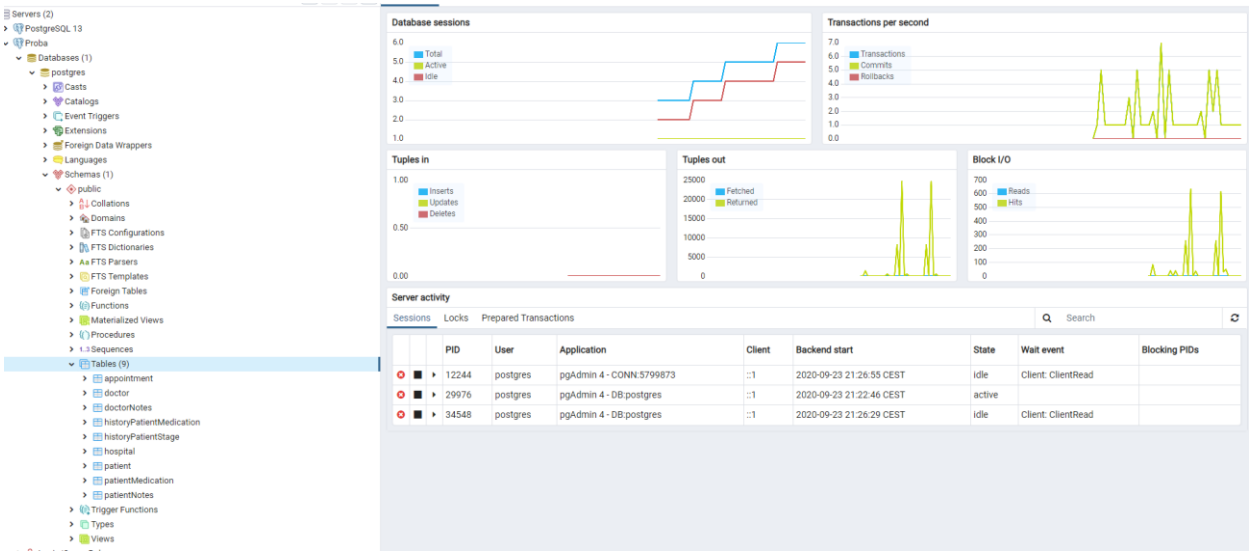
Programski kod 7.5. Kreiranja tablice pacijenata (2.dio)

U programskom kodu 7.5 definirani su svi stupci naše tablice pacijent. Postavljamo *id* (eng. *Identity*) koji je naš primarni ključ. Ime i prezime ograničavamo maksimalnu veličinu unesenog teksta (eng. *String)* za pojedino polje do 45 znakova i naznačavan se da podatci moraju biti uneseni. Kod registracije korisnika važna nam je adresa elektroničke pošte, a kako se sustav ne bi zloupotrebljavao i kako određeni korisnici ne bi pokušali koristiti istu adresu više puta postavljamo da adresa elektroničke pošte (eng. *Email*) mora biti jedinstvena (eng. *Unique*). Doktor također pri unosu pacijenta mora nam nešto reći o povijesti tog pacijenta, a pošto je pušenje jedan od najvećih uzroka raka pluća važno je znati da li je pacijent pušač i da li u njegovoj obitelji ima ljudi koji imaju problema s plućima. Kako bi mogli računati stadij karcinom pluća potrebni su nam T,N,M parametri. Doktor unosi vrijednosti pri kreiranju korisnika, a u ovisnosti o njima računa se stadij karcinom pluća. Iz sigurnosti razloga kreirana je još jedna tablica koja za promjene T,N,M vrijednosti unutar trenutne pacijent tablice prošlo stanje zapisuje u tablicu pacijentove povijesti. Također postoje polja za unos šifre u bazu podataka koju ćemo detaljnije kasnije opisati kod registracije korisnika. Pamti se vrijeme kada je pacijent unesen u sustav za praćenje raka pluća. Postavlja se početna slika za pacijenta, koju će imati mogućnost izmijeniti kada se prijavi u sustav. Također mu se pridodaje uloga da je pacijent kako ne bi morali provjeravati naknadno u sustavu da li je korisnik pacijent ili doktor, nego možemo jednostavno pozvati samo njegovu trenutnu ulogu(eng. *Role*).

Nakon kreiranja svih potrebnih vrijednosti za pacijenta potrebno je definirati relacije s ostalim tablicama u našoj bazi podatka koje će ovisiti o tablici pacijenta ili ona o njima. Kako bi povezali doktora i pacijenta moramo postaviti doktorov primarni ključ u bazu pacijenta. Doktor primarni ključ u tablici naše baze podataka postaje strani ključ, te preko njega možemo pronaći kod kojega doktora se pacijent liječi.

Nakon toga definiramo sve relacije prema djeci naše tablice pacijenata. Pošto će djeca ovisiti o tablici pacijenta važno je u svakoj od tablica djece imamo strani ključ preko kojeg ćemo moći naći pacijenta kojem pripada. U ovom primjeru vidimo da kreiramo relacije na tablice djece: povijest pacijentove bolesti(eng. *HistoryPatient*), pacijentovi lijekovi(eng. *PatientMedication*), termini(eng. *Appointment*) i bilješke (eng. *PatientNotes*). Postavljamo i konstruktor *__init__*, iako većinom bi SQLAlchemy sam dobro prepoznao i odradio zbog ozbiljnosti aplikacije odlučili smo napraviti siguran konstruktor kako ne bi imali problem kasnije. Mogli smo za parametre*__init__* koristiti (*self,**kvargs)* što bi dosta olakšalo pisanje i koji bi primio sve parametre. No ponovno iz sigurnosnih razloga i želje da znamo ponašanje aplikacije odlučili smo se za ovaj pristup jer znamo o kojim se točno parametrima radi, te smo odredili redoslijed i tip. Nakon što smo uvezli

db u *models.py* sve vezano za bazu radimo u njemu, jedino u *routes.py* kreiramo *shell_contex_processor* u kojem povezujemo nazive koje koristimo s varijablama.



Slika 7.6. Prikaz baze podataka u pgAdmin 4

Slika 7.6 prikazuje izgled kreiranih tablica u pgAdmin 4 koji je grafičko sučelje za rad s PostgreSQL tablicama. U njemu možemo pratiti spojene korisnike na našu bazu podataka, broj transakcija po sekundi, broj izmjene koje obuhvaćaju ubacivanja, pisanja i brisanja. Također se može vidjeti broj zahtjeva za čitanje i pisanje, te koliko je podataka dohvaćeno. Prednosti su što jednostavno možemo napraviti backup trenutne baze podataka, provjeriti relacije, tipove podataka u bazi, čitati podatke zapisane u bazu.

Programski kod 7.7 prikazuje kako bi naša tablica Pacijent izgledala da je napisna u SQL-u. Za razliku od programskog koda 7.5 uočava se razlika u pruženim mogućnostima i olakšavanju pisanja samog koda. Apstrakcija omogućava da se veća važnost pridoda samom rješenju umjesto postupku.

```
CREATE TABLE public.patient
(
    id integer NOT NULL DEFAULT nextval('patient_id_seq'::regclass),
    first_name character varying(65) COLLATE pg_catalog."default" NOT NULL,
    last_name character varying(65) COLLATE pg_catalog."default" NOT NULL,
    email character varying(150) COLLATE pg_catalog."default" NOT NULL,
    gender character varying(15) COLLATE pg_catalog."default" NOT NULL,
    smoking_status character varying(250) COLLATE pg_catalog."default" NOT NUL
L,
    lung_history character varying(250) COLLATE pg_catalog."default" NOT NULL,
    family_history character varying(250) COLLATE pg_catalog."default" NOT NUL
L,
    cancer_type character varying(250) COLLATE pg_catalog."default" NOT NULL,
    t_marker character varying(10) COLLATE pg_catalog."default" NOT NULL,
    n_marker character varying(10) COLLATE pg_catalog."default" NOT NULL,
    m_marker character varying(10) COLLATE pg_catalog."default" NOT NULL,
    cancer_stage character varying(100) COLLATE pg_catalog."default" NOT NULL,
    password character varying(128) COLLATE pg_catalog."default",
    created timestamp with time zone DEFAULT now(),
    image_file character varying(40) COLLATE pg_catalog."default" NOT NULL,
    role character varying(12) COLLATE pg_catalog."default",
    doctor_id integer,
    CONSTRAINT patient_pkey PRIMARY KEY (id),
    CONSTRAINT patient_email_key UNIQUE (email),
    CONSTRAINT patient_doctor_id_fkey FOREIGN KEY (doctor_id)
        REFERENCES public.doctor (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE public.patient
    OWNER to postgres;
```

Programski kod 7.7. Prikaz kreiranje tablice pacijent u SQL-u

7.3 Upravljanje korisničkim računima

Za prijavu korisnika u sustav potreban je račun elektroničke pošte i lozinka. koje su već uneseni u bazu podataka te unesene vrijednosti u polja na strani klijenta koje je izradila kolegica moraju biti točne kada se provjere s bazom podataka.

Za lakši rad sa sesijama i kako bi imali standardizirane funkcionalnosti koristimo *login_manager*. Za *login_manager* morali smo napraviti funkcije *user_loader, request_loader i unauthorized_handler* kako bi mogli baratati s korisničkim objektima. Korisničke objekte možemo osvježiti i ponovno učitati iz identifikacije spremljene u sesiju. Nedostatak pri radu na ovakav način je što se pri svakom zahtjevu učitavaju vrijednosti iz baze podataka, ali najveća prednost je sigurnost jer pri svakom zahtjevu provjeravamo autentifikaciji token. Ipak

razmišljajući o optimizaciji odlučili smo se na malo drugačiji način rada. U slučaju da korisnik zatraži neki od ne vitalnih podataka za naš sustav možemo im pristupiti preko vrijednosti spremljenih u trenutnog korisnika koji se obnavlja svakih par minuta.

Računanje autentifikacijskog tokena (eng. *Authentication token*) odvija se preko korisničke identifikacijske vrijednosti, računa elektroničke pošte i šifre. Namjerno ne koristimo ime i prezime, jer nam ti podatci ne predstavljaju sigurnosne smetnje i ne moraju biti jedinstvene vrijednosti. Scenariji u kojim korisnik promjeni svoju šifru u jednom prozoru, a ima otvoren drugi prozor koji se nalazi na našem web sučelju moramo ga izbaciti iz našeg sustava i natjerati ga da se ponovno prijavi. Zato je potrebno u autentifikacijom tokenu imati lozinku, s kojom iz sigurnosti razloga hashamo.

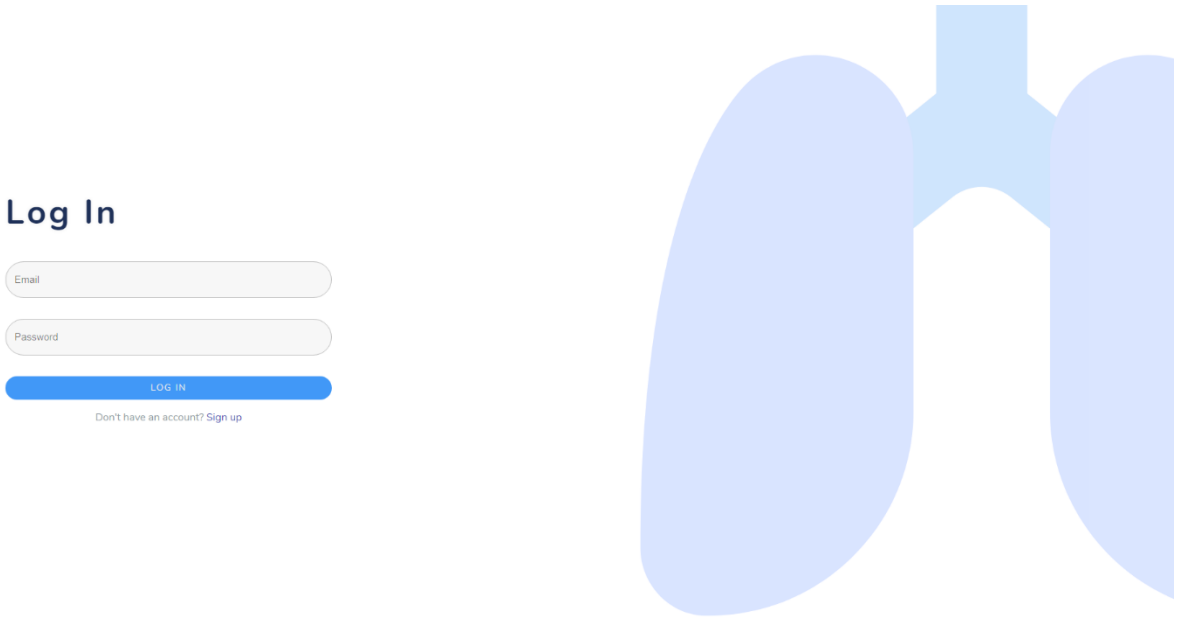
7.3.1 Prijava

Kod prijava u sustav standard je da se koristi *POST* metodom jer ne ostavlja tragove u jedinstvenom identifikatoru resursa. U novim sustavima može se koristiti i *GET* metoda, ali zahtjeva puno više pažnje i jako dobru konfiguraciju servera kako ništa od podataka ne bi bilo prikazano. Također možemo primijetiti na Slika 7.8 da imamo zahtjeve (eng. *Request*) u kojem tražimo da nam strana klijenta proslijedi podatke unutar elementa koji se zove *email* i odgovor spremamo u varijablu *email*. Također to isto radimo za korisničku lozinku. Nakon što su nam proslijeđeni podatci s korisničke strane moramo provjeriti ispravnost podataka s podatcima koji se nalaze u našoj bazi podataka. Adresa elektroničke pošte je jedinstvena u obje tablice naše baze podataka, tako da neovisno u kojoj tablici pronađemo tu adresu znamo da tražimo baš tog korisnika. U slučaju da adresa nije pronađena u tablicama doktor ili pacijent, trenutno korisnik se postavlja na ništa (eng. *None*). Također kako ne bi više puta morali pisati odgovor koji će naša funkcija vratiti strani klijenta kada se izvrši provjera postavljamo odgovor koji će zadovoljiti uvjete u slučaju da korisnik unese krive podatke. Sigurnosni razlozi nam nalažu da ne smijemo reći korisniku koji je od podataka netočan. Razlog tome je dosta jednostavan, a to je ako zna da je jedan podatak točan puno veća šansa je da će metodama pogađanja pogoditi i drugu vrijednost. Provjeravamo postoji li korisnik s tom adresom elektroničke pošte. U slučaju da postoji za tog korisnika provjeravamo hashiranu šifru unutar njegove tablice u našoj bazi podataka podudara li se s hashiranom šifrom koja je unesena na korisničkoj strani, a proslijeđena poslužiteljskoj. Također važno je znati da se šifra pomoću Bcrypta može samo šifrirati, a ne može dešifrirati. Hashirana šifra za iste podatke, isti tajni ključ i s istim parametrima šifriranja uvijek je jednaka zato možemo raditi provjere pomoću *bcyrpt.check_password_hash(šifra iz baze podataka, unesena šifra na korisničkoj strani).*

```
@routes.route("/login", methods=['POST'])
def Login():
    email = request.get_json()["email"]
    password = request.get_json()["password"]
    doctor = Doctor.query.filter_by(email=email).first()
    patient = Patient.query.filter_by(email=email).first()

    if doctor is not None:
        user = doctor
    elif patient is not None:
        user = patient
    else:
        user = None

    result = ("Incorrect username or password", 401, {
        "message": "Incorrect username or password.!",
        "error": "Unauthorized"})

    if user != None:
        if bcrypt.check_password_hash(user.password, password) != False:
            user_login = User()
            user_login.id = email
            user_login.role = user.role
            login_user(user_login, remember=True, fresh=False)
            hashed_password = bcrypt.generate_password_hash(password).decode(
"utf-8")
            access_token = create_access_token(identity={'id': user.id,
                                                         'email': user.email,
                                                         'password':hashed_pa
ssword
                                                         })
            session["token"]= access_token
            user_data = {
                "user_id": user.id,
                "first_name": user.first_name,
                "last_name": user.last_name,
                "email": user.email,
                "role": user.role
            }
            result = jsonify("user_data", user_data)
```

Programski kod 7.8. Ruta za prijavu na strani poslužitelja

Također vidimo kreiranje i spremanje autentifikacijskog tokena u kojem se nalazi korisnička identifikacija, adresa elektroničke pošte i šifra, te se to sve sprema u sesiju. Na kraju smo za uspješan ulazak u naš sustav pripremili odgovor u kojem se nalaze dogovoreni podatci koje vraćamo s poslužiteljske strane na klijentsku stranu u JSON formatu.



Slika 7.9. Izgled rute za prijavu na strani klijenta

Slika 7.9 je izgled */login* rute za prijavnu na strani klijenta, koju je izradila kolegica Maja i s koje dohvaćamo adresu elektroničke pošte i šifru.

7.3.2 Registracija

Registracija korisnika odvija se u dva dijela. Posebno je kreirana registracija za doktore, a posebno je kreirano dodavanje pacijenata. Namjerno koristimo riječ dodavanje pošto doktor mora biti prijavljen u web sustav kako bi mogao dodati pacijenta. Doktor se ima mogućnost registrirati prije prijavi u sustav, ili može sam otići na rutu za registraciju */register*.

Programski kod 7.10 predstavlja rutu za registraciju doktora. Pri registraciji koristimo GET metodu. Slično kao u prethodnim primjerima dohvaćamo podatke s klijentske strane pomoću zahtjeva kako bi dohvatili JSON za određeni element. Adresa elektroničke pošte je jedinstvena vrijednost u bazi podataka pa iz toga razloga, ako pronađemo prvu vrijednost da se poklapa s unesenom adresom elektroničke pošte možemo javiti korisniku da s unesenim podatcima nije moguće kreirati novi račun.

U slučaju da nakon provjere tablice doktora filtrirane po e-adresama nema poklapanja zaključujemo da kreirani korisnik smije imati tu e-adresu. Doktorima je omogućena kreacija profila samostalno iz tog razloga mogu unijeti željenu šifru pri registraciji. Lozinka se pomoću *bcrypt.generate_password_hash(unesena_sifra).decode(„utf-8“)* hasha.

```python
@routes.route("/register", methods=['POST','GET'])

def Register():
    first_name = request.get_json()["first_name"]
    last_name = request.get_json()["last_name"]
    email = request.get_json()["email"]
    password = request.get_json()["password"]
    created = datetime.datetime.utcnow()

    doctor = Doctor.query.filter_by(email=email).first()

    if doctor is None:
        hashed_password = bcrypt.generate_password_hash(
            password).decode("utf-8")
        new_doctor = Doctor(
            first_name,
            last_name,
            email,
            hashed_password
        )
        db.session.add(new_doctor)
        db.session.commit()

        user_data = {
            "first_name:": first_name,
            "last_name:": last_name,
            "email:": email,
            "created:": created,
            "role": "Doctor"
        }

        result = jsonify({"User succesfuly created": user_data})
    else:
        result = ("User probably exists", 401, {
                    "message": "User probably exists.!",
                    "error": "Incorect data"})


    return result
```

Programski kod 7.10. Funkcija za registracij

Pri registraciji, ako svi podatci prođu provjeru moramo unosi se novi liječnik u tablicu liječnika u bazi podataka. U *route.py* već je dodan model tablice liječnika, a podatke se unose poštivajući redoslijed konstruktora koji su postavljeni pri izradi tablice. Iako smo podatke unijeli u bazu podataka moramo proslijediti klijentskoj strani, ako je korisnik uspješno kreiran.

7.3.3 Dodavanje pacijenata

Dodavanje pacijenata obavlja se na ruti */add_patient*. Unutar funkcije *new_patient()* provjeravamo da li je trenutni korisnik (eng. *Current user*) autentificiran i da li je njegova uloga na stranici doktora. Važno je da samo doktori imaju pristup toj ruti i moraju biti prijavljeni u sustav inače će dobiti obavijest iz *unauthorized_handler* kreiranih unutar *login_managera.* Važno je napomenuti

da možemo zahtijevati da korisnik bude prijavljen pomoću jednostavnog dekoratora *@login_required* koji se nalazi unutar modula *flask_login.*

```
@routes.route("/add_patient", methods=['GET', 'POST'])
@login_required
def new_patient():
    if not current_user.is_authenticated and current_user.role == "Doctor":
        return current_user.login_manager.unauthorized()
    else:
        first_name = request.get_json()["first_name"]
        last_name = request.get_json()["last_name"]
        email = request.get_json()["email"]
        gender = request.get_json()["gender"]
        smoking_status = request.get_json()["smoking_status"]
        lung_disease_history = request.get_json()["lung_disease_history"]
        family_history = request.get_json()["family_history"]
        cancer_type = "NSCLC"
        t_value = request.get_json()["t_value"]
        n_value = request.get_json()["n_value"]
        m_value = request.get_json()["m_value"]
        cancer_stage = calculate_cancer_stage(t_value, n_value, m_value)
        doctor_id = current_user.id
        created = datetime.datetime.utcnow()
        result = "Problems with creating patient"

    doctor = Doctor.query.filter_by(email=email).first()
    patient = Patient.query.filter_by(email=email).first()
    result = ""
    if doctor is None and patient is None:
        password = generate_password()
        subject = "Generated password for Lung Cancer"
        send_mail  = mail_to(email,subject,password)
        if send_mail["status_code"] == 200 or password != "":

            hashed_password = bcrypt.generate_password_hash(
                password).decode("utf-8")

            new_patient = Patient(
                first_name,
                last_name,
                email,
                gender,
                smoking_status,
                lung_disease_history,
                family_history,
                cancer_type,
                t_value,
                n_value,
                m_value,
                cancer_stage["Cancer stage"],
                hashed_password,
                doctor_id
            )
            db.session.add(new_patient)
            db.session.commit()

            user_data = {
                "first_name:": first_name,
```

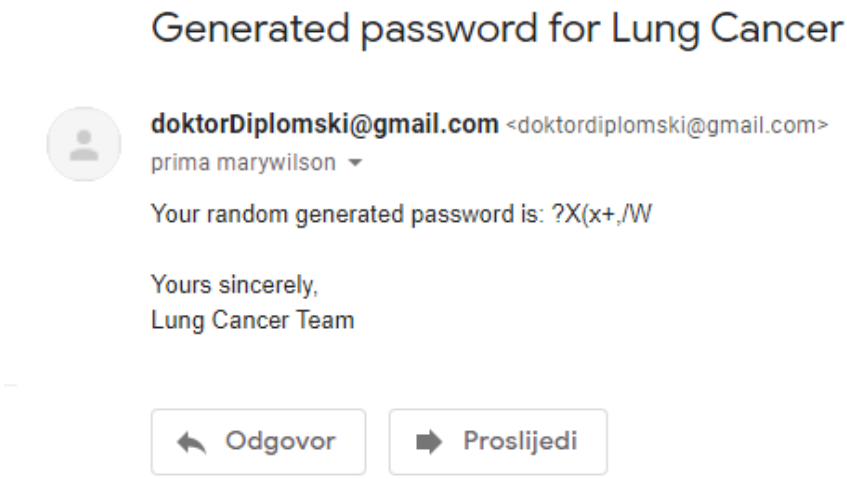
```
                "last_name:": last_name,
                "email:": email,
                "gender": gender,
                "smoking_status": smoking_status,
                "lung_disease_history": lung_disease_history,
                "family_history": family_history,
                "cancer_type": cancer_type,
                "t_value": t_value,
                "n_value": n_value,
                "m_value": m_value,
                "cancer_stage": cancer_stage,
                "created:": created,
                "doctor_id": doctor_id,
                "doctor_name": str(current_user.first_name) + " " + str(current_user.
last_name),
                "role": "Patients"
            }
            result = jsonify('Patient succesfuly created', 200, {
                      'response': user_data})
            print(result)
        else:
            result = jsonify("Error", 400, {"error": "Problems with generating passwo
rd or sending mail"})
    else:
        result = jsonify("Error", 400,{"error": "Email is alredy being used"})

    return result
```

Programski kod 7.11 Funkcija add_patient za dodavanje novih pacijenata

Podatci se spremaju unutar rječnika te se vraćaju početnoj funkciji. Provjerava se da li su vrijednosti za doktora i pacijenta prazne. U slučaju da postoje ne može se dodati pacijent u tablicu pacijenata. Nakon utvrđivanja da je adresa elektroničke pošte jedinstvena i svi podatci ispravni generira se početna šifra za korisnika. Početna duljina lozinke je postavljena na 8 znakova, iako se može generirati lozinka bilo koje duljine.

Ranije smo spomenuli da smo pokušali kreirati sustav koji zadovoljava što više zakona o prenosivosti i odgovornosti zdravstvenog osiguranja tako da smo odlučili korisniku slati mail s njegovom zaporkom kako zaporka ne bi bila komprimirana od strane doktora. Mail se šalje automatski pri dodavanju korisnika na unesenu adresu elektroničke pošte. Kako bismo mogli uspješno slati mailove s našeg servera morali smo postaviti je SMTP protokol za komunikaciju.

Podatci se spremaju unutar rječnika te se vraćaju početnoj funkciji. Provjerava se da li su naše vrijednosti za doktora i pacijenta prazne. U slučaju da postoje ne može se dodati pacijent u tablicu pacijenata. Nakon što smo utvrdili da je adresa elektroničke pošte jedinstvena i svi podatci ispravni moramo generirati početnu šifru za korisnika.



Slika 7.12 Izgled maila poslanog sa strane poslužitelja

Mail se šalje automatski pri dodavanju korisnika na unesenu adresu elektroničke pošte. Kako bismo mogli uspješno slati mailove s našeg servera morali smo postaviti je SMTP protokol za komunikaciju. Također postavili smo korisnički račun koji će predstavljati glavni korisnički račun s kojeg će se slati elektronička pošta. Korisnički račun koristi OAuth2 autentifikaciju kako bi se povezao na Google Gmail usluge. Kako si bi olakšali slanje mailova koristimo *yagmail* dodatak za Python koji u sebi ima olakšan rad s STMP protokolom i postavljanje Google računa, na siguran način, kako ne bi došlo do nepotrebnih gubitaka lozinke.

Pošto je koncept samog ovog sustava da bude iskoristiv i pisan funkcijski ,slanje elektroničke pošte je napisano unutar funkcije Programski kod 7.13 koja se može prilagoditi potrebama sustava. U slučaju da promijenimo sustav za slanje elektroničke pošte možemo lagano implementirati u već postojeću funkciju. Omogućeno je i slanje elektroničke pošte više korisnika odjednom samo unutar dijela kome se šalje moramo predati rječnik ili listu korisnika kojima je potrebno poslati elektroničku poštu. Poruka se može prilagoditi potrebi isto kao i naslov pri slanju u funkciju, iako je u ovom dijelu rada pokazano slanje s početnom porukom.

```
def mail_to(to, subject, password):
    '''Send email to user
    '''
    response = {"status_code": 400,  "Status": "Mail not send"}
    contents = [
        "Your random generated password is:" + password,
        "You can change it anytime you want under your profil"
        "Yours sincerely,"
        "Lung Cancer"
    ]
    if yagmail.SMTP("doktorDiplomski@gmail.com").send(to, subject, conten
ts):
        response = {"status_code": 200, "Status": "Mail successfuly sende
d"}

    return response
```

Programski kod 7.13. Prikaz funkcije za slanje mailova, sa predefiniranim tekstom korisnicima

Ugradili smo promjenu za slanje poruka kako bismo bili sigurni da je pacijent dobio šifru, jer se korisnik ne može registrirati ponovno s istom adresom elektroničke pošte i bila bi potrebna intervencija administratora kako bi se promijenila zaporka, koja bi se prije samoga unosa u bazu morala hashirati inače bi nastao problem. Nakon toga prilagođavamo podatke unutar tablice pacijent u bazi podataka. Podatci se nakon obrade moraju prilagoditi i poslati na stranu klijenta dogovorenim redom.

7.3.4 Računanje stadija karcinoma pluća

Dohvaćamo podatke o pacijentu sa strane klijenta isto kao što je već gore pokazano. Specifično je što nakon dohvaćanja podataka za T,N i M vrijednost moramo izračunati za svakog pacijenta razinu karcinom pluća. To radimo pomoću funkcije *calculate_cancer_stage(T,N,M)* kojoj predaje unesene vrijednosti za T,N,M.

Funkciju *calculate_cancer_stage(T,N,M)* računamo po internacionalnom sustavu za određivanje stadija karcinom pluća. Kod ovakvog sustava koji se mijenja učestalo mora postojati mogućnost mijenjanja vrijednosti unutar funkcije. Moramo omogućavamo da *stages* bude lagano promjenjiv sa svakim novim izdanjem *AJJC CANCER STAGE MANUAL* [13] kako bi vjerodostojnost naših podataka bila što veća, a pacijenti dobili pravovremeni tretman i informacije. Podatke vraćamo natrag s vrijednostima koje su mu predane radi dodatne provjere.

```python
def calculate_cancer_stage(T, N, M):

    stages = {"TX": ["Occult", "Occult", "Occult", "Occult"],
              "Tis": ["0", "0", "0", "0"],
              "T1a": ["IA", "IIA", "IIIA", "IIIB"],
              "T1b": ["IA", "IIA", "IIIA", "IIIB"],
              "T2a": ["IB", "IIB", "IIIA", "IIIB"],
              "T2b": ["IIA", "IIB", "IIIA", "IIIB"],
              "T3": ["IIB", "IIIA", "IIIB", "IIIC"],
              "T4": ["IIIA", "IIIA", "IIIB", "IIIC"],
              "M1": ["IVA", "IVA", "IVA", "IVA"],
              "M1a": ["IVA", "IVA", "IVA", "IVA"],
              "M1b": ["IVB", "IVB", "IVB", "IVB"],
              }
    if N[1] == "X":
        cancer_stage = "Nearby lymph nodes cannot be assessed due to lack of information."
    elif T == "T0":
        cancer_stage = "There is no evidence of a primary tumor."
    elif M == "M1":
        cancer_stage = stages[M][int(N[1])]
    elif M == "M0":
        cancer_stage = stages[T][int(N[1])]
    elif M != "M0":
        cancer_stage = stages[M][int(N[1])]

    result = {"T": T, "N": N, "M": M, "Cancer stage": cancer_stage}

    return result
```

Programski kod 7.14. Funkcija za računanje stadija karcinoma pluća

7.4 Dodavanje lijekova, terapija i termina

Kako bismo mogli pratiti stanja pacijenata potrebni su nam podatci vezani uz njihove terapije i lijekove koje uzimaju. Morali smo kreirati zasebne rute za dodavanje lijekova, uređivanje terapije pacijenata i promjenu količine lijekova, dodavanje zakazani termina posijete doktoru i mogućnosti praćenja povijesti.

Doktor za svakog pacijenta mora moći dodati njegovu terapiju kako bi mogao pratiti napredak i liječenje. Na strani klijenta doktor ima mogućnost pacijentu dodati lijek, a klijentska strana prosljeđuje pacijentov identifikacijski broj, *login_manager* kojeg smo kreirali na početku i koji cijelo vrijeme nadzire naš sustav, može dostaviti o kojem prijavljenom korisniku se radi pomoću *current_user*.

```python
@routes.route("/add_medication/<int:patient_id>",methods=['GET','POST]')
@login_required
def add_medication(patient_id):
    if not current_user.is_authenticated:
        return current_user.login_manager.unauthorized()
    else:
        name = request.get_json()["name"]
        dose = request.get_json()["dose"]
        frequency = request.get_json()["frequency"]
        route = request.get_json()["route"]
```

```python
    new_medication = PatientMedication(
                name,
                dose,
                frequency,
                route,
                patient_id
    )
    db.session.add(new_medication)
    db.session.commit()

    user_data = {
    "name": name,
    "dose": dose,
    "frequency": frequency,
    "route": route
    }

    status_code = 200

    return jsonify({"status code": status_code, "result": user_data})
```

Programski kod 7.15. Dodavanje lijekova za odabranog pacijenta

Dohvaćamo vrijednosti koje je doktor upisao u formu na klijentskoj strani te sve te podatke spremamo u tablicu pacijentovi lijekovi (eng. *Patient Medication*). Vraćamo korisničkoj strani unesene podatke kako bi mogli biti prikazani korisniku.

Funkcija *patient_medication()* imamo provjeru o pravima pristupa ovoj ruti, te primjer korištenja *serialize* funkcija zbog promjene u tipu podataka. Pošto vraćeni tip podatka nije pogodan za ispis morali smo kreirati serijaliziraj funkciju koja će pokupiti vraćene podatke i prilagoditi ih za ispis.

Iako je korištena petlja u ovoj funkciji, koriste se elementi funkcijskog programiranja poput čistoće elemenata i nepromjenjivosti.

```
@routes.route("/medication",methods=['GET','POST'])
@login_required
def patient_medication():
    result =""
    if not current_user.is_authenticated:
        return current_user.login_manager.unauthorized()
    else:
        user_datas = PatientMedication.query.filter_by(patient_id=current_user.i
d).all()
        result = serialize_list(user_datas)

    return jsonify({"status code": 200, "result": result})

def serialize(userdata):
    return {c: getattr(userdata, data) for data in inspect(userdata).attrs.keys(
)}

def serialize_list(user_datas):
    return [serialize(userdata) for userdata in user_datas]
```

Programski kod 7.16. Popis terapija

Doktor mora imati mogućnost dodati kada će koji pacijent imati pregled, termin liječenja ili bilo koju drugu potrebu dolaska kod njega. To smo riješili na ruti */add_apppointment* za koju je kreirana funkcije *new_appointment()*. Doktor u formi na strani klijenta unosi podatke o pacijentu. Predviđeno vrijeme trajanja susreta, vrijeme susreta i razlog dolaska. Na strani poslužitelja moramo zatražiti podatke sa strane klijenta. Želimo da nam zahtjev vrati odgovor u JSON obliku, gdje će vrijednosti koje preuzimamo sa strane klijenta biti zadana unutar uglatih zagrada. Vrijednosti unutar uglatih zagrada moraju se isto nazivati i na strani klijenta inače strana poslužitelja neće moći dobaviti podatke. Sve dobivene vrijednosti spremamo u tablicu sastanaka u bazi podataka.

Kako bi korisnička strana to sve mogla prikazati moramo vratiti cijeli objekt pacijenta, vrijednosti za doktora i vrijednosti za susret u JSON formatu.

```python
@routes.route("/add_appointment", methods=['GET', 'POST'])
@login_required
def new_appointment():
    patient_id = request.get_json()["patient_id"]
    datetime_appointed = request.get_json()["datetime_appointed"]
    datetime_finished = request.get_json()["datetime_finished"]
    purpose = request.get_json()["purpose"]

    new_appointment = Appointment(
        datetime_appointed,
        datetime_finished,
        purpose,
        patient_id
    )

    db.session.add(new_appointment)
    db.session.commit()
    patient = Patient.query.get_or_404(patient_id)
    user_data = {
                    "patient_id": patient_id,
                    "first_name:": patient.first_name,
                    "last_name:": patient.last_name,
                    "email:": patient.email,
                    "gender": patient.gender,
                    "smoking_status": patient.smoking_status,
                    "lung_disease_history": patient.lung_disease_history,
                    "family_history": patient.family_history,
                    "cancer_type": patient.cancer_type,
                    "t_value": patient.t_value,
                    "n_value": patient.n_value,
                    "m_value": patient.m_value,
                    "cancer_stage": patient.cancer_stage,
                    "created:": patient.created,
                    "doctor_id": current_user.doctor_id,
                    "doctor_name": str(current_user.first_name) + " " + str(curr
ent_user.last_name),
                    "role": current_user.role,
                    "datetime_appointed": datetime_appointed,
                    "datetime_finished": datetime_finished,
                    "purpose":purpose
                }
    response = jsonify(user_data)

    return response
```

Programski kod 7.17. Dodavanje novog termina

7.5 Komunikacija i razmjena bilješki, personalizacija profila i rukovanje datotekama

Problemi ili nedoumice koje se mogu stvoriti kod pacijenta, pitanja na koja ne može naći odgovor ili jednostavno potreba da se čuje s liječnikom. Sve smo to omogućili tako da dozvoljavamo dodavanje bilješki koje se prikazuju na strani klijenta kada uđe u sustav. Komunikacija nije jednosmjerna već dvosmjerna što znači da isto tako doktor može javiti ili poslati obavijest pacijentu.

```
@routes.route("/add_notes", methods=['GET', 'POST'])
@login_required
def add_notes():
    note = request.get_json("note")
    patient_id = request.get_json("patient_id")

    results = jsonify("Success", 200, {"success": "New note added"})
    if patient_id is not None and current_user.role == "Doctor":
        new_note = Notes(
            note,
            patient_id,
            current_user.id
        )
        db.session.add(new_note)
        db.session.commit()
    elif current_user.role == "Patient":
        new_note = Notes(
            note,
            current_user.id,
            current_user.doctor_id
        )
        db.session.add(new_note)
        db.session.commit()
    else:
        results = jsonify("Error", 400, {"error": "Problems with sending note"})

    return results
```

Programski kod 7.18. Dodavanje nove bilješke

Svi podatci uneseni se moraju moći nadzirati u slučaju nepravilnosti ili problema kako bi se mogle poduzeti odgovarajuće mjere. Za uređivanje podataka unutar pacijenta imamo tablice koje pamte prošla stanja kako bi tablice od kojih zahtijevamo rezultate češće bile što responzivnije.

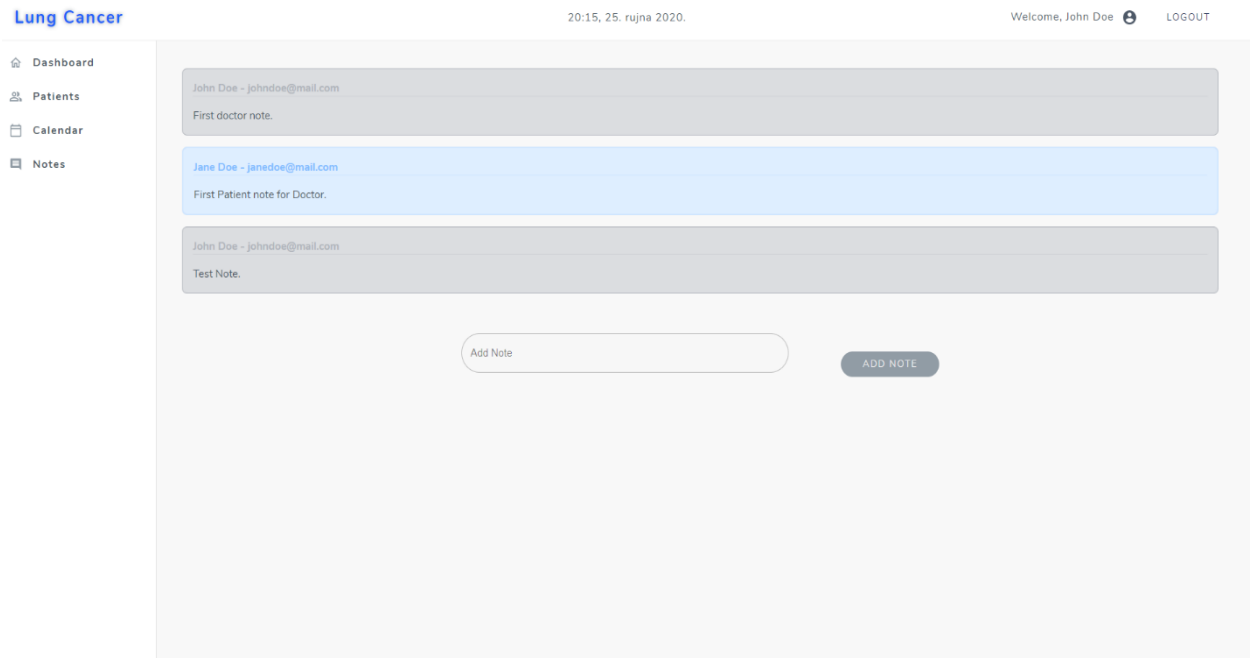
Trudili smo se pri izradi baza da budu što responzivnije i lakše za pretraživanje zbog velike ovisnosti baza o sličnim parametrima, tako da nam se pojavljuju slučajevi gdje pretražujemo ili spajamo 3 ili 4 tablice, dok za određene slučajeve bi se moglo zatražiti i više. Za određene slučajeve imamo funkcije višeg reda koje nam pomažu u nepotrebnom dupliciranju koda i olakšavaju posao. Također potrudili smo kreirati što više funkcija koje odrađuju pojedinu stvar kako bi nam rezultati za iste unose podataka bili isti.

Funkcija *list_notes* nalazi se na ruti */list_notes* i prilagođava ispis ovisno da li pacijent ili doktor uđe na nju. Primjer je jedna od više funkcija u kojim istovremeno tražimo poklapanja u više tablica. Želimo iz tablice doktora, pacijenta i bilješki pronaći povezana zapise. Nakon što smo rekli o kojima se tablicama unutar naše sesije radi, zbog visoke razine apstrakcije koju nam *SQLAlchemy* omogućava taj problem je jako jednostavno riješen. Tražimo gdje se primarni ključ od doktora poklapa sa stranim ključem unutar pacijenta, isto tako potrebno nam je pronaći gdje se nalazi primarni ključ pacijenta u zabilješkama stranih ključeva. Kada su ta dva uvjeta ispunjena onda možemo reći po kojem uvijete pretražujemo, a to je u ovom slučaju da vrijednost stranog ključa u tablici bilješki mora biti jednaka vrijednosti primarnog ključa trenutnog korisnika. U slučaju da radimo prikaz za pacijenta razlika je da od trenutnog korisnika gledamo vrijednost stranog ključa koji je doktorov primarni ključ. *.all()* Označava da želimo ispis za sva poklapanja kako bi imali sve zabilješke na jednom mjestu, iako se može filtrirati koliko od njih želimo prikazati korisniku.

```
def list_notes():
    patient_id = request.get_json("patient_id")
    list_of_notes = "Empty"
    if patient_id is not None and current_user.role == "Doctor":
        list_of_notes = db.session.query(Doctor, Patient, Notes,).filter(Doctor.id
== Patient.doctor_id,).filter(Patient.id == Notes.patient_id,).filter(
            Notes.doctor_id == current_user.id,).all()
    elif current_user.role == "Patient":
        list_of_notes = db.session.query(Doctor, Patient, Notes,).filter(Doctor.id
== Patient.doctor_id,).filter(Patient.id == Notes.patient_id,).filter(
            Notes.doctor_id == current_user.doctor_id,).all()

    return jsonify({"status code": 200, "result": list_of_notes})
```

Programski kod 7.19. Funckija za ispis bilješki

Strana poslužitelja i klijenta moraju komuniciraju kako bi se mogli dogovoriti kako će naši podatci koje šaljemo i primamo biti prikazani pacijentu i doktoru. U slučaju da se dogodi nesuglasice u izmjenama podataka može se dogoditi da se dogode sigurnosni propusti jer se pojavljuju vrijednosti koje ne pripadaju skupu podataka, a mi ih ne možemo prepoznati. Prednost primjene slanja specifičnih vrijednosti i parametara za koje znamo što točno predstavljaju je lagana manipulacija i provjera da nisu maliciozni dijelovi koda. Jedna od često zanemarenih stvari u Pythonu je predefinicija tipa podataka. Za razliku od C/C++ u kojima je potrebno najaviti tip podataka za varijable u Pythonu to nije potrebno, to ostavljamo interpreteru da sam odluči. Jedno od optimizacija kako bi ubrzali responzivnost strane poslužitelja može biti predefiniranje tipa podataka.



Programski kod 7.20. Prikaz bolješki na strani klijenta

Važan dio ovog sustava je i samo spremanje vrijednosti i dokumenata u našem sustavu. Kako bi sustav bio što više prilagođen doktoru i pacijentu i u ovako teškim razdobljima olakšao neke osnovne funkcije kako se ne bi morali o njima brinuti morali smo riješiti spremanje dokumenata i mogućnost pristupa samim u digitalnom formatu. Doktor može postaviti dokumente za pacijenta kojima i doktor i pacijent moraju moći pristupiti u svakom trenutku. Za to smo se odlučili da ćemo postaviti dokumente različitih formata na Google Drive kako smo već ranije spomenuli.

Google drive je usluga koja je pokrenuta 2012. godine i za početne korisnike koji su besplatni nudi 15GB prostora u vrijeme pisanja ovog diplomskog rada. Zbog same sigurnosti koje Google usluga pruža ipak smo se odlučili da bi to bilo presudno u ovom našem slučaju. Nakon što smo postavili račun, odradili konfiguraciju i autentifikaciju možemo početi koristiti Google usluge. Odlučili smo se da će to sve biti riješeno na strani poslužitelja kako bi se strana korisnika mogla više brinuti kako će korisnicima olakšati rad zbog velike razlike u informatičkoj spremnosti oboljelih željeli smo sustav napraviti što intuitivnijim i lakšim za korištenje.

Korisnik izabere datoteku za postavljanje na Google Drive, sustav prvo provjerava da li ima potrebne privilegije kako bi mogao koristiti tu uslugu. Ako korisnik nema privilegije ili je prijavljen na stranicu, a izmijenio je neki od ključnih parametara za naš sustav.

```
@routes.route("/upload", methods=['GET', 'POST'])
@login_required
def upload():

    result = ""
    file_path = request.get_json()["file_path"]
    parentID = "1ZTiDS_q4yjXdRuu8eOXuls5dGQo8x7b"
    service = services()
    items_inside = list_all(parentID, service)
    folder_id = ""
    exist = False

    if not current_user.is_authenticated:
        return current_user.login_manager.unauthorized()
    try:
        result = jsonify(
            {"status_code": 200, "Message": "Upload is succesfuly finished."})
        print(result)
    except:
        result = jsonify(
            {"status_code": 400, "Message": "Upload is unsuccesfuly finished."
})

    for item in items_inside:
        if current_user.email == item["name"]:
            folder_id = item["id"]
            exist = True
            break

    if exist == False:
        folder_id = create_new_folder(current_user.email, parentID, service)
        exist = True
    if exist == True:
        if folder_id != "":
            uploader = upload_new_file(file_path, service, folder_id)

    result = jsonify({"status_code": 200, "response": items_inside})
```

Programski kod 7.21. Prikaz funkcije i rute upload na Google Drive

Provjerimo da li je korisnik autentificiran, ako nije izbacimo kreiranu funkciju na početku za neovlašteno rukovanje s aplikacijom. U slučaju da je sve u redu provjerava se da li je autentifikacija s Google uslugom valjana pomoću dobivenog odgovora od strane njihovog poslužitelja.

Google Drive nema klasičnu stablastu raspodjelu sustava datoteka, pa moramo paziti kako će naš sustav spremati i znati koja datoteka pripada kojem korisniku. Problem smo riješili generiranjem datoteka sa specifičnim parametrima koje opisuju pacijenta u koje možemo spremati korisničke podatke. Pacijenti koji već imaju postavite datoteke u naš sustav pri pozivima s klijentske strane */upload* rute, imaju kreiranu svoju datoteku unutar roditeljske datoteke.

Odgovor koji ćemo vratiti strani klijenta u slučaju da je datoteka uspješno kreirana, i željeni dokument postavit biti će status kod 200 i predefinirani odgovor, a u slučaju da se dogodila greška poslat ćemo 400 kako bi strana klijenta mogla ispisati obavijest sukladnu tom status kodu i poduzela potrebnu akciju u vezi korisnika.

Kako bi strana klijenta mogla prikazati profile slike koje se isto spremaju u posebnu strukturu na našem Google Drive servisu. Morali smo kreirati generator linkova koji će poslati link na klijentsku stranu i omogućiti prikaz korisnicima. Također kako ne bi imali problema s gubitkom profilne slike te podatke moramo spremiti kao izmjenu u tablici pacijent u bazi podataka. U poglavlju 5.1 spomenuto je da izmjene korisničkih vrijednosti najčešće se rade preko PUT metode.

Uz sustav koji omogućava postavljanje dokumenata korisnik mora imati neki način na koji će moći te dokumente i preuzeti. Za svaki file koji postavimo imamo njegovu identifikacijsku oznaku preko koje možemo pristupiti toj datoteci. Pošto samo skidamo, a ne mijenjamo stanje na serveru koristimo metodu GET. POST se većinom koristi kada se pri skidanju rade određene izmjene na serveru.

```
@routes.route("/download/<file_id>", methods=['GET'])
@login_required
def download(file_id):

    status_code = 400
    service = services()
    status = download_file(file_id, service)
    if status != None:
        response = "File is succesfuly downloaded"
        status_code = 200
    response = ("Status code", status_code, {"response": status})

    return response
```

Programski kod 7.22. Prikaz funkcije i rute download sa Google Drive

Za većinu ovih ruta postoje i rute za uređivanje samih podataka i mijenjanje vrijednosti tablica, funkcije za obradu podataka, provjeru baza, pokretanje u pozadini, ali zbog količine koda koja bi zatrpala ovaj diplomski prikazat ćemo samo još jednu funkciju koja se bavi izmjenom podataka unutar tablice pacijent.

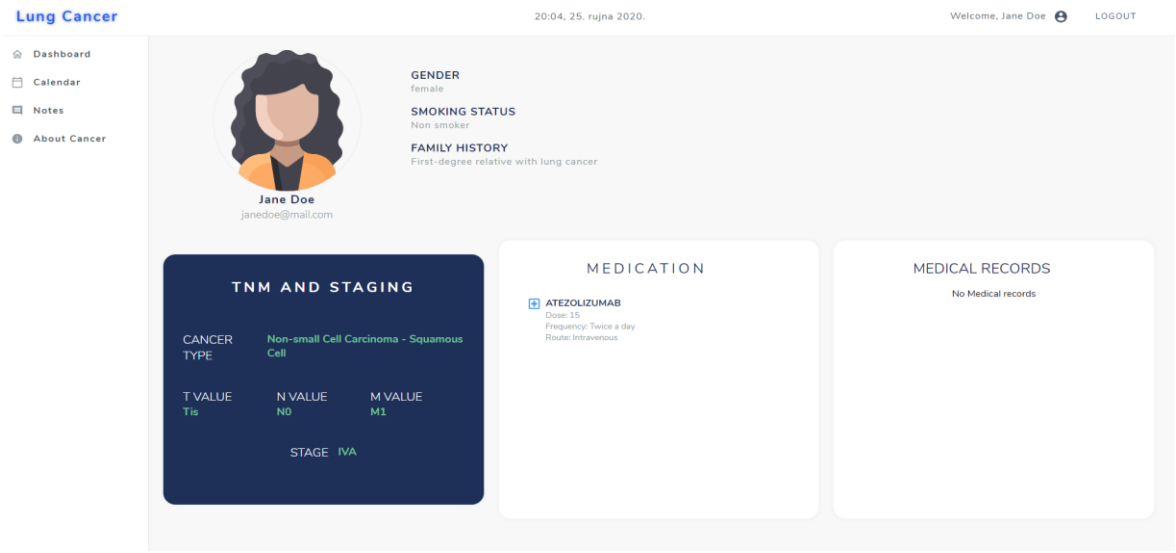
```
@routes.route("/edit_patient/<int:patient_id>", methods=['GET', 'POST'])
@login_required
def EditPatient(patient_id):

    status_code = 200
    doctor_id = current_user.id
    changed = False
    if doctor_id == None:
        result = "Doctor not found."
        status_code = 400

    patient = Patient.query.get_or_404(patient_id)
    if request.get_json()["first_name"] is not None:
        first_name = request.get_json()["last_name"]
        patient.first_name = first_name

    if request.get_json()["last_name"] is not None:
        last_name = request.get_json()["last_name"]
        patient.last_name = last_name

    t_value = patient.t_marker
    n_value = patient.n_marker
    m_value = patient.m_marker
    old_cancer_stage = patient.cancer_stage

    if request.get_json()["smoking_status"] is not None:
        patient.smoking_status = request.get_json()["smoking_status"]

    if request.get_json()["lung_history"] is not None:
        patient.family_history = request.get_json()["lung_history"]

    if request.get_json()["family_history"] is not None:
        patient.family_history = request.get_json()["family_history"]

    if request.get_json()["t_value"] is not None:
        t_value =  request.get_json()["t_value"]
        patient.t_marker = request.get_json()["t_value"]
        changed = True
    if request.get_json()["n_value"] is not None:
        n_value = request.get_json()["n_value"]
        patient.n_marker = n_value
        changed = True

    if request.get_json()["m_value"] is not None:
        m_value = request.get_json()["m_value"]
        patient.m_marker = m_value
        changed = True
```

```
    if changed is True:
        new_patient_history = HistoryPatient(
                    t_value,
                    n_value,
                    m_value,
                    old_cancer_stage,
                    patient_id
                )
        db.session.flush()
        db.session.commit(new_patient_history)

    cancer_stage = calculate_cancer_stage(t_value, n_value, m_value)
    patient.cancer_stage = cancer_stage["Cancer stage"]

    db.session.flush()
    db.session.commit()

    result = Patient.query.filter_by(patient_id=current_user.patient_id).first()

    return jsonify({"status code": status_code, "response": result})
```

Programski kod 7.23. Prikaz funkcije za uređivanje pacijenta

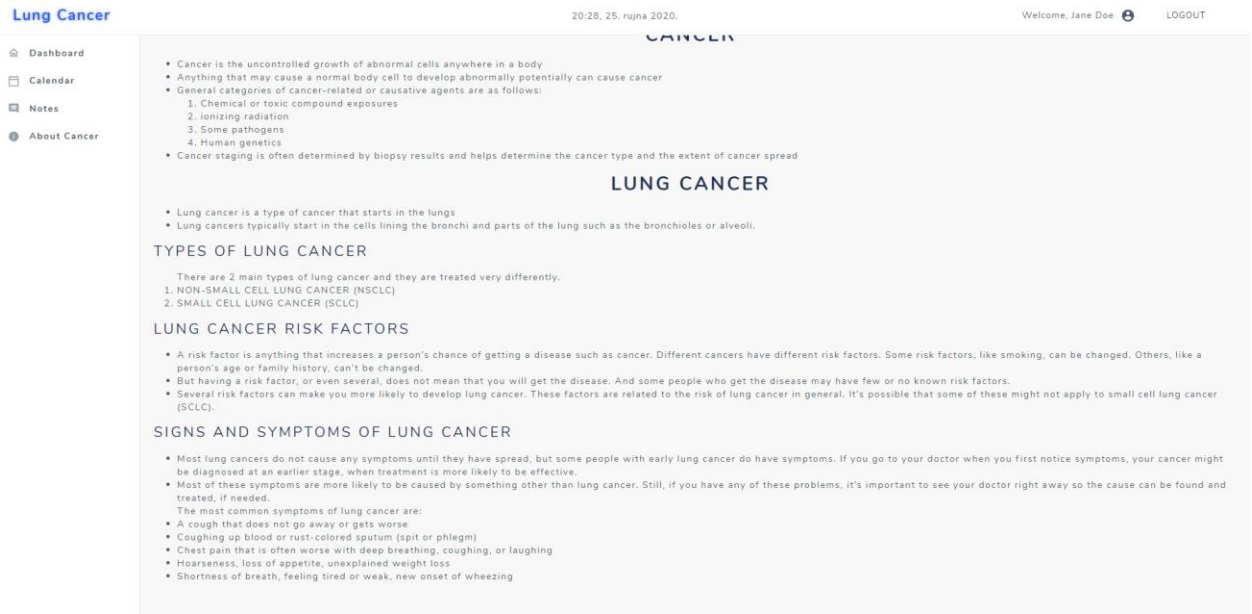
Pošto smo validaciju podataka objasnili, na prošlim rutama i funkcijama nećemo se na tome zadržavati u ovoj. Primijetimo da preuzimamo podatke iz forme koju korisnik mijenja na strani klijenta, te se provjerava da li su uneseni podatci vezani uz stadij rak pluća ili su mijenjani samo korisnički podatci vezani uz račun. Kada su podatci povezani sa stadijem uneseni u sustav, moramo izračunati novi stadij raka i spremiti prethodne podatke u tablicu prošlih stanja pacijenata. Strani klijenta vraćamo nove informacije o pacijentu u JSON formatu, te status kod prikladan odrađenoj radnji.

Slika 7.24 prikazuje izgled što bi pacijent vidio kada se prijavi u sustav sa svojim računom na strani klijenta. S lijeve strane se nalaze izbornici kojima može pristupiti, i posjetiti te rute, dok ostale informacije nema pravo mijenjati vezane uz stadij raka, medicinske dokumente ili lijekove koje koristi.



Slika 7.24. Prikaz korisnika prijavljenog kao pacijent na strain klijenta

Također na Slici 7.25 vidimo edukacijske sadržaje općenito o raku, ali isto tako o raku pluća. Edukacijskom sadržaju mogu pristupiti pacijenti odlaskom na */about*, koji se na strani klijenta prikazuje kao gumb *About*. Informacije su zamišljene da prate trenutno stanje u području i pacijentima omoguće točne i filtrirane informacije kako bi im olakšalo suočavanje s rakom.



Slika 7.25. Prikaz edukacijskih sadržaja unutar sustava na strani klijenta

7.6 Analiza rješenja i primjene funkcijskog programiranja

Iako smo u ovom poglavlju opisali programski kod i pokazali funkcije koje su nam potrebne za rad važno nam je pokazati da te funkcije i rute stvarno rade kako smo zamislili. Također pokušat ćemo prikazati da smo se držali načela funkcijskog programiranja, korištenjem funkcija kao građana prvog reda.

U programu jedini način mijenjanja vrijednosti je kroz funkcije, ne postoje globalne varijable osim postavljenih vrijednosti kod konfiguracije servisa i usluga, ali i za njih su omogućene promijene vrijednosti preko funkcija, tako da smo se pokušavali pridržavati pravila nepromjenjivosti.

Jedna od stvari koje su dosta česte u radu na web servisima je obrada, unos i mijenjanje podataka unutar baze podataka. Pobrinuli smo se da sve izmjene unutar baze podataka budu odrađene unutar funkcija te da nema vanjskih promjena stanja.

Pri radu nismo se u cijelosti pridržavali pravila funkcijskog programiranja niti je to bilo moguće, jer ne radimo u jeziku koji je namijenjen čistom funkcijskom programiranju. Zbog same čitljivosti i lakšeg ispravljanja grešaka koriste se petlje umjesto rekurzija za rad s korisnicima i podatcima. Rekurzija nije najbolje rješenje u slučaju velikog broja korisnika, zbog problematike spomenute u prethodnom poglavlju. U slučaju da u sustavu bude velika količina korisnika, a pri provjeri baze podataka kroz koju prolazili rekurzijom, dogodi se greška problemi koji bi nastali bili bi puno ozbiljniji nego problem korištenja petlje umjesto rekurzije.

Stvari koje se smatraju nepotrebnim u funkcijskom programiranju, su ispisi (eng. *Print())* u programskom kodu. U dogovoru s kolegicom Ivezić odgovori na pojedine rute su predefinirani, te se usklađeni odgovori i zahtjevi koje ćemo primati.

Ovaj diplomski rad pokazao je da funkcijsko programiranje ne mora biti fokus pri radu, ali da može poboljšati učinkovitost samog koda. Olakšati pregled i snalaženje samim time što će učiniti odgovore na iste zahtjeve predvidljivim što čini rad sa sustavom na strani klijenta, a i na strani poslužitelja puno jednostavnijim.

U tablici 7.26 prikazat ćemo primjer na par odgovora strane poslužitelja na zahtjeve za određenim funkcijama i rutama. Zbog veličine i količine ruta ne bi bilo moguće prikazati sve, ali se mogu lagano provjeriti pri pokretanju aplikacije. Može se primijetiti da svi odgovori imaju zadani format, te pri svakom upitu za iste predane parametre i iste podatke u bazi se dobiju iste vrijednosti.

Tablica 7.26. Prikaz odgovora na strain poslužitlja za određene funkcije

Ruta	Funkcija	Predano	Odgovor

/login	Login()	Ispravni: e-pošta ,lozinka	Status_code: 200
			[
			"user_data",
			{
			"email": "xaw41122@eoopy.com",
			"first_name": "Mato",
			"last_name": "Marulic",
			"role": "Doctor",
			"user_id": 1
			}
			]

/login	Login()	Neisravni: e-pošta ,lozinka	Status_code: 401 Incorrect username or password

/add_patient	New_patient()	Ispravni korisnički podatci	[
			"Patient succesfuly created",
			200,
			{
			"response": {
			"cancer_stage": {
			"Cancer stage": "IVA",
			"M": "M1a",
			"N": "N1",
			"T": "T2a"
			},
			"cancer_type": "NSCLC",
			"created:": "Sat, 20 Sep 2020 18:32:53 GMT",
			"doctor_id": 1,
			"doctor_name": "Mato Marulic",
			"email:": "Pdiplomski@gmail.com",
			"family_history": "Pusaci",
			"first_name:": "Testni",
			"gender": "female",
			"last_name:": "Pacijent",
			"lung_disease_history": false,
			"m_value": "M1a",
			"n_value": "N1",
			"role": "Patients",
			"smoking_status": false,
			"t_value": "T2a"
			}
			}
			]

/add_patient	New_patient()	Isti mail kao u prethodnom primjeru	[
			"Error",
			400,
			{
			"error": "Email is alredy being used"
			}
			]

/patients	Patient_identify()	Uneseni pacijent s appointmentom	{
			"result": [
			{
			"email:": "Pdiplomski@gmail.com",
			"first_name:": "Testni",
			"last_appointment": "Sun, 27 Sep 2020 20:41:30 GMT",
			"last_name:": "Pacijent",
			"next_appointment": "Sun, 27 Sep 2020 20:41:30 GMT"
			}
			],
			"status code": 200
			}

/patients	Patient_identify()	Nema appointment	[ ]
/list_appointemts	List_appointments()	Nema appointmenta	[ ]
Izgled doktor objekta koji se dohvaća iz baze			Doctor(id=1, first_name='Mato', last_name='Marulic', email='xaw41122@eoopy.com', password='$2b$12$6bMW.1GLljkdC8HIcdvufOCTrJ5eEbHt7iebHicTXa/dFnSr6bklW', created=datetime.datetime(2020, 9, 20, 19, 45, 2, 899529, tzinfo=psycopg2.tz.FixedOffsetTimezone(offset=120, name=None)), image_file='default.jpg', role='Doctor')
Izgled pacijent objekta koji se dohvaća iz baze			Patient(id=1, first_name='Testni', last_name='Pacijent', email='Pdiplomski@gmail.com', gender='female', smokin_status='false', lung_history='false', family_history='Pusaci', cancer_type='NSCLC', t_marker='T2a', n_marker='N1', m_marker='M1a', password='$2b$12$cWjwToZYYZmHGrD4SJM/8eksbglUqF3T/.aTb6pORcC7ZnWeFgBC.', created=datetime.datetime(2020, 9, 20, 20, 32, 53, 972544, tzinfo=psycopg2.tz.FixedOffsetTimezone(offset=120, name=None)), image_file='default.jpg', role='Patient')

/login (ulazak u sustav s gore kreiranim pacijentom)		Login()	[
			"user_data",
			{
			"email": "Pdiplomski@gmail.com",
			"first_name": "Testni",
			"last_name": "Pacijent",
			"role": "Patient",
			"user_id": 1
			}
			]

8. ZAKLJUČAK

Izrada ovog diplomskog rada ukazuje na potrebu za specijaliziranim i personaliziranim sustavima koji uz primjenu računalne tehnologije i pametnih programskih rješenja olakšavaju pacijentima i liječnicima praćenje i liječenje bolesti. Ciljani korisnici ovakvih sustava su bolnice, liječnici i pacijenti koji svakodnevno vode bitke za život. Sustav ima cilj poboljšati komunikaciju, olakšati praćenje oboljelih pacijenata, te omogućiti da pacijenti imaju sustav prilagođen njihovim potrebama kako bi se što brže oporavili. Prije početka rada na programskom rješenju, razrađena je logika, funkcionalnosti i zahtjevi koje bi web sustav trebao moći obraditi na strani poslužitelja i pripremiti podatke za prikaz na strani klijenta.

Razvijeni web sustav prilagođen je za rad s pacijentima i liječnicima, gdje svaka korisnička grupa ima svoj pogled. Korištenje funkcijskog programiranja u ovakvom tipu sustava predstavlja izazov, ali dobro iskustvo i priliku za učenje. Funkcijsko programiranje omogućava pregledan i intuitivan način kontrole podataka. Pisane su čiste i nepromjenjive funkcije koliko su platforma i zahtjevi sa strane klijenta to omogućavali. Liječnici imaju mogućnost registracije, prijave, dodavanja pacijenata u sustav, prikaz informacija o pacijentima, uređivanje terapije, pregledavanje nalaza, zakazivanja termina, sučelja za bilješke i uređivanje vlastitog profila. Pacijenti u svom pogledu imaju mogućnosti prijave, pregled vlastitog profila, prikaz kalendara s terminima, sučelje za bilješke, edukacijske sadržaje o raku i mogućnost preuzimanja digitaliziranih nalaza.

Sustav bi se mogao unaprijediti na mnoge načine, zbog opširne tematike i složenosti problema koji obuhvaća. Unaprjeđenje komunikacije između pacijenata, dodavanje računalnog učenja, poboljšanja funkcionalnost i povećavanjem točnosti u detekciji raka pluća samo su neke od mnogih mogućnosti koje bi se mogle poboljšati ili dodati.

LITERATURA

[1] Rak pluća, http://hlpr.hr/rak/vijest/rak-pluca, Pristupljeno: 9.07.2020.

[2] Medicinski priručnik za pacijente, Rak pluća,http://www.msd-prirucnici.placebo.hr/msd-za-pacijente/bolesti-pluca-i-disnih-putova/rak-pluca, Pristupljeno: 9.07.2020

[3] Onkologija, Rak pluća, http://www.onkologija.hr/rak-pluca/, Pristupljeno: 9.07.2020

[4] Lung Cancer Detection Using Image Segmentation by Means of Various Evolutinary Algorithms, https://www.hindawi.com/journals/cmmm/2019/4909846/, Pristupljeno: 10.07.2020

[5] Moovcare, https://www.moovcare.com/, Pristupljeno: 10.07.2020

[6] CancerAid, https://www.canceraid.com/, Pristupljeno: 10.07.2020

[7] Science Direction: Initial Evaluation of Patient With Lung Cancer: Symptoms,Signs,Laboratory Tests and Paraneoplastic Syndromes, https://www.sciencedirect.com/science/article/abs/pii/S0012369215329871, Pristupljeno: 12.07.2020

[8] Medicinski priručnik dijagnostika i terapija, Karcinom pluća, http://www.msd-prirucnici.placebo.hr/msd-prirucnik/pulmologija/tumori-pluca/karcinom-pluca, Pristupljeno: 12.07.2020

[9] Onkologija Rak pluća-ciljano biološko liječenje,http://www.onkologija.hr/rak-pluca/rak-pluca-lijecenje/rak-pluca-ciljano-biolosko-lijecenje/, Pristupljeno: 12.07.2020

[10] Lung Cancer and Key Statistics, https://www.cancer.org/content/dam/CRC/PDF/Public/8703.00.pdf, Pristupljeno:12.07.2020

[11] Onkologija Rak pluća – radioterapija,http://www.onkologija.hr/rak-pluca/rak-pluca-lijecenje/rak-pluca-radioterapija/, Pristupljeno: 12.07.2020

[12] Onkologija Rak pluća – potporno liječenje,http://www.onkologija.hr/rak-pluca/rak-pluca-lijecenje/rak-pluca-potporno-lijecenje/ , Pristupljeno: 12.07.2020

[13] American Joint Committee On Cancer, AJCC CANCER STAGING MANUAL 7th edition, Springer, 2015.

[14] K.L.R , Lung Cancer: Treatment and Research(Cancer Treatment and Research), Springer, 2016

[15] World Health Organization, WHO report on cancer: setting priorities, investing wisely and providing care for all, https://www.who.int/publications/i/item/who-report-on-cancer-setting-priorities-investing-wisely-and-providing-care-for-all, pristupljeno: 10.9.2020.

[16] TowardsDataScience, 10 In-Demand programming languages in 2020, https://towardsdatascience.com/top-10-in-demand-programming-languages-to-learn-in-2020-4462eb7d8d3e , Pristupljeno: 20.08.2020

[17] Python, https://www.python.org/ , Pristupljeno: 22.8.2020

[18] Flask, https://flask.palletsprojects.com/en/1.1.x/ , Pristupljeno: 22.8 .2020

[19] PostgreSql, https://www.postgresqltutorial.com/postgresql-vs-mysql/ , Pristupljeno: 24.8.2020

[20] Fer-Modularizacija i objektno usmjerena arhitektura 2. Dio, http://www.zemris.fer.hr/predmeti/opp/Notes/OO_2.ppt , Pristupljeno: 24.8.2020

[21] Toptal, Client side vs server side rendering https://www.toptal.com/front-end/client-side-vs-server-side-pre-rendering, Pristupljeno: 25.8.2020

[22] K. Konshin ,Next.js Quick Start Guide, Server side done right, , Packt , Srpanj,2018

[23] E. Chou ,Mastering Python Networking Second Edition, Lipanj,2017

[24] JSON, https://www.json.org/json-en.html, Pristupljeno: 27.8.2020

[25] Hrvatski matematički elektronički časopis, funkcijsko programiranje lamba računa kao osnovice funkcijskog programiranja, http://e.math.hr/category/klju-ne-rije-i/funkcijsko-programiranje, Pristupljeno: 5.09.2020

[26] Python,Functional,https://docs.python.org/3/howto/functional.html, Pristupljeno: 26.09.2020

[27] S. Lott, Functional Python Programming, Packt, 2015

[28] M,Anaya, Clean Code in Python, Packt ,2018

[29] B. Allbee Hands-On Software Engineering with Python, Packt, 2018

[30] Comparison of machine learning methods for classifying mediastinal lymph node metastasis of non-small cell lung cancer from F-FDG PET/CT images, https://link.springer.com/article/10.1186/s13550-017-0260-9 , Pristupljeno: 29.8.2020

[31] M. Ivezić,Programsko rješenje web sustava na strani klijenta za personalizirano praćenje pacijenata oboljelih od zloćudnih bolestiDiplomski rad, Fakultet elektrotehnike, računarstva i informacijskih tehnologija, 2020.

[32] http://www.msd-prirucnici.placebo.hr/msd-prirucnik/pulmologija/tumori-pluca/karcinom-pluca , Pristupljeno: 2.9.2020

SAŽETAK

U ovom diplomskom radu cilj je izrada web aplikacije na strani poslužitelja korištenjem programske paradigme funkcijskog programiranja za pomoć pacijentima oboljelim od raka pluća. Pacijentima se omogućuje kvalitetniji način života, pospješuje se oporavak detaljnim praćenjem i olakšava komunikacija s liječnikom. Na temelju istraživanja tržišta i razvijanja sustava za poboljšanje kvalitete života oboljelih pacijenata, primjećuje se potreba za sustavima koji bi pomogli rizičnim skupinama i olakšali im svakodnevni život pomoću tehnologije koja je svuda dostupna oko njih. Razvijena strana poslužitelja temelji se na obradi podataka poslanih sa strane klijenta, vraćanju obrađenih podataka spremnih za prikaz korisnicima koji mogu biti liječnici ili pacijenti. Poslužiteljska strana pri obradi podataka koristi se paradigmama funkcijskog programiranja.

Ključne riječi: funkcijsko programiranje, Python, rak pluća, strana poslužitelja, web sustav.

ABSTRACT

In this thesis, the goal is to create web application on the server-side using programmatic paradigms of functional programming to help patients with lung cancer. They provide patients with a better quality of life, facilitate recovery through detailed monitoring, and facilitate communication with the doctor. Based on market research and the development of systems to improve the quality of life of sick patients, the need for systems that can help at-risk groups and make everyday life easier is applied using technologies that are available everywhere around them. The developed side of the server is based on the processing of data sent by the client, the return of processed data ready for display by users who may be doctors or patients. The server side uses functional programming paradigms in the data processing.

Keyword: functional programming, lung cancer, personalized medicine ,web systems,

ŽIVOTOPIS

Mato Antunović rođen 27.07.1994 godine u Bochumu, Njemačka. Odrastao u Čepinu i završava matematičko prirodoslovnu gimnaziju u Osijeku. Upisao je Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku 2014. godine odabravši smjer računarstvo. Na trećoj godini studija zapošljava se u Kod Savjetovanja gdje radi do danas.

_____

Mato Antunović

PRILOZI

Prilog 1: Diplomski rad u .pdf formatu

Prilog 2: Diplomski rad u .docx formatu

Prilog 3: Programski kod aplikacije u .zip formatu